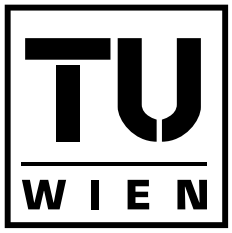

TECHNISCHE UNIVERSITÄT WIEN

VIENNA UNIVERSITY OF TECHNOLOGY

D I P L O M A R B E I T

Large Vocabulary Continuous Speech Recognition Systems and Maximum Mutual Information Estimation

Ausgeführt am Institut für

Institut für Statistik und Wahrscheinlichkeitstheorie
der Technischen Universität Wien

unter der Anleitung von
Univ.Prof.Dipl.-Ing.Dr.techn. Friedrich Leisch

durch

Markus Cozowicz
Name

Bleichergasse 1/5, A-1090 Wien
Anschrift

August 15, 2006 Unterschrift (Student)

Contents

Acknowledgments

I want to thank Sail Labs Technology AG for sponsoring this thesis and providing machines and corpora for experiments. Thanks goes to Andreas Türk for providing his in-depth knowledge of the HTK toolkit. Many thanks to Gerhard Backfried and Norbert Pfanner for supporting the Sail Labs speech recognizer. I would like to thank Jürgen Riedler for sharing his knowledge on the arabic language. Finally, thanks for guidance and constructive criticism to Prof. Friedrich Leisch.

Abstract

German

Diese Arbeit gibt eine allgemeine Einführung in den Bereich der automatisierten Spracherkennung mit Hilfe von Hidden Markov Modellen (HMM). Es wurde eine vollständige Trainingsumgebung von Sprachmodellen inklusive Erzeugung von Mix Modellen unter Verwendung des Hidden-Markov-Toolkit (HTK) und eines Spracherkenners von Sail Labs' erstellt. Um die Erkennungsrate zu erhöhen, wurde Maximum Mutual Information (MMI) Parameterschätzung implementiert. Ein 93h umfassender arabischer Broadcast News Korpus wurde für die Experimente verwendet. Eine Verbesserung der Erkennungsrate durch MMI am verwendeten Korpus konnte nicht festgestellt werden, es wird aber vermutet, dass die nötige Modell Umwandlung um HTK trainierte Modelle in Sail Labs' Spracherkenner zu verwenden, dafür verantwortlich ist.

English

This thesis presents a general introduction to automatic speech recognition based on Hidden Markov models (HMM). Using the Hidden-Markov-Toolkit (HTK) and Sail Labs' speech recognizer a complete trainings environment including mixture model training was created. To improve accuracy Maximum Mutual Information (MMI) estimation was implemented. Experiments were carried out using an 93h Arabic broadcast news corpus. MMI could not improve the accuracy on the Arabic corpus, but it is presumed that model transformations needed for usage of HTK trained models in Sail Labs' speech recognizer are responsible.

Keywords: speech recognition, hidden Markov models, maximum mutual information training, discriminative training.

1 Motivation

The Sail Labs Technology AG created a near real time speech recognizer and trained acoustic models for various languages including English, German, French, Arabic and many more. Business demands constant improvement of the recognizer and models.

Supporting training of models using a public available toolkit and the increasing accuracy is such an important improvement. One of the standard methods to increase accuracy is model estimation using the Maximum Mutual Information (MMI) criteria.

This thesis describes the complete training process of statistical models for Sail Labs' speech recognizer. As the targeted toolkit does not support MMI estimation, it is extended. A detailed description of the implementation of MMI estimation is given. The accuracy of the current Arabic model is not as excellent as the English model and therefore experiments are carried out using the Arabic corpus.

2 Introduction

Automatic speech recognition has been researched for more than 30 years. Today a state of the art toolkit, namely the Hidden-Markov-Toolkit [1] (HTK), is available for the public. Based on this toolkit and the included HTK Book [7] a general introduction to speech recognition is given. To improve accuracy the toolkit is extended to support maximum mutual information estimation.

The speech recognition itself is performed using Sail Labs' [2] speech recognizer and therefore the interaction with HTK and associated tools are described. The automatic speech recognition task is commonly split into three parts [2].

- A **Front-End** transforming the speech signal into feature vectors containing spectral and/or temporal information. Common methods are fast Fourier transform (FFT), linear predictive coding (LPC) and Mel Frequency Cepstral Coefficients (MFCCs).
- An **Acoustic Unit Matching System** matches units of features. Units can be words or sub-words, such as phonemes or syllables. Based on the task (e.g. single digit or continuous speech recognition) the unit size is chosen. Continuous speech recognition typically uses triphones (a phoneme with a left and a right context).

[1] http://htk.eng.cam.ac.uk/
[2] http://www.sail-technology.com

- **Language Model:** Apart from basic acoustic information, knowledge about the grammatical structure of a language is used. The structure can either be obtained by specifying a formal grammar of the language or gathering statistics from big text corpora (e.g. word bi-grams which is the number of occurrences of word W_1 followed by word W_2).

2.1 Human Speech

When humans speak, air is pressed out of the lungs, passing the vocal cords and finally either the mouth or the nose. Different sounds are generated by varying all the previously mentioned parts, called the vocal tract.

The smallest linguistic unit is called a phoneme [6]. Each word can be represented by multiple pronunciations consisting of a sequence of phonemes. The International Phonetic Association (IPA) [1] provides a standardized alphabet. A list of phonemes for the Arabic language used in all experiments is given in Table 1.

Besides the IPA alphabet, the table also lists Sail Labs' (SL) internal phoneme representation derived from Buckwalter encoding and Arabic sample letters. To overcome HTK limitations certain phones are re-mapped and listed in the *HTK* column.

Furthermore the phonemes are categorized into consonants in Tables 2 and vowels in Table 3 categorized by linguistic criteria. v and u denote voiced and unvoiced. The Sail Labs' internal encoding is used for phonemes in Tables 2 and 3.

2.1.1 Place of Articulation

The different phonemes are formed by constriction of the vocal tract. This constriction can happen at different places. A subset used for the Arabic language is

- **bilabial**: The sound is formed by the closure of the lips.
- **labiodental**: The lower lip is pressed against the upper teeth to produce the sound.
- **interdental**: Interdental consonants are produced by placing the blade of the tongue against the upper incisors.
- **dental**: Dentals are articulated with either the lower or the upper teeth, or both, rather than with the gum ridge.
- **dentalveolar**: Dentalveolars are articulated with the flexible front part of the tongue.
- **alveolar**: The alveolar ridge, which is just behind the top front teeth is touched by the tongue.

SL	IPA	HTK	Arabic	SL	IPA	HTK	Arabic
A	/ʔ/, /aː/		ا	d	/d/		د
l	/l/		ل	$	/ʃ/	X	ش
Y	/ʔ/		ئ	2	/ʕ/	B	ع
n	/n/		ن	6	/ð/	C	ذ
s	/s/		س	?	/ʔ/	G	ع
H	/ħ/		ح	q	/q/		ق
b	/b/		ب	D	/ɖ/		ض
x	/χ/		خ	f	/f/		ف
t	/t/		ت	T	/ŧ/		ط
r	/r/		ر	g	/ʁ/		غ
k	/k/		ك	O	/θ/		ث
w	/w/, /uː/		و	S	/s̻/		ص
j	/ʒ/		ج	Z	/z̻/		ظ
m	/m/		م	J	/ʔ/		أ
z	/z/		ز	BRT			
h	/h/		ه	GRB			
y	/j/, /iː/		ي	sil			
W	/ʔ/		ؤ				

Table 1: Phoneme Alphabet

- **palatal**: When the tongue touches the middle part of the palate.
- **velar**: The back part of the palate is touched by the tongue.
- **uvular**: Uvulars are articulated with the back of the tongue against or near the uvula.
- **pharyngeal**: The root of the tongue is pressed against the pharynx.
- **glottal**: This sound is produced by using the glottis.

A complete list can be found in [25].

2.1.2 Manner of Articulation

Another classification criteria is the manner of articulation, or how the vocal tract constricts. Again, the subset used for the Arabic language is

		bi-labial	labio-dental	inter-dental	dental	dent-alveolar	al-veolar
plosive	v	b			d		D
	u				t		
fricative	v			6,Z		z	
	u		f	O		s	S
nasal		m				n	
trill							r
lateral							l

		palatal	velar	uvular	pharyngeal	glottal
plosive	v					J,W,Y
	u		k	q	2	?
fricative	v	j		g		
	u	$			H	h

		non-emphatic	emphatic
plosive	v	d,W,Y	D,2
	u	t,?	T,q
fricative	v	z,k,j,J	Z,g
	u	s,$,h	S,x,H

Table 2: Sail Labs encoded categorized consonants

- **plosive**: Plosives are produced by stopping the airflow in the vocal tract.
- **fricative**: The vocal tract is nearly closed to produce these phonemes.
- **nasal**: If the air leaves the vocal tract through the nose instead of the mouth.
- **trill**: The sound is produced by vibrations between the articulator and the place of articulation.
- **lateral**: Laterals are pronounced with an occlusion made somewhere along the axis of the tongue, while air from the lungs escapes at one side or both sides of the tongue.

Additional information can be found in [26].

	open	close
front	A	y
back		w

Table 3: Sail Labs encoded categorized vowels

2.2 Signal Transformation

The very rich information of the human speech found in the audio signal is split into various features. The Mel Frequency Cepstral Coefficients (MFCCs) are commonly used for speech recognition ([7]). The following description is based on [4]. To extract the features from the audio signal several steps have to be performed:

1. **Divide into frames**: The signal is divided into frames by applying an overlapping windowing function in fixed intervals (e.g. 10ms). Typically a Hamming window defined as

$$w(n) = \begin{cases} 0.54 - 0.46\cos\left(\frac{2\pi n}{M}\right), & 0 \leq n \leq M \\ 0, & \text{otherwise,} \end{cases}$$

   where $M + 1$ ist the window size and $w(n)$ are the coefficients, is used and removes edge effects at the start and end of the frame. Oppenheim & Schafer [3] give a description of commonly used window functions in signal processing. This process generates a Cepstral feature vector for each frame.

2. **Discrete Fourier Transform** for each frame.

3. **Log of amplitude spectrum**, discards the phase information but retains the amplitude information. According to [5] the amplitude is most important for speech perception.

4. **Mel-scaling and smoothing**: The spectrum is smoothed and more important areas are emphasized by transforming the data to the Mel-scale. Pitch is not perceived in a linear manner and therefore the Mel-scale is based on a mapping between actual frequency and perceived pitch. This scale is linear up to 1kHz and logarithmic above.

5. **Discrete Cosine Transform** (DCT): The components of the Mel-spectral vectors of each frame are highly correlated. To decorrelate the components the DCT [23] is applied.

For all experiments a 45 dimensional Cepstral feature vector is calculated for frames of 10ms. A more detailed description can be found in [2].

2.3 Modeling Phonemes

Based on MFCCs different approaches to model phonemes have been developed. Hidden Markov models are predominantly used for modeling phonemes. Such systems are IBM ([9], [10]), CMU ([12], [13]), Philips [11], BBN/LIMSI ([14], [15]) and HTK [16]. Recent development effort was put into discriminative training of HMMs using as Maximum Mutual Information (MMI) estimation (see chapter 3.4) and Minimum Classification Error (MCE) [19]. Chapter 3 gives a detailed description on HMMs. As an alternative neural networks are used in [17] or combinations of HMMs and neural networks [18].

3 Hidden Markov Model

This section describes the statistical model in use for speech recognition. The hidden Markov model (HMM) is based on the assumption that the speech signal can be well characterized as a parametric random process. The basic theory was published by Baum [8] in the late 1960s and a complete description can be found in [2].

3.1 Markov Model

A Markov model consists of a set of N distinct states. In Figure 1 a weather system with states sunny, rainy and stormy is presented. The system changes its current active state in regular intervals according to probabilities associated to each state. The time instants associated with the state changes are denoted as $t = 1, 2, ...$, the active state is denoted by r_t. The time period for the weather system is a single day. A complete probabilistic description of the system would require the active state and all the preceding states that lead to the current active state. In the case of a first order Markov chain it is assumed that the current state depends only on a single preceding state.

$$P[r_t = j|r_{t-1} = i, r_{t-2} = k, ...] = P[r_t = j|r_t{-}1 = i]$$

Therefore only state-transition probabilities a_{ij} involving two states need to be specified of the following form

$$a_{ij} = P[r_t = j|r_{t-1} = i], \qquad 1 \leq i, j \leq N$$

with these properties

$$a_{ij} \geq 0 \qquad \forall i, j$$
$$\sum_{j=1}^{N} a_{ij} = 1 \qquad \forall i$$

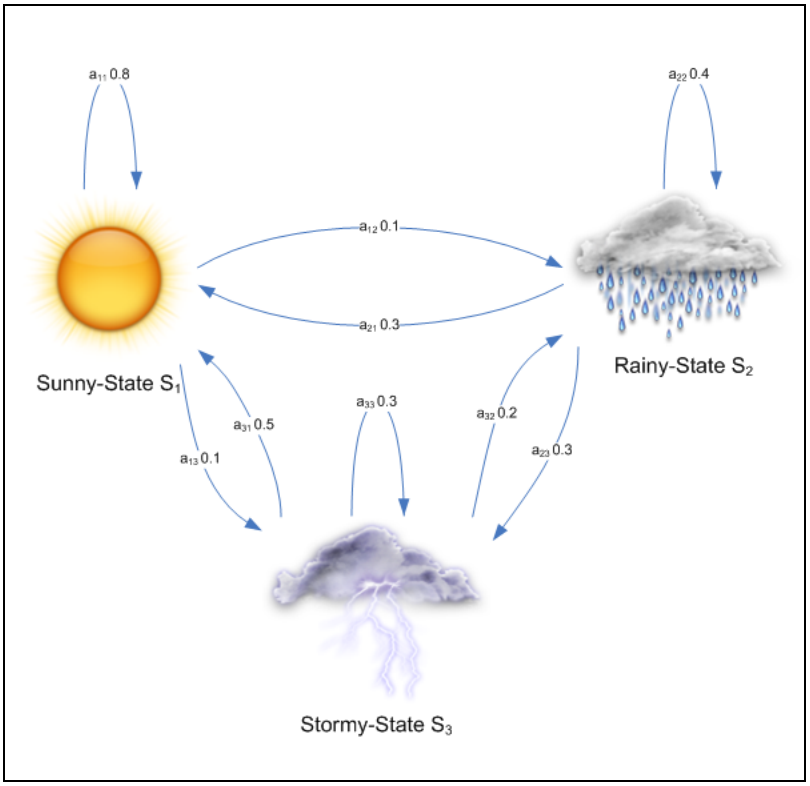

Figure 1: A Markov model of the weather

The transition probabilities of the weather example are found on top of the arrows between the states. Finally initial probabilities for the first state defined as

$$\pi_i \geq 0 \qquad 1 \leq i \leq N$$
$$\sum_{i=1}^{N} \pi_i = 1$$

are required.

The output of an *observable* Markov model are the visited states as time passes by. The states correspond directly to the observed events. A question answered by the model in figure 1 could be:

What is the probability of this week's forecast

Sunny-Sunny-Rainy-Stormy-Rainy

according to the model?

The observation sequence, **O**, is defined as

$$\begin{array}{rlccccc} \mathbf{O} & = (& \mathit{Sunny}, & \mathit{Rainy}, & \mathit{Rainy}, & \mathit{Stormy}, & \mathit{Rainy} &) \\ & = (& S_1, & S_2, & S_2, & S_3, & S_2 &) \\ \mathbf{t} & = (& 1, & 2, & 3, & 4, & 5 &) \end{array}$$

corresponding to this week's forecast. The probability of $P(\mathbf{O}|Model)$ is

$$\begin{aligned} P(\mathbf{O}|Model) &= P(S_1, S_2, S_2, S_3, S_2|Model) \\ &= P[S_1]P[S_2|S_1]P[S_2|S_2]P[S_3|S_2]P[S_2|S_3] \\ &= \pi_{S_1} a_{12} a_{22} a_{23} a_{32} \\ &= (1.0)(0.1)(0.3)(0.3)(0.2) \\ &= 0.0018, \end{aligned}$$

that is, the probability π_{S_1} of being in state S_1 at time 1 multiplied by a_{11} accounting for the transition from state S_1 to state S_2 and so on. π_{S_1} is 1 because $t = 1$ is today.

3.2 Hidden Markov Model

The association between events and states in Markov models are deterministic. Markov models are extended so that the observation, the speech signal, is a probabilistic function of the state. The result is a double stochastic process, where the underlying process, the state transitions, cannot be observed directly, but through another stochastic process that produces the sequence of observations.

A hidden Markov model consists of

1. a distinct number of states **N**;
2. a state-transition probability distribution $A = \{a_{ij}\}$ as specified for Markov models;
3. M observation symbols and a subset $V = \{v_1, v_2, ..., v_M\}$ per state;
4. an observation symbol probability distribution, $B = \{b_j(k)\}$, in which

$$b_j(k) = P[o_t = v_k|r_t = j], \quad 1 \leq k \leq M,$$

defines the symbol distribution in state $j, j = 1, 2, ...N$;

5. an initial state distribution $\Pi = \{\pi_i\}$, in which

$$\pi_i = P[r_1 = i], \quad 1 \leq i \leq N.$$

For convenience the compact notation $\lambda = (A, B, \Pi)$ is used. Applying the above to continuous speech recognition, a single HMM models units of features (e.g. phonemes or syllables).

1. States represent the sequence of audio.

2. The transition probability distribution defines a graph by which the HMM might be passed through.

$$\begin{aligned} a_{ij} &= 0 \quad \forall i > j, \\ a_{ij} &> 0 \quad \forall i \leq j + 1, \\ & \quad 1 \leq i, j \leq N \end{aligned}$$

   The graph is normally forward only and provides the ability to skip the next state as presented in Figure 2.

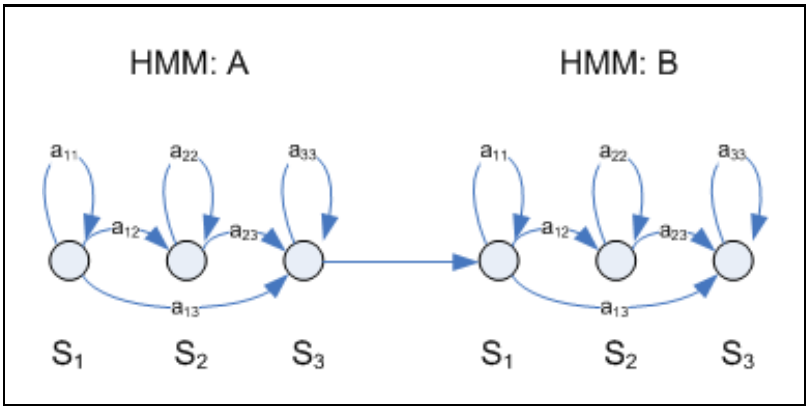

Figure 2: Two HMMs with three states for the phone A and B

   The corresponding transition matrix for phone A is given by

	S_1	S_2	S_3
S_1	a_{11}	a_{12}	a_{13}
S_2	0	a_{22}	a_{23}
S_3	0	0	a_{33}

   During training and recognition consecutive HMMs for phonemes are connected at there start/end states.

3. The observation symbols are signal features as presented in chapter 2.2.

3.3 Three Basic Problems

HMMs introduce three basic problems: evaluating the

1. probability (or likelihood) of a sequence of observations;
2. best sequence of model states;
3. model parameters to fit to the observations.

3.3.1 Probability of an Observation Sequence

The observation sequence **O** and the state sequence **r** are vectors of length T:

$$\mathbf{r} = (r_1, r_2, ...r_T)$$

$$\mathbf{O} = (o_1, o_2, ...o_T)$$

The probability of an observation sequence given the model λ is defined as

$$P(\mathbf{O}|\lambda) = \sum_Q P(\mathbf{O}|\mathbf{r}, \lambda)P(\mathbf{r}|\lambda)$$

which is the sum over the joint probability of state sequence **r** given the model λ and the observation sequence **O** given the state sequence **r** and λ for all possible state sequences Q.

The probability of the state sequence **r** is given by

$$P(\mathbf{r}|\lambda) = \pi_{r_1} a_{r_1 r_2} a_{r_2 r_3} ... a_{r_{T-1} r_T}$$

which is the product of the initial probability π_{r_1} of being in state r_1 and the product of transition probabilities $a_{r_i r_j}$ according to the state sequence **r**. The beginning of the path is modeled by π_{r_1} and for each further transition from r_i to r_j, transition probabilities $a_{r_i r_j}$ must be factored in. Given a state sequence **r** the probability of the observation sequence **O** is

$$P(\mathbf{O}|\mathbf{r}, \lambda) = b_{r_1}(o_1) b_{r_2}(o_2) ... b_{r_T}(o_T),$$

the product of observation symbol probabilities for each state r_i. Therefore

$$\begin{aligned} P(\mathbf{O}|\lambda) &= \sum_Q P(\mathbf{O}|\mathbf{r}, \lambda)P(\mathbf{r}|\lambda) \\ &= \sum_{r_1, r_2, ..., r_T} \pi_{r_1} b_{r_1}(o_1) a_{r_1 r_2} b_{r_2}(o_2) ... a_{r_{T-1} r_T} b_{r_T}(o_T) \end{aligned}$$

which describes the process as follows: Initially the state r_1 is selected with probability π_{r_1} emitting the symbol o_1 with probability $b_{r_1}(o_1)$. Then the

process continues from state r_1 to r_2 with probability $a_{r_1 r_2}$ and emitting symbol o_2 with probability $b_{r_2}(o_2)$. The process stops at state r_T. This process is computationally infeasible as it requires $2TN^T$ calculations. The Forward Procedure is more efficient.

The Forward Procedure defines the forward variable $\alpha_t(i)$ as

$$\alpha_t(i) = P(o_1, o_2, ..., o_t, r_t = i|\lambda) \tag{1}$$

that is, the probability of the partial observation, $o_1, o_2, ..., o_t$ (until time t) and state i at time t, given the model λ. $\alpha_{1(i)}$ at time 1 for state i is defined as

$$\alpha_1(i) = \pi_i b_i(o_1), \qquad 1 \leq i \leq N.$$

The remaining $\alpha_t(j)$ are recursively defined as:

$$\alpha_{t+1}(j) = \Big[\sum_{i=1}^{N} \alpha_t(i) a_{ij}\Big] b_j(o_{t+1}), \qquad \begin{array}{c} 1 \leq t \leq T-1 \\ 1 \leq j \leq N \end{array}$$

Instead of evaluating all possible paths, similar to first order Markov models, all previous paths $\alpha_t(i)$ are merged into $\alpha_{t+1}(j)$ weighted by the corresponding transition probabilities a_{ij}. Based on equation 1, one can easily see that

$$\begin{aligned} P(\mathbf{O}|\lambda) &= \sum_{i=1}^{N} P(o_1, o_2, ..., o_T, r_T = i|\lambda) \\ &= \sum_{i=1}^{N} \alpha_T(i). \end{aligned}$$

The computation requires only N^2T compared to $2TN^T$. The key difference between the straight forward definition and the Forward Procedure, is that all possible state sequences at time $t-1$ will merge into $\alpha_t(i)$ for $t > 1$.

The Backward Procedure defines the backward variable $\beta_t(i)$ in a very similar manner to $\alpha_t(i)$:

$$\beta_t(i) = P(o_{t+1}, o_{t+2}, ..., o_T|r_t = i, \lambda)$$

This is the probability of the partial observation sequence from $t+1$ to the end, given state i at time t and the model λ. As with $\alpha_t(i)$, $\beta_t(i)$ is defined inductively as:

$$\begin{aligned} \beta_T(i) &= 1 && 1 \leq i \leq N \\ \beta_t(i) &= \sum_{i=1}^{N} a_{ij} b_j(o_{t+1}) \beta_{t+1}(j) && t = T-1, T-2, ..., 1 \end{aligned}$$

As with the α variable, succeeding paths are merged into $\beta_t(i)$. The following problems can now be solved by using the α and β variable.

3.3.2 Choosing the Optimal State Sequence - Viterbi Algorithm

To find the optimal state sequence one has to define an optimality criterion. Choosing only the best state for each time t might not result in a valid path, as state transition probabilities might be zero. Thus one needs to find the best valid path $\mathbf{r} = (r_1, r_2, ..., r_T)$. This can be done by using the Viterbi Algorithm. A quantity for scoring a path is given by

$$\delta_t(i) = \max_{r_1, r_2, ..., r_{t-1}, r_t} P(r_1 r_2 ... r_{t-1}, r_t = i, o_1 o_2 ... o_t | \lambda)$$

Similar to the Forward Procedure, $\delta_{t+1} i$ is defined as

$$\begin{aligned} \delta_1(i) &= \pi_i b_i(o_1) \\ \delta_{t+1}(j) &= [\max_i \delta_t(i) a_{ij}] \cdot b_j(o_{t+1}). \end{aligned}$$

Each $\delta_{t+1}(i)$ is formed by re-using the maximum of $\delta_t(i)$ weighted by the corresponding transition probability a_{ij}. To actually determine the optimal state sequence, one has to keep track of each $\max \delta_t(i)$ and collect $r_{\arg\max[\delta_t(i)]}$ using backtracking.

3.3.3 Estimation of Model Parameters

A far more difficult problem arises when it comes to estimation of model parameters. There is no closed form solution to maximize the probability of the observation sequence given a model, but an iterative procedure commonly known as the Baum-Welch Method or EM (expectation-maximization) method ([27], [28], [29], [30], [31]) exists.

At first, two probabilities need to be defined:

$$\begin{aligned} \gamma_t(i) &= P(r_t = i | \mathbf{O}, \lambda) \\ \xi_t(i, j) &= P(r_t = i, r_{t+1} = j | \mathbf{O}, \lambda) \end{aligned}$$

$\gamma_t(i)$ is the probability of being in state r_i at time t and $\xi_t(i, j)$ is the probability of being in state r_i at time t and in state r_j at time $t + 1$. $\gamma_t(i)$ can be expressed through $\alpha_t(i)$ and $\beta_t(i)$ as produced by the Forward and Backward Procedures:

$$
\begin{aligned}
\gamma_t(i) &= P(r_t = i|\mathbf{O}, \lambda) \\
&= \frac{P(r_t = i, \mathbf{O}|\lambda)}{P(\mathbf{O}|\lambda)} \\
&= \frac{P(r_t = i, \mathbf{O}|\lambda)}{\sum_{i=1}^{N} P(\mathbf{O}, r_t = i|\lambda)} \\
&= \frac{P(o_1, o_2, ..., o_t, r_t = i|\lambda) P(o_t, o_{t+1}, ..., o_T|r_t = i, \lambda)}{\sum_{i=1}^{N} P(\mathbf{O}, r_t = i|\lambda)} \\
&= \frac{\alpha_t(i)\beta_t(i)}{\sum_{i=1}^{N} \alpha_t(i)\beta_t(i)}
\end{aligned}
$$

$\alpha_t(i)$ accounts for the partial observation sequence $o_1, o_2, ..., o_t$ and $\beta_t(i)$ for $o_t, o_{t+1}, ..., o_T$. $\xi_t(i, j)$ can be expressed in terms of α and β

$$
\begin{aligned}
\xi_t(i, j) &= P(r_t = i, r_{t+1=j}|\mathbf{O}, \lambda) \\
&= \frac{P(r_t = i, r_{t+1=j}|\mathbf{O}, \lambda)}{P(\mathbf{O}|\lambda)} \\
&= \frac{P(r_t = i, r_{t+1=j}|\mathbf{O}, \lambda)}{\sum_{i=1}^{N} P(\mathbf{O}, r_t = i|\lambda)} \\
&= \frac{P(o_1, o_2, ..., o_t, r_t = i|\lambda) a_{ij} b_j(o_{t+1}) P(o_{t+1}, o_t, ..., o_T|r_t = j, \lambda)}{\sum_{i=1}^{N} P(\mathbf{O}, r_t = i|\lambda)} \\
&= \frac{\alpha_t(i) a_{ij} b_j(o_{t+1}) \beta_{t+1}(j)}{\sum_{i=1}^{N} \alpha_t(i)\beta_t(i)}
\end{aligned}
$$

Comparing $\xi_t(i, j)$ and $\gamma_t(i)$, one can easily see that a specific path can be modeled by adding the required transition probability a_{ij} and the output probability $b_j(o_{t+1})$ to the product.

The re-estimation of parameters of an HMM are defined using $\gamma_t(i)$ and $\xi_t(i)$:

$$\begin{aligned}
\bar{\pi}_j &= \gamma_1(i) \\
\bar{a}_{ij} &= \frac{\sum_{t=1}^{T-1} \xi_t(i,j)}{\sum_{t=1}^{T-1} \gamma_t(i)} \\
\Gamma(k,t) &= \begin{cases} 1, & \text{if } v_k = o_t \\ 0, & \text{otherwise} \end{cases} \\
\bar{b}_j(k) &= \frac{\sum_{t=1}^{T-1} \Gamma(k,t)\gamma_t(j)}{\sum_{t=1}^{T-1} \gamma_t(j)}
\end{aligned}$$

Updating the model parameters λ as shown above is the maximization step of the EM algorithm. Baum and his colleagues [8] have proven that a re-estimated model $\bar{\lambda} = (\bar{A}, \bar{B}, \bar{\pi})$ defines either a critical point of the likelihood function or $\bar{\lambda}$ is more likely than λ.

3.3.4 Continuous Observation Densities in HMMs

Until now only discrete output symbols have been considered, but in speech recognition one has to deal with a continuous signal. According to [2] reestimation formulas exist if the density used to replace the observation symbol probabilities B, is a finite mixture of log-concave or elliptically symmetric densities. Usually Gaussian mixtures with M components are used to model the signal. Parameter estimation as implemented in HTK [7] starts with a single Gaussian per feature with mean vector μ and covariance matrix $\mathbf{U}$ and is split according to the following procedure: The weight c_{jk} of the j-th state and k-th mixture component is first halved and then the mixture is cloned. The two identical mean vectors are then perturbed by adding 0.2 standard deviations to one and subtracting the same amount from the other. In the second step, the mixture component with the largest weight is split as above. This is repeated until the required number of mixture components are obtained.

$$b_j(o) = \sum_{k=1}^{M} c_{jk} \mathcal{N}(\mathbf{o}, \mu_{jk}, \mathbf{U}_{jk}), \qquad 1 \leq j \leq N, 1 \leq k \leq M$$

As before, o is the observation, c_{jk} is the k-th mixture weight at state j and $\mathcal{N}$ a Gaussian distribution with mean vector μ_{jk} and covariance matrix $\mathbf{U}_{jk}$. To ensure consistent updating of the model, the mixture weights must adhere to stochastic constraints.

$$\sum_{k=1}^{M} c_{jk} = 1, \quad 1 \leq j \leq N$$
$$c_{jk} \geq 0, \quad 1 \leq j \leq N, 1 \leq k \leq M$$

For parameter estimation the EM algorithm is used. The expectation step calculates the α and β variable in the same manner by the Forward/Backward Algorithm for continuous observation densities as it was done for discrete observations. The maximization step estimates a_{ij} as previously presented, but the components of $b_j(k)$ are given by

$$\bar{c}_{jk} = \frac{\sum_{t=1}^{T} \gamma_t(j,k)}{\sum_{t=1}^{T}\sum_{k=1}^{M} \gamma_t(j,k)}$$

$$\bar{\mu}_{jk} = \frac{\sum_{t=1}^{T} \gamma_t(j,k) \cdot \mathbf{o}_t}{\sum_{t=1}^{T} \gamma_t(j,k)}$$

$$\bar{\mathbf{U}}_{jk} = \frac{\sum_{t=1}^{T} \gamma_t(j,k) \cdot (\mathbf{o}_t - \mu_{jk})(\mathbf{o}_t - \mu_{jk})'}{\sum_{t=1}^{T} \gamma_t(j,k)}$$

The mixture weight $\bar{c}_{jk}$ is the ratio of the expected number of times in state j using the k-th mixture and the expected total number of times in state j.

$$\gamma_t(j,k) = \left(\frac{\alpha_t(j)\beta_t(j)}{\sum_{j=1}^{N} \alpha_t(j)\beta_t(j)} \right) \left(\frac{c_{jk}\mathcal{N}(\mathbf{o}, \mu_{jk}, \mathbf{U}_{jk})}{\sum_{m=1}^{M} c_{jm}\mathcal{N}(\mathbf{o}, \mu_{jm}, \mathbf{U}_{jk})} \right)$$

$\gamma_t(j)$ is multiplied by the weighted output probability to accommodate for the mixture components resulting in $\gamma_t(j,k)$, that is the probability of being in state j using the k-th mixture component.

3.4 Maximum Mutual Information Estimation

3.4.1 Introduction

In general, maximum likelihood estimation (MLE) as presented in the previous chapter is used to train HMMs due to the fact that it is an optimal estimate (unbiased with minimum variance).

But MLE is based on the assumption that the observed speech is produced by the HMM. The following wrong assumptions are made:

- Output independence.
- Markov model, a state at time t depends only on states $t-1$.
- Continuous probability density.

In contrast to MLE, maximum mutual information (MMI) estimation relies less on model assumptions and was successfully used by [20], [21] and [22]. The basic idea of mutual information: how much information does one random variable provide about another one? In the case of speech recognition we want to know how much information the observed audio (O) holds about the origin words (S). By maximizing the mutual information between observed audio (O) and origin words (S), the knowledge of S is maximized. The mutual information I between O and S is given by

$$I(O,S) = \sum_{O,S} P(O,S) \log \frac{P(O,S)}{P(O)P(S)}$$

where $P(O,S)$ is the joint probability of the observed audio and the origin words and $P(O)$ and $P(S)$ are the corresponding marginal distributions. It is easy to see that if O and S are independent, that is $P(O,S) = P(O)P(S)$,

$$\log \frac{P(O)P(S)}{P(O)P(S)} = \log 1 = 0 \rightarrow I(O,S) = 0$$

An alternative representation based on entropy (H) is given by

$$\begin{aligned}
I(O,S) &= H(S) - H(O|S) \\
H(O|S) &= -\sum_{o \in O} \sum_{s \in S} P(O=o, S=s) \log P(O=o|S=s) \\
&= -\sum_{o \in O} \sum_{s \in S} P(O=o, S=s) \log \frac{P(O=o, S=s)}{P(S=s)} \\
&= -\sum_{o_1 \in O} \sum_{s \in S} P(O=o_1, S=s) \log \frac{P(O=o_1, S=s)}{\sum_{o_2 \in O} P(O=o_2, S=s)}
\end{aligned}$$

The entropy of the source $H(S)$ cannot be changed, therefore only $H(O|S)$ can be adapted to maximize $I(O,S)$. The MMI criteria is defined as

$$\begin{aligned} F_{\mathrm{MMI}}(\lambda) &= \sum_{r=1}^{R} log P(S_r|O_r,\lambda) \\ &= \sum_{r_1=1}^{R} log \frac{P(O_{r_1}|S_{r_1},\lambda)P(S_{r_1})}{\sum_{r_2=1}^{R} P(O_{r_2}|S_{r_2},\lambda)P(S_{r_2})} \end{aligned} \tag{2}$$

where R is the number of utterances.

$P(S)$ is the probability of the origin word sequence, commonly referred to as the language model score. The denominator term represents the sum of probabilities of all possible word sequences. To overcome the computational infeasibility it is approximated by the N-Best hypothesis or a word lattice produced by a recognition run.

3.4.2 Estimation of Model Parameters

To define the estimation formulas we need to specify two quantities

$$\begin{aligned} \gamma_{jm} &= \sum_{t=1}^{T} \gamma_t(j,m) \\ \theta_{jm} &= \sum_{t=1}^{T} \gamma_t(j,m) \cdot \mathbf{o}_t. \end{aligned}$$

γ_{jm} is the occupation count for state j using the k-th mixture component, θ_{jm} is the weighted sum of output symbols. Two different θ and γ are used for reestimation of μ_{jm} and σ_{jm}: $\theta_{jm}^{\mathrm{num}}$ correspond to the numerator term of (2) and θ^{den} to the denominator term. The numerator part is based on a single transcription using MLE, the denominator part uses a lattice including confusability of the speech recognizer. The new $\bar{\mu}_{jm}$ and $\bar{\sigma}_{jm}^2$ are given by

$$\begin{aligned} \bar{\mu}_{jm} &= \frac{\{\theta_{jm}^{\mathrm{num}}(\mathbf{O}) - \theta_{jm}^{\mathrm{den}}(\mathbf{O})\} + D\mu_{jm}}{\{\gamma_{jm}^{\mathrm{num}} - \gamma_{jm}^{\mathrm{den}}\} + D} \\ \bar{\sigma}_{jm}^2 &= \frac{\{\theta_{jm}^{\mathrm{num}}(\mathbf{O}^2) - \theta_{jm}^{\mathrm{den}}(\mathbf{O}^2)\} + D(\sigma_{jm}^2 + \mu_{jm}^2)}{\{\gamma_{jm}^{\mathrm{num}} - \gamma_{jm}^{\mathrm{den}}\} + D} - \hat{\mu}_{jm}^2 \end{aligned}$$

where D is set on a per Gaussian level, that is for every state j and every mixture component m. D is set so that all variances σ^2 are positive. A few useful definitions to shorten the transformation.

$$\begin{aligned}\Theta_\sigma &= \theta_{jm}^{\text{num}}(\mathbf{O}^2) - \theta_{jm}^{\text{den}}(\mathbf{O}^2)\\ \Theta_\mu &= \theta_{jm}^{\text{num}}(\mathbf{O}) - \theta_{jm}^{\text{den}}(\mathbf{O})\\ \Gamma &= \gamma_{jm}^{\text{num}} - \gamma_{jm}^{\text{den}}\end{aligned}$$

D is set twice the minimum given by

$$\begin{aligned}0 &= \frac{\Theta_\sigma + D(\sigma_{jm}^2 + \mu_{jm}^2)}{\Gamma + D} - \hat{\mu}_{jm}^2\\ &= \frac{\Theta_\sigma + D(\sigma_{jm}^2 + \mu_{jm}^2)}{\Gamma + D} - \Big(\frac{\Theta_\mu + D\mu_{jm}}{\Gamma + D}\Big)^2\\ &= \frac{(\Theta_\sigma + D(\sigma_{jm}^2 + \mu_{jm}^2))(\Gamma + D) - (\Theta_\mu + D\mu_{jm})^2}{(\Gamma + D)^2}\\ &= (\Theta_\sigma + D(\sigma_{jm}^2 + \mu_{jm}^2))(\Gamma + D) - (\Theta_\mu + D\mu_{jm})^2\\ &= D^2(\sigma_{jm}^2) + D(\Gamma(\sigma_{jm}^2 + \mu_{jm}^2) + \Theta_\sigma - 2\Theta_\mu\mu_{jm}) + \Theta_\sigma\Gamma - \Theta_\mu^2\end{aligned}$$

A full derivation is found in [24].

4 MMIE Implementation

4.1 Introduction

The HTK was used to train the large vocabulary continuous speech recognition system. In chapter 5 an overview of the trainings process is given and appendix A has a detailed description of all required steps.

At the time of writing only HTK 3.2.1 is available, which does not include MMIE. As the HTK already provides a comprehensive framework for HMM training, it is chosen as a base for MMIE implementation. This chapter focuses on the implementation of MMIE, based on the MLE implementation found in the HTK tool *HERest*. *HERest* is built from a multitude of sources files. The following list contains the source files modified or extended for MMIE:

- *HTKTools/HERest.c*: Main entry point of *HERest*. It includes loading of data files and model update.
- *HTKLib/HFB.c*: Accumulation of α and β variable of the forward/backward algorithm for transcripts.

- *HTKLib/HNet.h*: Defines data structures holding lattices (see Figure 3 and 4).
- *HTKLib/HMMI.c*: Accumulation of α and β variable of the generalized forward/backward algorithm for lattices.

The original parameter estimation and statistic accumulation of the α and β variable according to the forward/backward algorithm is done using *HERest*. It can process transcripts and calculate statistics needed for MLE. MMIE needs statistics accumulated from lattices and therefore *HERest* was generalized to support these. A comparison of a transcript and an associated lattice generated by a recognition run is found in Figure 3.

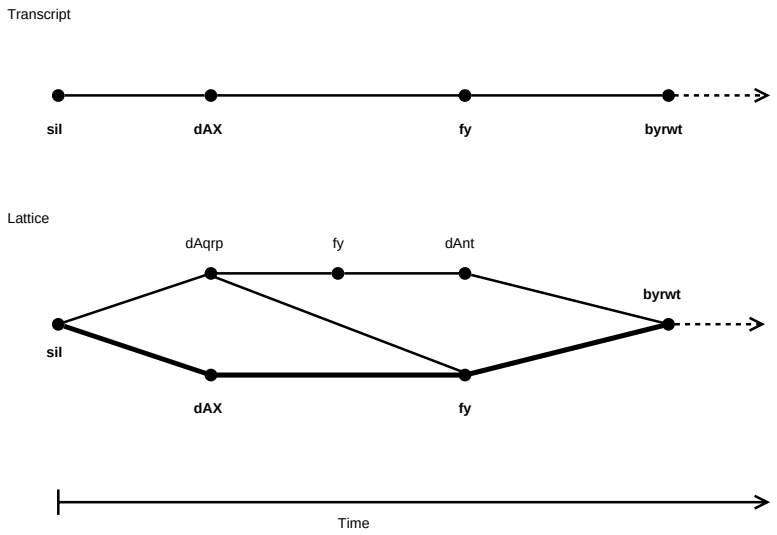

Figure 3: Transcript (MLE) vs. Lattice (MMIE)

The implementation of statistic accumulation for transcripts can be found in *HTKLib/HFB.c* and the actual parameter update is done in *HTKTools/HERest.c*. To fit into the current structure, *HTKLib/HMMI.c* contains statistic accumulation for lattices and *HTKTools/HERest.c* was extended with MMI based parameter re-estimation.

MLE statistic accumulation uses label files containing transcripts for the audio. Depending on the matching unit modeled by an HMM the transcripts are either word, monophone or triphone based. A sample of word level transcripts can be found in chapter 5.3.4. MMIE accumulation needs lattices from a recognition fitting with the matching units. Sail Labs' recognizer produces word lattices in HTK's standard lattice format (slf). The full process of generating the lattices and any post-processing required is described in chapter A.5.

The generated lattices can be read and held in memory with functions and data structures already provided by HTK. To accumulate statistics the right transcript level is needed and the lattice is expanded to match the HMMs by HTK's *ExpandWordNet* function. The function requires a dictionary containing each word with multiple associated pronunciations. To expand each word into its pronunciation, the lattice needs to contain a pronunciation ID per word. The returned *Network* structure found in *HTK-Lib/HNet.h* serves as the root for the expanded lattice.

The nodes in the expanded lattice are linked in a chain for fast unordered processing and each node contains an array of *NetLink* structures which link to the succeeding nodes. For MMIE additional information for each node during the forward/backward algorithm is needed, namely

- a list of succeeding and preceding nodes including language model scores for the transition,
- a flag if the model is active at time t,
- time based pruning information,
- a model ID q to fit the MLE implementation.

The *NetNode* structure representing each HMM was extended with *MMIInfo* structure holding the above information. Figure 4 shows the graph of the data structures in use for the first words of the already expanded lattice from Figure 3.

The *forwardlinks* and *backlinks* are single linked lists generated by recursively walking through the lattice. For now only a single initial and final HMM is allowed due to initialization restrictions.

The word lattice contains language model scores between each word. This score is used as a transition probability in the forward/backward algorithm. The language model score $P(W_2, W_1)$ for the word sequence W_1, W_2 is spread equally among the expanded monophone or triphone nodes.

4.2 Backward Procedure

HERest calculates first the β variable and in a following step the α variable. The MLE implementation assigns Q identifiers ordered by time to each HMM. This numbering is used to process the HMMs in the correct order and to prune the number of active HMMs at each time t. For more details on pruning see chapter 4.5.

Each HMM has an identifier q and N states with an associated transition probability matrix a. The first and last state are always non-emitting states. Due to the time-based numbering, successors and predecessors are referenced by $q + 1$ and $q - 1$ respectively. For the case of lattices, multiple successors

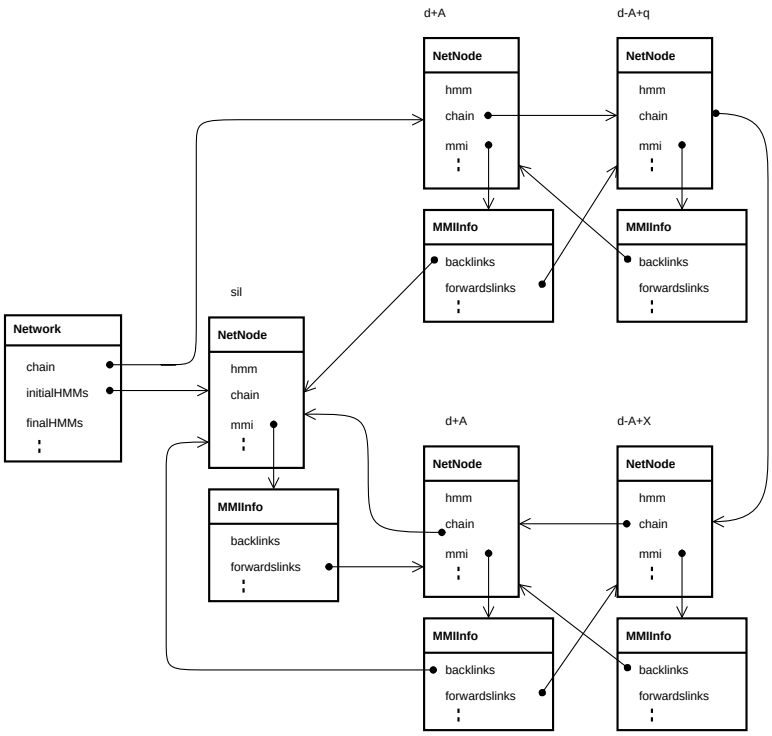

Figure 4: Data structures used for lattices

and predecessors may exist and therefore each HMM has an associated set F and B implemented by the single linked lists *forwardlinks* and *backlinks* found in *MMIInfo*. Each link holds the previously spread language model l. The backward step starts at time $t = T$ and progresses toward $t = 1$. Before β is evaluated, the output probabilities b for each active HMM and state are computed. β for the last state of each HMM active at time $t = T$ is calculated by

$$\begin{aligned} \beta_{T,q,N} &= 1 && \text{if } q = Q \\ \beta_{T,q,N} &= \beta_{T,q+1,N} \cdot a_{q+1,1,N} && \text{if } q \neq Q \end{aligned}$$

The generalized version is given by

$$\begin{aligned} \beta_{T,q,N} &= 1 && \text{if } |F| = 0 \\ \beta_{T,q,N} &= \sum_{f \in F_q} \beta_{T,f,N} \cdot a_{f,1,N} && \text{else} \end{aligned}$$

It should be noted that $a_{1,N} > 0$ is only true for the small pause model (sp). Furthermore, the summation over F needs to be done in order, that is if f is part of the active set, $\beta_{T,f,N}$ needs to evaluated prior to q.

β of the emitting states for MLE and MMIE is given by

$$\beta_{T,q,i} = a_{i,N} \cdot \beta_{T,q,N} \qquad 2 \leq i < N$$

And finally $\beta_{T,q,1}$ is

$$\beta_{T,q,1} = a_{1,j} \cdot b_j \cdot \beta_{T,q,j} \qquad 2 \leq j < N$$

For the remaining time t, the β variable is given by

$$\beta_{t,q,N} \quad = \quad \beta_{t+1,q+1,1} + \beta_{t,q+1,N} \cdot a_{q+1,1,N}$$

Again $a_{q+1,1,N}$ uses the property that the q's are ordered by time and therefore the generalized version needs to adapted.

$$\beta_{t,q,N} = \sum_{f \in F_q} \beta_{t+1,f,1} \cdot l_{q,f} + \beta_{t,f,N} \cdot a_{f,1,N}$$

The calculation needs to be done in the correct order as before. Beside the adaptation for multiple successors, the sum includes the language model score $l_{q,f}$.

β of states 2 to N are given by

$$\beta_{t,q,i} \quad = \quad a_{q,i,N} \cdot \beta_{t,q,N} + \sum_{j=2}^{N-1} a_{q,i,j} \cdot \beta_{t+1,q,j} \quad i = N-1 \rightarrow 2$$

$$\beta_{t,q,1} \quad = \quad \sum_{j=2}^{N-1} a_{q,1,j} \cdot b_{q,j} \cdot \beta_{t,q,j}$$

4.3 Forward Procedure

The forward procedure calculates the α-variable and accumulates statistics for parameter estimation, iterating from time $t = 1$ to T. The accumulated statistics are an expression of α_t, α_{t-1}, β_t and β_{t-1}.

Due to pruning, only a subset of βs are calculated during the backward procedure, thus only αs for this subset need to be calculated. Each step $t = t + 1$, for $t > 1$ performs:

1. Calculate α_t
2. Accumulate statistics
3. Swap α_t and α_{t-1} to save memory

For MLE, α at time t, HMM q, state 1 is given by

$$\alpha_{t,q,1} = \alpha_{t-1,q-1,N} + \alpha_{t,q-1,1} \cdot a_{q-1,1,N}$$

where N is the number of states of HMM $q-1$. The HMM $q-1$ is before HMM q in the transcription. As lattices may contain multiple predecessors, this and all following expressions containing references to $q-1$ must be generalized.

For MMI, $\alpha_{t,q,1}$ is given by

$$\alpha_{t,q,1} = \sum_{b \in B_q} \alpha_{t-1,b,N} \cdot l_{q,b} + \alpha_{t,b,1} \cdot a_{b,1,N}$$

$l_{q,f}$ is the spread language model score. α for the remaining states 2 to N is given by

$$\begin{aligned}
\alpha_{t,q,j} &= a_{q,1,j} \cdot \alpha_{t,q,j} + \left(\sum_{i=2}^{N-1} a_{q,i,j} \cdot \alpha_{t-1,q,i} \right) \cdot b_{q,j} \quad j = 2 \rightarrow N-1 \\
\alpha_{t,q,N} &= \sum_{i=2}^{N-1} a_{q,i,N} \cdot \alpha_{t,q,i}
\end{aligned}$$

4.4 Parameter Estimation

For each HMM, state and mixture, the following variables are accumulated during the forward procedure

$$x_{t,j} = \frac{\left(a_{1,j} \cdot \alpha_{t,1} + \sum_{i=2}^{N-1} a_{i,j} \cdot \alpha_{t-1,i} \right) \cdot \beta_{t,j}}{P(\mathbf{O}|\lambda)} \cdot c_{j,m} \cdot b_{j,m} \quad 1 < j < N$$

where $P(\mathbf{O}|\lambda)$ is estimated with $\beta_{1,1,1}$, that is β at time 1, for the first HMM of the transcript and state 1. Due to this estimation, a lattice must not contain multiple HMMs at the beginning. $c_{q,j,m}$ is the m-mixture weight at state j.

$$\gamma = \sum_{t=1}^{T} x_t$$

$$\theta(\mathbf{O})_k = \sum_{t=1}^{T} (\mathbf{O} - \mu_k) \cdot x_t$$

$$\theta(\mathbf{O}^2)_k = \sum_{t=1}^{T} (\mathbf{O} - \mu_k)^2 \cdot x_t$$

μ and σ of each mixture component k are updated by

$$\hat{\mu}_k = \mu_k + \frac{\theta(\mathbf{O})_k}{\gamma}$$

$$\hat{\sigma}_k = \frac{\theta(\mathbf{O}^2)_k}{\gamma}$$

For MMIE the accumulated γ, μ and σ based on lattices are stored separately. MMI estimation of μ and σ uses transcript and lattice accumulations, where γ^{num}, μ^{num}, σ^{num} refer to the transcript and γ^{den}, μ^{den}, σ^{den} to the lattice. The estimation is given by

$$\hat{\mu}_k = \frac{(\theta(\mathbf{O})_k^{\text{num}} + (\gamma^{\text{num}} \cdot \mathbf{O}) - \theta(\mathbf{O})_k^{\text{den}} + (\gamma^{\text{den}} \cdot \mathbf{O}) + D \cdot \mu_k)}{\gamma^{\text{num}} - \gamma^{\text{den}} + D}$$

$$\hat{\sigma}_k = \frac{(\theta(\mathbf{O}^2)_k^{\text{num}} - \theta(\mathbf{O}^2)_k^{\text{den}}) + (D \cdot \sigma_k + \mu_k^2)}{\gamma^{\text{num}} - \gamma^{\text{den}} + D} - \hat{\mu}^2$$

where D is found in chapter 3.4.2.

4.5 Pruning

Pruning is a vital part of the trainings process. Consider the sentence

"The weather is very warm".

At time $t = 1$ only the word "the" needs to be looked at and not all the other words occurring in the sentence. As the time advances during the forward/backward algorithm only words, respectively the associated HMMs, in a certain time window should be considered. This strategy has the advantages of severely reducing the required amount of computation and assures proper alignment of the transcript and the audio data. Suppose all words are considered at every time and the current position in the audio is at the beginning of the word "weather". Both words "weather" and "warm"

have some phonetic similarity at the beginning and therefore the α and β variable would be increased, although the audio data actually corresponds to the word "weather". To avoid this situation the following two pruning techniques are used:

- Threshold based
- Time based

Threshold based A HMM q is removed from the current active set as soon as

$$max_q(\beta_{q,t,i}) - \beta_{q,t,i} < \epsilon$$

where i is the state number and ϵ is a pruning threshold. In some cases the forward/backward algorithm fails to process an utterance due to a very tight threshold. Therefore *HERest* provides three pruning parameters: start threshold, step size and maximum threshold. The start threshold specifies the initial threshold. If the forward/backward algorithm fails, the threshold is increased by the step size and the utterance is processed again. The threshold is increased until the maximum threshold is reached or the utterance was processed successfully. Unlike the transcript implementation, the lattice version needs to prune each HMM separately. The transcript version defines a pruning beam specifying boundaries based on HMM IDs, which correlate directly to their chronological occurrence in the transcript. As a lattice holds several possible transcripts, multiple HMMs occur at the same time. Therefore no chronologically ordered IDs for each HMM can be given and pruning needs to be applied to each HMM part of the active set. The main purpose of threshold based pruning is speed and memory improvement.

Time based Each HMM has an associated minimum processing duration. Based on this minimum duration, the transcript version creates two vectors for the previously mentioned pruning beam boundaries, q_{lo} and q_{hi}. As soon as the sum of minimum durations of all predecessors of an HMM is larger than time t, the HMM can be included in the active set/pruning beam of the forward procedure. Likewise for the backward procedure, the sum of minimum durations of all successors of an HMM needs to be smaller than $T - t$, where T is the time span of the current utterance. This models the idea, that an HMM should not be used as long as all its successors or predecessors respectively, have not been processed. For the generalized lattice case, the earliest usage of an HMM is defined as the sum of minimum durations of the all HMMs on shortest path from the first or last HMM to the one in question. The time based pruning needs to be equal for forward and backward, especially in connection with *sp*. If the timing is off-by-one the forward and backward procedure could produce unequal $\alpha_{t=T}$ and $\beta_{t=1}$.

4.6 Descriptive Statistics

To validate and compare the MMIE implementation to the existing MLE, intermediate values have been visualized. The HMMs and there states are ordered by time form the y-axis. Each data point is either the value of α, β or $\alpha \cdot \beta$ for the combination of HMM, state and time.

In Figure 5 the α and β according to the forward/backward procedure for a sample utterance are presented. One can see that the value of α and β decreases with the direction of the recursive definition, that is with increasing respectively decreasing time. White areas mark combinations of HMM, state and time where no value is available due to pruning. As one can see the pruning beam is very tight and thus only a few HMMs are active at a time.

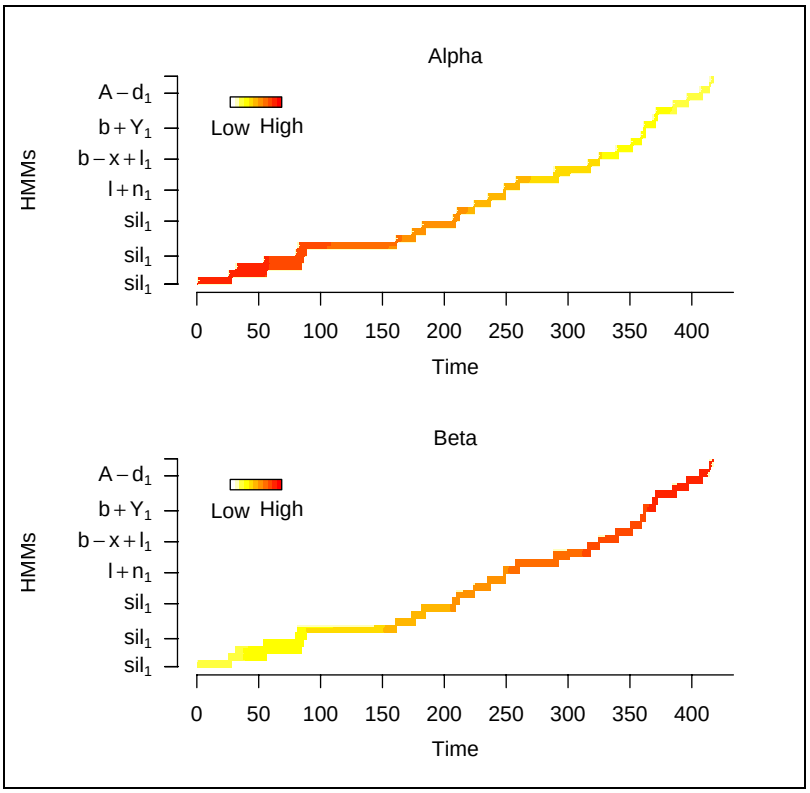

Figure 5: α/β variable for an utterance

Figure 6 presents α and β of a lattice matching the utterance of figure 5. Comparing the lattice with the utterance, one can see that the pruning beam is much wider as similar words are considered at the same time. In difference to the backward procedure calculating the β variable, the forward procedure is able to perform better alignment to the correct utterance, resulting in high values of the α variable for only a few HMMs at a time. The HMMs on the

y-axis are no longer strictly ordered by time due to the internal storage structure.

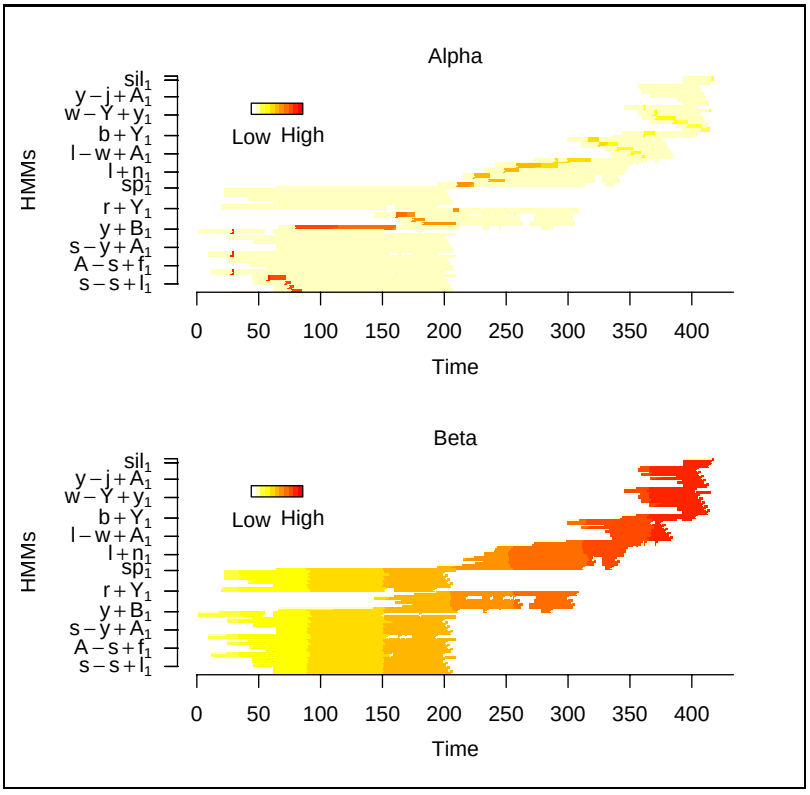

Figure 6: α/β variable for a lattice

In Figure 7 a detail view of an HMM and its states is presented. Each phoneme consists of a series of sounds. The fact that this series is ordered, is modeled by the transition probability matrix that defines a graph that can only be traversed forward. Thus each state models a part of the series. In Figure 7 each state has a series of darker and wider dots accounting for the amount of time it models the audio.

Figure 8 presents $\alpha \cdot \beta$ of a lattice. As one can see the number of active HMMs at a time is much larger compared with the corresponding utterance. The HMMs are approximately ordered by time, but due to the structure of a lattice the path is spread across the plot.

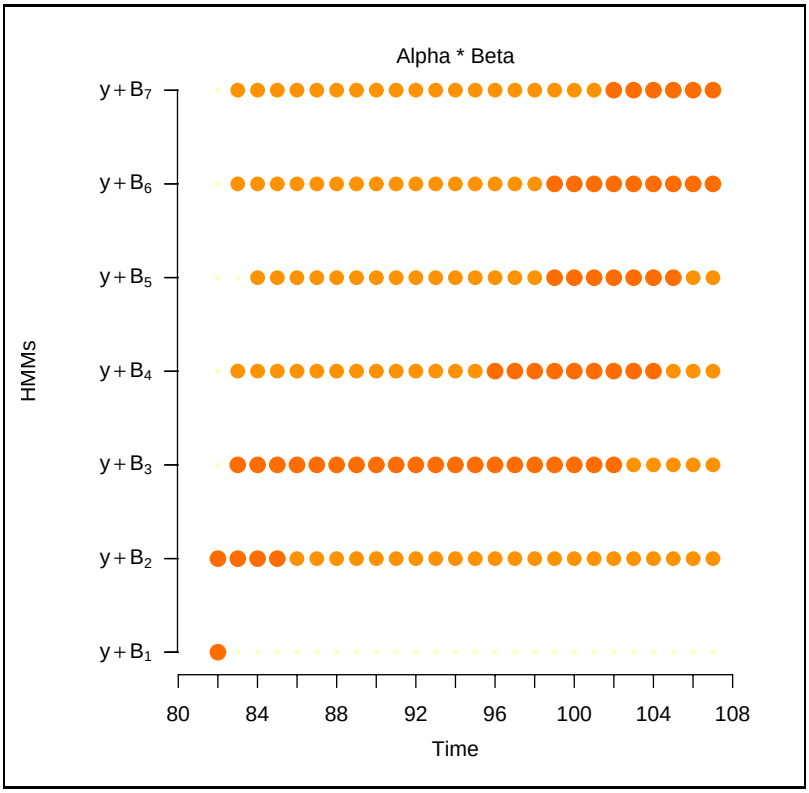

Figure 7: $\alpha \cdot \beta$ variable for an utterance

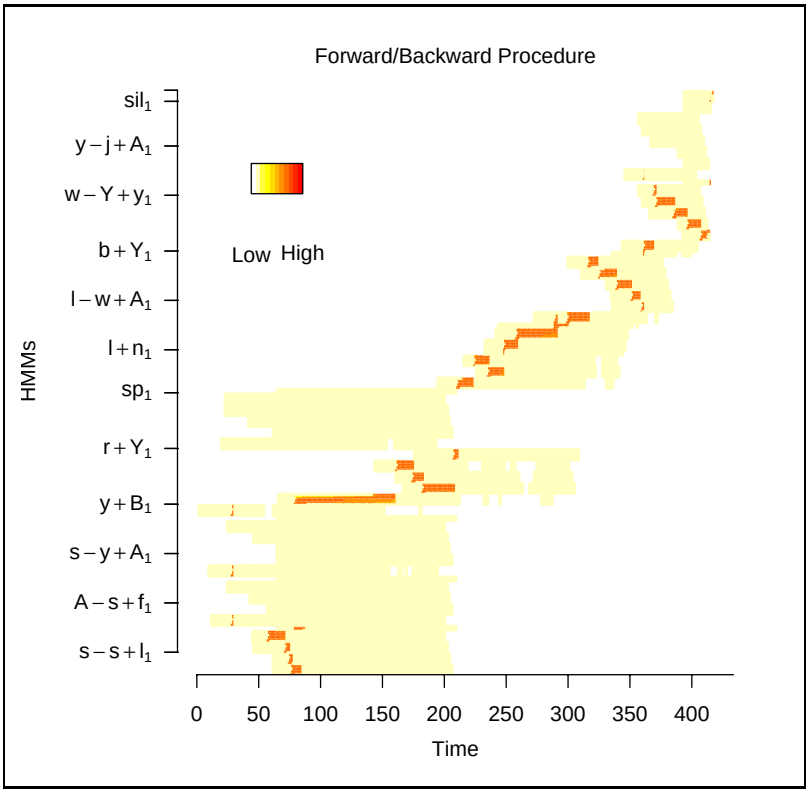

Figure 8: $\alpha \cdot \beta$ variable for a lattice

5 Training a Large Vocabulary Continuous Speech Recognition System

5.1 Introduction

A speech recognition model is trained based on the HTK Book's tutorial [7] . The tutorial describes training for a simple voice dialing application. Our system will recognize broadcast media and hence require a larger dictionary and more audio data. The tutorial is extended by the description of input data conversion and training of multi-mixture models. Additional to the standard MLE training, the newly implemented MMI training procedure will be applied.

Filenames below are always formatted in *italic*, except in the summary tables.

5.2 Data Introduction

The experiments are based on an Arabic broadcast corpus. Among the channels transcribed are Al Arabia, Aljazeera and Saudi Channel. The audio is stored in wav files, using PCM encoding, 16 bit mono 16 kHz. For each audio file a transcript file exists.

The syntax, a sample and the corresponding translation for an utterance:

word word ... (filename-starttime)
`w hkCA nHSl fy AlnhAyp ElY [GRB] (ANN_20050117_1830-000085)`
(and) (so) (we get) (in the end) [garbage]

The text is Buckwalter encoded requiring no Unicode support from any tool. Items surrounded by square brackets are markers for special items (e.g. [GRB] is garbage). Furthermore a dictionary holding all words and appropriate phone level pronunciations is required. Each word can have several associated pronunciations.

The corpus is split into training and test sets:

	Total Time	Transcribed Time
Training Set	113 h	93 h
Test Set	49 min	25 min

The *total* column accounts for the overall amount of audio data, whereas the *transcribed* column only includes data which has been transcribed.

5.3 Data Preparation

The original data used for training is formatted for an outdated training environment and needs to be converted for use with HTK with format differences range from separators to special characters.

5.3.1 Dictionary Format Conversion

The dictionary must list at least all the words and associated pronunciations occurring in training transcripts.

Input:	/sail/ar_ar/3rd/input/master.dict
Type:	Sail Labs style dictionary
Format:	<Prefix> <Word> <Pronunciation> [<Alternative-Pronunciation> ...]

Excerpt from *master.dict*:

```
...
>$ACp $-A-6-h $-A-6-t
>$AE $-A-2
...
```

Output:	output/converted.dict
Type:	HTK style dictionary
Format:	<Word> <Pronunciation>
Tool:	sail2htk.dict.pl

Note: The HTK style dictionary must be sorted.

Excerpt from *converted.dict*:

```
...
XACp X A C h
XACp X A C t
...
```

5.3.2 Phoneme List

All operations on a set of HMM models requires a list of phonemes. Most of the times the phone lists correspond directly to HMM models.

Input:	output/converted.dict	Dictionary with converted phone set
Output:	output/master.dict	Copy of *converted.dict*
	output/phones	List of phonemes used in *converted.dict*

The *converted.dict* dictionary is copied unchanged to *master.dict*. The *-n* parameter produces a list of phonemes and some overview statistics of the dictionary. To model silence, a *sil* entry has to be added to the phoneme list in contrary to the HTK tutorial where each dictionary entry has a *sil* item at the end and therefore the phoneme list already contains the entry.

5.3.3 Dictionary and Phoneme List with sil/sp

Input:	input/master.dict	Dictionary with converted phone set
Output:	output/master-sil.dict	Dictionary with *sil* and non-*sil* postfixed pronunciation with *sp* and *sil* appended
	output/phones-sp	List of phonemes including *sp*
Tool:	bbn2htk.dict.pl	

The new dictionary *master-sil.dict* contains each pronunciation twice, one with *sil* and one with *sp* appended. *sil* is the silence phone and has the same topology as all other phonemes. *sp* (short-pause) has only a single emitting state, that is actually tied to the center state of sil. During training this state can be skipped due to its topology. During alignment (see chapter A.1.6) *HVite* selects the pronunciation that fits best. *phones-sp* is the list of phonemes (equivalent to *phones*), but includes *sp*.

Excerpt from *master-sil.dict*:

```
...
XACp X A C h sil
XACp X A C h sp
...
```

5.3.4 Word Level Transcripts

Input:	config/cfg.list	List of audio files for training
	audio/*.trans	Directory containing transcript files
Output:	output/word-train.mlf	Master label file. Concatenation of transcripts in HTK format
Tool:	trans2mlf.pl	Produces *word-train.mlf*

Sample input transcript *ANN_20050308_0710_News.trans*:

```
AlSAdr bhCA Al$Jn AlEAm (ANN_20050308_0710_News-004212)
[BRT] mn jAnb TAlb AlmnSq (ANN_20050308_0710_News-004454)
...
```

Each .trans file contains utterances ending with an identifier in brackets. [GRB] is a special entry for garbage.

Sample output *word-train.mlf*:

```
"*/ANN_20050308_0710_News-004212.lab"
AlSAdr
bhCA
AlXJn
AlEAm
.
"*/ANN_20050308_0710_News-004454.lab"
[BRT]
mn
jAnb
TAlb
AlmnSq
.
```

Every utterance starts with a quoted path and an utterance identifier (or label identifier in HTK terminology).

5.3.5 Monophone Transcripts

Input:	output/master.dict	Monophone dictionary without *sp*
	output/word-train.mlf	Word level transcription
Output:	output/phone-train.mlf	Monophone transcription
	output/ phone-sp-train.mlf	Monophone transcription with *sp* word separator
Tool:	word2phone_mlf.pl	Produces *phone-train.mlf* and *phone-sp-train.mlf*

Excerpt from *phone-train.mlf*:

```
"*/ANN_20050308_0710_News-004212.lab"
sil
A
S
S
A
d
r
b
h
C
A
A
X
X
J
n
A
l
B
A
m
sil
.
...
```

Each word from *word-train.mlf* is transformed into its monophone representation found in *output/master.dict*. The dictionary might contain multiple pronunciations per word. In this case the first pronunciation is arbitrarily selected. The "correct" or more probable pronunciation is selected at the alignment step (see chapter A.1.6).

The *phone-train-sp.mlf* is similar to *phone-train.mlf* with the exception that words are separated by a *sp*.

5.3.6 Audio Data

Input:	*.wav	Audio files
	training.filelist	List of files to use
Output:	*.ftrs	Extracted audio features
Tool:	HTKcompFeaturesDumper	Extract features from audio files
	wcut	Cuts audio files into pieces
	cut_ar_ar.pl	Driver script for *wcut* and *HTKcompFeaturesDumper*

Each wav file contains audio associated with multiple utterances (e.g. a news episode). The *HTKcompFeaturesDumper* extracts feature vectors as described in (see chapter 2.2) into HTK compatible feature files (.ftrs). Using HTK's *HCopy* would generate a single feature file (.mfc) for the input audio. The *HTKcompFeaturesDumper* needs to be fed with utterance-based audio files because it isn't capable of normalizing audio according to a segmentation file. A segmentation file (.seg) contains time boundaries for each utterance. Therefore *wcut* and a multitude of generated Makefiles is used to cut and extract features. The Arabic corpus used for evaluation contains approximately 96000 utterances.

5.4 Trainings Process

This chapter gives an overview over the training process. A detailed description of the individual steps is found in appendix A.

5.4.1 Training Monophone HMMs

The training starts with HMMs modeling monophones.

1. Compute global means and variances from the audio features.
2. Create HMMs for each monophone initialized with global means and variances.
3. Perform forward/backward algorithm.
4. Creating the small pause model *sp* with a center state tied to the *sil* model.
5. Perform forward/backward algorithm.

6. Assign most probable pronunciation to each word in the transcript (forced-alignment).

7. Perform forward/backward algorithm.

5.4.2 Training Triphone HMMs

To improve accuracy context sensitive HMMs are trained. Each triphone consists of a center phone with a left and right context.

1. Create triphone transcripts.

2. Initialize each triphone HMM with the monophone HMM of its center state (e.g. A-B+C is initialized with B).

3. Perform forward/backward algorithm.

4. Tie HMMs found in dictionary, but not available in transcripts, to existing HMMs based on linguistic criterias.

5. Perform forward/backward algorithm.

6. Increasing Gaussian mixtures of each feature component.

5.4.3 Training a Language Model

Apart from acoustic information, speech recognition employs structural information of the language. HTK's and Sail Labs' recognizer both use uni-, bi- and/or tri-grams.

1. Count n-grams in text corpora.

2. Create a forward tree, containing the pronunciations sorted by prefix.

3. Create the language model, that is calculating probabilities based on the n-gram counts.

5.4.4 Export HTK Models to Sail Labs Models

In a final step the HTK models need to be converted to Sail Labs format.

1. Tie HMMs found in forward tree with existing HMMs based on linguistic criterias.

2. Create composite models (e.g. A-B+C,D is created from A-B+C and A-B+D).

3. Conversion of HTK model to Sail Labs format.

5.5 MMI Experiments

All experiments where performed using Sail Labs recognizer on the 49 minute test set. When comparing the correct transcript with the recognizer output, different errors can occur. The following list gives a list of possible errors and a sample for each.

- Insertion

Transcript	The	weather	is		very	nice.
Recognized	The	weather	is	**a**	very	nice.

- Substitution

Transcript	The	weather	is	very	nice.
Recognized	The	**warning**	is	very	nice.

- Deletion

Transcript	The	weather	is	very	nice.
Recognized	The	weather		very	nice.

For each experiment the following statistics using dynamic programming are calculated:

Ref	Number of words found in the reference
Ins	Number of words inserted
Sub	Number of words substituted
Del	Number of words deleted

$$\begin{aligned}
\text{Corr\%} &= \frac{\text{Ref} - \text{Sub} - \text{Del}}{\text{Ref}} \\
\text{Error\%} &= \frac{\text{Sub} + \text{Del} + \text{Ins}}{\text{Ref}} \\
\text{Ins\%} &= \frac{\text{Ins}}{\text{Ref}} \\
\text{Sub\%} &= \frac{\text{Sub}}{\text{Ref}} \\
\text{Del\%} &= \frac{\text{Del}}{\text{Ref}}
\end{aligned}$$

At first models with a varying number of mixtures and 4 iterations of MLE training are evaluated and produced the following results:

Mixtures	Sub %	Del %	Ins %	Corr %	Err %
2	44.38	5.32	5.38	50.30	55.10
4	40.22	5.62	5.14	54.16	51.02
6	38.12	5.92	4.56	55.94	48.58
8	36.80	5.68	5.24	57.50	47.74
10	35.18	5.70	5.12	59.14	46.02
12	36.30	5.60	5.64	58.08	47.52

Due to the decreased performance of the 12-mixture model, another 3 iterations of MLE training have been applied.

Mixtures	Iter	Sub %	Del %	Ins %	Corr %	Err %
12	5	35.00	5.62	4.90	59.38	45.50
12	6	34.62	5.44	4.88	59.90	44.98
12	7	33.96	5.98	4.86	60.08	44.76

Lattices needed for MMIE training were generated with the 12 mixture, 4 iterations models. At first MMIE was applied to this model **without** using the language model scores.

Iter	Sub %	Del %	Ins %	Corr %	Err %
1	35.42	5.74	4.10	58.88	45.22
2	35.34	6.14	3.50	58.52	44.94
3	38.68	8.02	2.32	53.30	49.02
4	39.12	13.38	2.14	47.50	54.60
5	45.98	18.06	0.96	35.94	65.04
6	55.28	16.68	1.10	28.06	73.02
7	61.48	14.42	1.88	24.14	77.76

To verify convergence of all 6.3 million μ and σ (for each HMM, state and mixture), the mean and median of the absolute difference of μ before the update and $\hat{\mu}$ after MMIE update are found in figure 9.
As one can see the parameters converge, but do not decrease the error rate.

For the second experiment again the 12 mixture, 4 iteration MLE trained model was used as a base line, but MMIE was applied **including** the language model scores.

Iter	Sub %	Del %	Ins %	Corr %	Err %
1	34.96	5.24	3.84	59.80	44.04
2	36.08	5.92	3.40	57.98	45.40
3	38.18	8.02	2.84	53.80	49.02
4	40.94	12.64	2.06	46.46	55.60

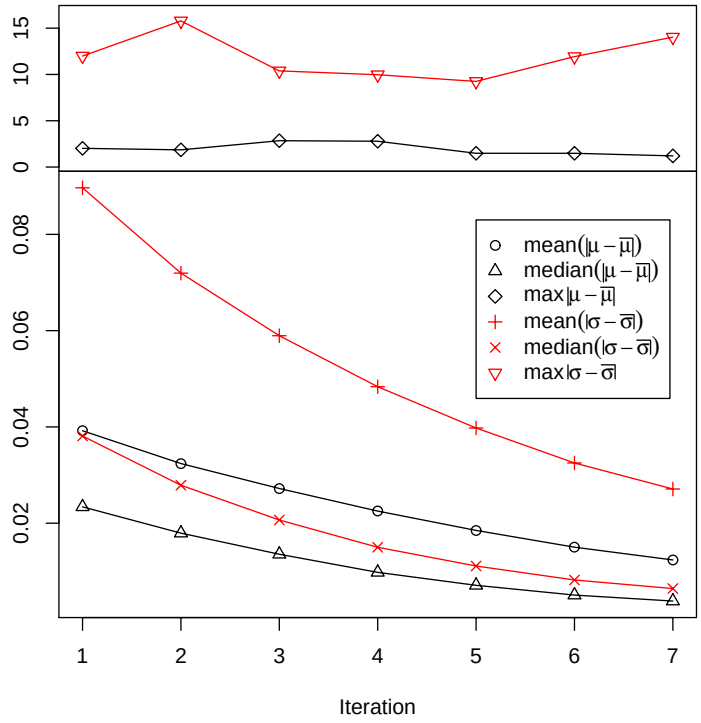

Figure 9: $|\mu - \bar{\mu}|$ and $|\sigma - \bar{\sigma}|$ of MMIE based on 12 mixture 4 iterations MLE model **without** using language model scores

Similar to the first experiment, the error rate decreases, but further iterations increase it again. Statistics on the update are presented in figure 10 to verify the convergence.
Two more experiments using the previously presented 12 mixture 7 iterations MLE trained model were carried out. The performance of MMIE **without** using the language model scores.

Iter	Sub %	Del %	Ins %	Corr %	Err %
1	38.68	8.02	2.32	53.3	49.02
2	35.96	5.96	3.00	58.12	44.88
3	49.90	9.48	3.26	40.64	62.64
4	41.42	12.54	2.12	46.06	56.08

The performance of MMIE **including** the language model scores.

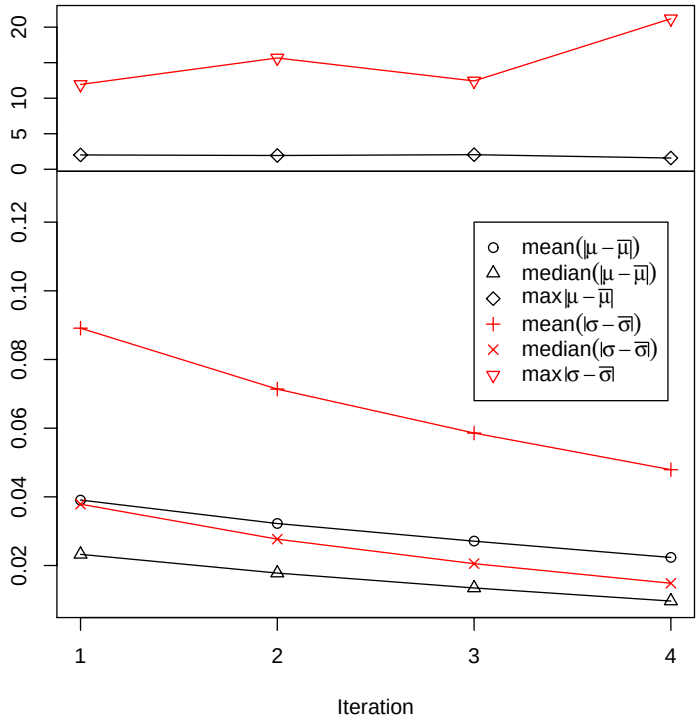

Figure 10: $|\mu - \bar{\mu}|$ and $|\sigma - \bar{\sigma}|$ of MMIE based on 12 mixture 4 iterations MLE model **including** language model scores

Iter	Sub %	Del %	Ins %	Corr %	Err %
1	35.58	5.00	4.08	59.40	44.70
2	36.46	5.96	2.80	57.60	45.22
3	38.90	7.50	2.40	53.60	48.76
4	40.78	12.42	2.04	46.80	55.24
5	47.38	17.24	1.34	35.38	65.98
6	53.42	19.60	1.14	26.98	74.18
7	61.70	15.48	1.74	22.80	78.92
8	63.70	16.94	1.46	19.34	82.14

5.6 Conclusion

The HTK provides a comprehensive framework to quickly train acoustic models. Generating the required recognition outputs for MMIE nor evaluation of models can be done using HTK's recognizer *HVite*, as it is too time consuming. Thus Sail Labs' recognizer was used instead. Several HTK func-

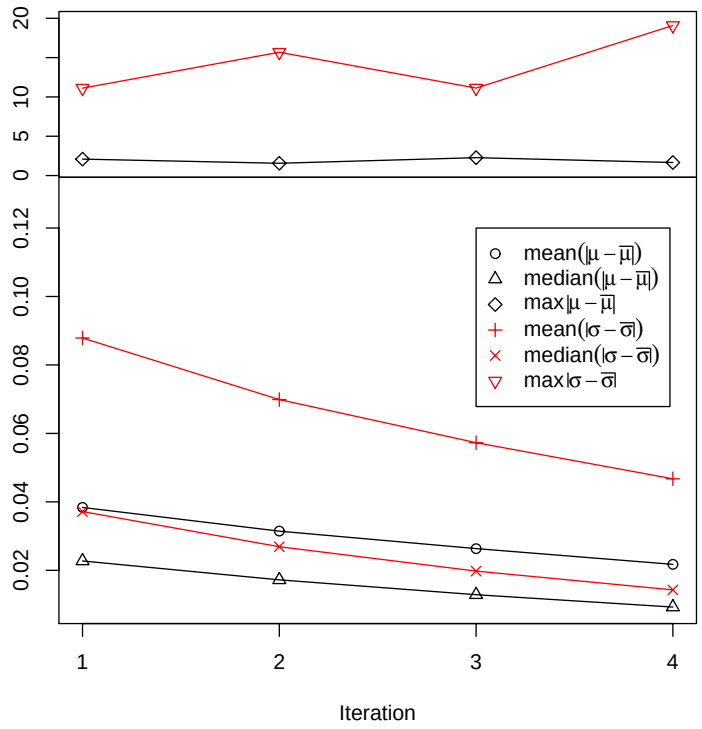

Figure 11: $|\mu - \bar{\mu}|$ and $|\sigma - \bar{\sigma}|$ of MMIE based on 12 mixture 7 iterations MLE model **without** using language model scores

tions were modified to use Intel Performance Primitives [3], which resulted in a major speed improvement.

Based on the results in chapter 5.5, MMIE converges but does not result in a reduction of error rate after the first iteration. As MMIE was only evaluated on a single data set it might yield better results on another corpus.

It is suspected that the creation of composite models, which merges several models into a single one, is responsible for this effect. Therefore the training process will be adapted to minimize the gap between models used during training and recognition.

A Acoustic and Language Model Training Process

This chapter contains a detailed description of the overall trainings process of an acoustic and language model used for the experiments and evaluation of MMIE.

[3] http://www.intel.com/support/performancetools/libraries/ipp/index.htm

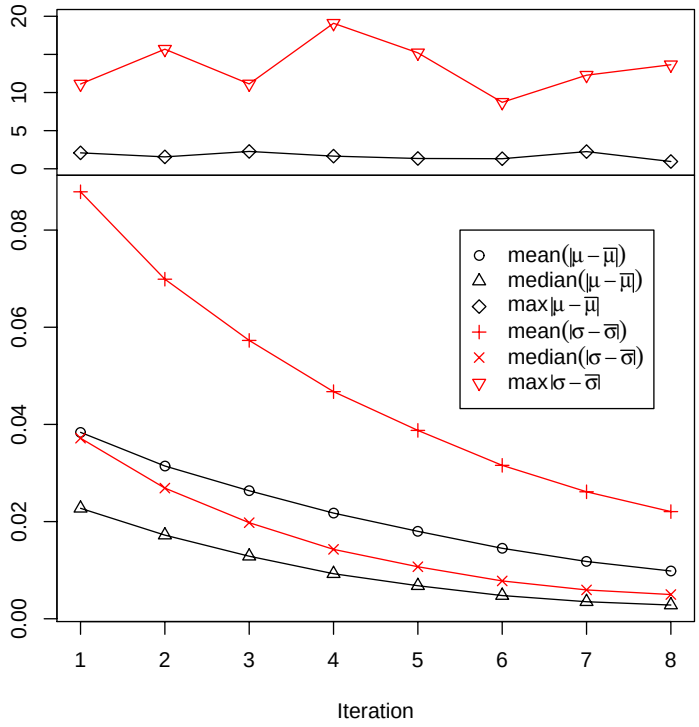

Figure 12: $|\mu - \bar{\mu}|$ and $|\sigma - \bar{\sigma}|$ of MMIE based on 12 mixture 7 iterations MLE model **including** the language model scores

A.1 Training Monophone HMMs

Each time an HTK tool processes feature files, a script file (specified with *-S script* parameter) needs to supplied. This is always *output/train-seg.scp* and contains absolute paths to the .ftrs files as generated in chapter 5.3.6. Most of the HTK tools need a *-C config* parameter describing the feature input. Throughout the training *TARGETKIND = MFCC_0_D_A_Z* is used.

A.1.1 Global Means and Variances

Input:	proto	HMM topology
	audio	Audio features
Output:	output/proto	HMM topology with global means and variances
	output/vFloors	Global variance floors
Tool:	HCompV	Computes global mean and variances

HCompV processes all feature files and computes the global mean and variance for all 45 feature components. The topology of all HMMs is specified in *proto* and an updated version containing the global mean and variances is written to *output/proto*. The *-f 0.01* parameter passed to *HCompV* ensures that no variance will fall below $0.01 * \text{global variance}$. *vFloors* contains variance floors, that is the global variance multiplied by 0.01 (from -f).

The *proto* file containing the topology of a single HMM (in difference to the HTK tutorial, the Sail Labs Engine uses a 7 state (5 emitting states) topology):

```
~o
<VECSIZE> 45
<MFCC_D_A_0>
~h "proto"
<BEGINHMM>
<NUMSTATES> 7
<STATE> 2
<MEAN> 45
0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
0.0 0.0 0.0
<VARIANCE> 45
1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0
1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0
1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0
1.0 1.0 1.0
...
<STATE> 6
<MEAN> 45
0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
0.0 0.0 0.0
<VARIANCE> 45
1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0
1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0
1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0
1.0 1.0 1.0
<TRANSP> 7
0.0 1.0 0.0 0.0 0.0 0.0 0.0
0.0 0.4 0.2 0.2 0.0 0.2 0.0
0.0 0.0 0.4 0.3 0.3 0.0 0.0
0.0 0.0 0.0 0.4 0.3 0.3 0.0
```

```
0.0 0.0 0.0 0.0 0.6 0.4 0.0
0.0 0.0 0.0 0.0 0.0 0.6 0.4
0.0 0.0 0.0 0.0 0.0 0.0 0.0
<ENDHMM>
```

A.1.2 Creating Flatstart HMMs

Input:	output/vFloors	Global variance floors
	output/phones	Monophone list
	output/proto	HMM topology with global means and variances
Output:	output/flatmodel	HMMs with global means and variances
Tool:	concat.flatmodel.sh	Produces *output/flatmodel*

The *flatmodel* file contains:

1. Header from *proto*.
2. Global variance floors from *vFloors*.
3. A copy of *proto* (excluding the header) for each phone listed in *output/phones*.

Thus the *flatmodel* contains HMMs for each phone holding global means and variances.

A.1.3 Training using Monophone Transcripts

Input:	output/flatmodel	HMMs with global means and variances
	output/phones	Monophone list
	output/phone-train.mlf	Monophone transcription
	audio	Audio features
Output:	output/monophone/3/ monophone.model	Monophone HMMs after 3 iterations of training
Tool:	HERest	Performs HMM training
	train.sh	Driver script for HERest

The forward/backward re-estimation algorithm as described in chapter 3.3.3 is performed in parallel. As a starting point the *flatmodel* is used and re-estimated using phone level transcripts found in *phone-train.mlf*. Each parallel instance of *HERest* generates an accumulator file, which are merged and used for re-estimation in a final step generating the model for the next

iteration. The accumulator files store the α and β variables as described in chapter 3.3.1.

A.1.4 Creating *sp* and Tying with *sil* Center State

Input:	output/monophone/3/ monophone.model	Monophone HMMs
Output:	output/monophone/sp1/ monophone-sp.model	Monophone HMMs including the *sp* model
Tool:	copy_sil2sp.pl	Produces *monophone-sp.model*

The *sp* model is created as a 3 state tee HMM, containing a single emitting state that is tied to the center state of *sil*. *sp* is the only model containing a non-zero transition probability from the first to the last state, allowing to skip the emitting state. It should be noted that this phone is specifically handled during forward/backward accumulation of α and β.

A.1.5 Training using Monophone including *sp* Transcripts

Input:	output/monophone/sp1/ monophone-sp.model	Monophone HMMs including the *sp* model
	output/phones-sp	Monophone list including the *sp*
	output/phone-sp-train.mlf	Monophone transcription including *sp* word separator
	audio	Audio features
Output:	output/monophone/sp3/ monophone-sp.model	Monophone HMMs including the *sp* after 3 iterations of training
Tool:	HERest	Performs HMM training
	train.sh	Driver script for HERest

Two iterations of the forward/backward re-estimation algorithm are applied to the monophone HMMs. Compared to A.1.3, a phone list extended by *sp* and transcripts including *sp* after each pronunciation are used for training.

A.1.6 Alignment using Monophone Transcripts

Input:	output/monophone/sp3/ monophone-sp.model	Monophone HMMs including *sp* model
	output/phones-sp	Monophone list including *sp*
	output/word-train.mlf	Word level transcription
	output/master-sil.dict	Dictionary with *sil* and non-*sil* postfixed pronunciation
	audio	Audio features
Output:	output/ phone-aligned-train.mlf	Aligned monophone transcription
	output/ train-seg-aligned.scp	List of feature input files where a corresponding utterance is found in *word-train.mlf*
Tool:	HVite	Speech recognizer producing *phone-aligned-train.mlf*
	find-unalignable.pl	Produces *train-seg-aligned.scp*

The HTK speech recognizer *HVite* is used to do forced-alignment. As mentioned in chapter 5.3.3, up to now the first pronunciation in *phone-train.mlf* is arbitrarily selected. *HVite* tries to align words found in *word-train.mlf* to the most probable pronunciation found in *master-sil.dict*, producing a new transcription *phone-aligned-train.mlf*. *find-unalignable.pl* compares *train-seg.scp* and *phone-aligned-train.mlf* and produces a list of the utterances that *HVite* was able to align. This discrepancy can originate from inaccurate transcriptions.

A.1.7 Training using Monophone aligned Transcripts

Input:	output/monophone/sp3/ monophone-sp.model	Monophone HMMs including the *sp* model
	output/phones-sp	Monophone list including *sp*
	output/ phone-aligned-train.mlf	Aligned monophone transcription
	train-seg-aligned.scp	List of audio files to use
	audio	Audio features. Only the subset *HVite* was able to align
Output:	output/monophone/ aligned2/aligned.model	Monophone HMMs after 2 iterations of training
Tool:	HERest	Performs HMM training
	train.sh	Driver script for *HERest*

Two iterations of the forward/backward re-estimation algorithm are applied to the monophone HMMs using the most probable pronunciations as determined by forced alignment.

A.2 Training Triphone HMMs

A.2.1 Creating Triphone Transcripts and Lists

Input:	output/ phone-aligned-train.mlf	Aligned monophone transcription
	output/master-sil.dict	Dictionary with *sil* and non-*sil* postfixed pronunciation
	mktri.led	Script for *HLEd*
	output/phones	Monophone list
Output:	output/ triphone-aligned-train.mlf	Triphone transcription
	output/triphones	Triphone list
Tool:	HLEd	Produces *triphone-aligned-train.mlf*
	cat, sort -u	Post processing

Triphone transcripts are created based on the corresponding monophone aligned transcripts. As a by-product the list of all triphones occurring in the data is produced and written to *output/triphones*. The monophone list found in *output/phones* is merged into the sorted triphone list.

The *mktri.led* script contains the following commands:

```
# wb ... word boundary
# TC ... create tri-phones
WB sil
WB sp
# missing GRB, BRT, ... (non-speech events)
TC
```

WB specifies word boundaries. As a result no context dependent *sil* and *sp* triphones are generated. *TC* commands *HLEd* to produce the triphone transcripts found in *triphone-aligned-train.mlf*.

Excerpt from *output/triphones*:

```
A
```

```
A+A
A+B
A+C
A+D
...
A-A+T
A-A+h
A-A+l
...
```

The list not only contains triphones, but also mono- and biphones. Context-dependent phones are specified in the following form:

X-Y	X before Y
Y+Z	Y before Z
X-Y+Z	Y between X and Z

Y is called the "center phone", regardless of whether a left or right context is specified.

A.2.2 Creating the Triphone Clone Scripts

Input:	output/triphones	Triphone list
	output/phones-sp	Monophone list including *sp*
Output:	output/mktri.hed	Triphone cloning script for *HHEd*
Tools:	perl-scripts	Produce *mktri.hed*

The initial set of triphone HMMs is based on the monophone aligned HMMs found in *output/monophone/aligned2/aligned.model*. For each triphone found in *output/triphones* the monophone HMM equal to the triphones center-phone is selected; e.g. the HMM A-B+D found in *output/triphones* will be a copy of the monophone HMM B from *output/.../aligned.model*. This is done with the clone command (*CL output/triphones*) instructing *HHEd*. The tying commands (TI) are created to reduce the number of parameters, tying the transition matrices of triphones, biphones and monophones with an identical center phone (*Note: *-A matches only biphones with A as center phone*). Thus all triphones and biphones with center phone A and monophone A will share the same transition matrix.

Sample *output/mktri.hed*:

```
CL output/triphones
TI T_A {(*-A+*,*-A,A+*,A).transP}
TI T_sil {(*-sil+*,*-sil,sil+*,sil).transP}
```

```
TI T_sp {(*-sp+*,*-sp,sp+*,sp).transP}
TI T_l {(*-l+*,*-l,l+*,l).transP}
TI T_Y {(*-Y+*,*-Y,Y+*,Y).transP}
...
TI T_J {(*-J+*,*-J,J+*,J).transP}
TI T_BRT {(*-BRT+*,*-BRT,BRT+*,BRT).transP}
TI T_GRB {(*-GRB+*,*-GRB,GRB+*,GRB).transP}
```

A.2.3 Initializing Triphone HMMs with Monophone HMMs

Input:	output/monophone/ aligned2/aligned.model	Monophone HMMs
	output/mktri.hed	Script for *HHEd* (see chapter A.2.2)
	output/triphones	Triphone list; referenced in *mktri.hed*
	output/phones-sp	Monophone list including *sp*
Output:	output/triphone/1/ triphone.model	Triphone HMMs
Tools:	HHEd	Produces *triphone.model*

HHEd initializes and ties the triphone HMMs with the associated monophone HMMs as specified in chapter A.2.2.

A.2.4 Performing Forward/Backward using Triphone Transcripts

Input:	output/triphone/1/ triphone.model	Triphone HMMs
	output/triphones	Triphone list
	output/ triphone-aligned-train.mlf	Triphone transcription
	audio	Audio features
	train-seg-aligned.scp	List of audio files to use
Output:	output/triphone/3/ triphone.model	Triphone HMMs after 2 iterations of training
Tool:	HERest	Performs HMM training
	train.sh	Driver script for *HERest*

Two iterations of the forward/backward re-estimation algorithm are applied to the triphone HMMs.

A.2.5 Creating a Triphone based Dictionary and a Complete Triphone List

Input:	output/triphones	Triphone list
	output/master-sil.dict	Dictionary with monophone pronunciations
	mktri.ded	Script for *HDMan*
Output:	output/triphones-fulllist	Complete list of triphones
	output/master-triphone.dict	Dictionary with triphone pronunciations
Tools:	HDMan	Produces *triphones-fulllist*
	cat, sort	Sorting of *master-triphone.dict*

In general the dictionary is a superset of words occurring in training transcripts, representing all words and their associated pronunciations the recognizer should be able to detect. The *output/triphones* list and the *output/triphone/3/triphone.model* contains only the triphone HMMs found in the training transcripts. To accommodate for triphones found in the dictionary but not part of *output/triphones*, a similar triphone is selected in the next step.

HDMan produces a complete list of triphones and a triphone based dictionary. The dictionary is only required for MMI training. Each pronunciation is stored once with a *sil* and once with a *sp* postfix, just as in *master-sil.dict*.
The *mktri.ded* script contains the following commands:

```
# create triphones
TC
```

Excerpt from *master-triphone.dict*.

```
...
XACp X+A X-A+C A-C+h C-h sil
XACp X+A X-A+C A-C+h C-h sp
XACp X+A X-A+C A-C+t C-t sil
XACp X+A X-A+C A-C+t C-t sp
...
```

A.2.6 Tying HMMs according to Linguistic Criteria

Input:	output/triphones-fulllist	Triphone list occurring in dictionary
	output/triphones	Triphone list occurring in training transcripts
	output/phones-sp	Monophone list including *sp*
	output/triphone/3/ triphone-model	Triphone HMMs
	output/ triphones-aligned-train.mlf	Triphone transcription
	output/triphone3/ stats	Statistics on HMMs
	questions.list	Classes of phonemes
Output:	output/tree.hed	Script for *HHEd*
	output/trees	Clustering decision tree
	output/tiedlist	Compressed list of triphones
	output/triphone/tied1/ tied.triphone.model	Tied triphone HMMs
Tools:	HHEd	Produces *tied.triphone.model*
	question.list2tree.hed.pl	Produces *tree.hed*

In the first step a decision tree for tying HMMs is created by iteratively clustering all HMMs found in *output/triphones*. Then this decision tree is used to tie HMMs and produce missing HMMs specified by *output/triphones-fulllist*. This is done on a per state level. Tying HMM states increases the model robustness and reduces the parameters which have to be estimated. Linguistically and acoustically similar contexts (e.g. A as center phone) are grouped together.

For decision tree clustering the following input files are used:

- *questions.list* holds the classes of phonemes grouped by linguistic criteria, such as place and manner of articulation (see chapter 2.1).

  Excerpt from *questions.list*

  ```
  close: y w
  open: A
  front: y A
  ...
  ```

- The *stats* file provides information about the number of occurrences

and occupation statistics for HMMs. Each entry contains the HMM name, number of occurrences and occupation statistics for each state.

Excerpt from *stats*

```
"w-r+X"   37   55.698982  13.440671  ...
"w-j+r"    7   13.021901  17.921888  ...
"s-w+r" 1337  2420.234131 830.687561 ...
...
```

The intermediate *tree.hed* script generated by *question.list2tree.hed.pl* for *HHEd* contains the following commands

- Questions to be used in the construction of the decision tree

```
QS "L_close" {y-*,w-*}
QS "R_close" {*+y,*+w}
QS "L_open" {A-*}
QS "R_open" {*+A}
QS "L_front" {y-*,A-*}
QS "R_front" {*+y,*+A}
...
```

- List of trees to be generated; for instance in the following example each combination of center phone and state gets a separate tree.

```
TB 350.0 "A_2" {(A, *-A, *-A+*, A+*).state[2]}
TB 350.0 "l_2" {(l, *-l, *-l+*, l+*).state[2]}
...
TB 350.0 "Y_2" {(Y, *-Y, *-Y+*, Y+*).state[3]}
...
```

- Assign HMM states for unseen triphones using the above specified decision trees.

```
AU output/triphones-fulllist
```

- List of outputs, that is a compressed list of triphones found in *output/tiedlist* and the decision tree *output/trees*.

```
CO output/tiedlist
ST output/trees
```

A.2.7 Performing Forward/Backward on Tied HMMs

Input:	output/triphone/tied1/ tied.triphone.model	Tied Triphone HMMs
	output/tiedlist	Compressed list of triphones
	output/ triphone-aligned-train.mlf	Triphone transcription
	audio	Audio features
	train-seg-aligned.scp	List of audio files to use
Output:	output/triphone/tied3/ tied.triphone.model	Tied triphone HMMs after 2 iterations of training
Tool:	HERest	Performs HMM training
	train.sh	Driver script for *HERest*

Two iterations of the forward/backward estimation algorithm are applied to the tied triphone HMMs.

A.2.8 Increasing Mixtures

Until now each of the 45 features was represented by a single mean and variance value for each state and HMM. To improve recognition performance, Gaussian mixtures are used to represent each feature. This stage iteratively increases the number of mixtures and applies the forward/backward estimation algorithm after each increase. Initial experiments used 2, 4, 6, 8, 10 and 12 mixtures and 3 iterations of the estimation.

Input:	output/triphone/tied3/ tied.triphone.model	Tied Triphone HMMs
	output/tiedlist	Compressed list of triphones
	output/ triphone-aligned-train.mlf	Triphone transcription
	audio	Audio features
	train-seg-aligned.scp	List of audio files to use
	output/mixture/12/ inc.mixture	Script for *HHEd*
Output:	output/mixture/12/4 model	12 Gaussian mixture tied triphone HMMs after 3 iterations of training
Tool:	HERest	Performs HMM training
	HHEd	Increases the number of mixtures
	train.sh	Driver script for *HERest*

The *inc.mixture* script for *HHEd* contains the following command:

```
MU 12 {*.state[2].mix,*.state[3].mix,*.state[4].mix,
       *.state[5].mix,*.state[6].mix}
```

It instructs *HHEd* to increase the number of mixtures to 12 for each of the 5 emitting states for every HMM.

A.3 Training a Language Model

Besides the acoustic information, the speech recognizer also uses structural information inherent to the language. The type of information ranges from simple uni-grams (the probability of a single word occurring) to grammars such as parts-of-speech. The speech recognizer supplied by HTK as well as Sail Labs uses uni-, bi- and/or tri-grams, but stores them differently for efficient usage. This chapter describes the language model training process using the Sail Labs Language Model Toolkit (LMT).

Most of the text input files are Buckwalter [4] encoded and need to be converted to match the transformed Buckwalter encoding used for HTK (see chapter 2.1).

The training process is implemented using Makefiles, shell- and perl scripts. To avoid time consuming dependency checks for stages that depend on a large number of files produced by previous stages, *<target>.done* files created after each stage are used instead.

Each section is split into:

1. a general introduction to the stage
2. a table holding the Makefile target, input and output files, used tools and sample input and output file contents
3. a process description.

A.3.1 Text Corpora and Audio Transcript Conversion

The text corpora and the transcripts of the audio used for the acoustic models are the basis for LMT.

[4] http://www.qamus.org/transliteration.htm

Target:	output/lm/trans.done output/lm/trans.audio.done
Input:	*.trans
Output:	*.txt
Tools:	trans2matador.txt.sh
	trans2matador.txt.pl
Sample Input:	AlY AlsAdp Alm$trkyn (AFA19940513.0002-0)
Sample Output:	AlY AlsAdp AlmXtrkyn

At the end of each input utterance an identifier is specified in parenthesis. For LMT training the identifier needs to be stripped and the character encoding needs to be adapted (see chapter 2.1).

A.3.2 Text Input File Lists

The list of files used for LMT.

Target:	output/lm/trans.filelist
	output/lm/trans-audio.filelist
Input:	directory contents
Output:	output/lm/trans.filelist, output/lm/trans-audio.filelist
Tools:	find

A.3.3 Create Chunks of Files

For efficient processing of the text, the files are grouped into chunks based on the number of words found in each file.

Target:	output/lm/idify.split.done
Input:	*.txt produced in chapter A.3.1
Output:	lmFileList.for_idify.part0000
	lmFileList.for_ngrammer.part0000
Tools:	lm.idify.split.pl

lmFileList.for_idify.part0000 contains all filenames with absolute path for identification, that is replacing each word with its numeric ID from the vocabulary. *lmFileList.for_ngrammer.part0000* contains all filenames with absolute paths for the *ngrammer*. The *ngrammer* will count n-grams (uni-grams, bi-grams, tri-grams, ...) for the language model.

A.3.4 ID'ing Text

All LMT statistic tools require the text to be ID'ed.

Target:	output/lm/idify.done
Input:	output/lm/idify/lmFileList.for_idify.part0000
	output/lm/idify-audio/lmFileList.for_idify.part0000
	vocab (the vocabulary)
Output:	output/lm/idify/*.id
	output/lm/idify-audio/*.id
Tools:	idify

Each word is replaced by its numeric ID found in the vocabulary or a reserved ID for unknown words. The results are stored in binary format.

A.3.5 Counting N-grams

The bigrams and trigrams are used to predict a word based on the previous one or two words during recognition.

Target:	output/lm/ngrammer.done
	output/lm/ngrammer-audio.done
Input:	output/lm/idify/lmFileList.for_ngrammer.part0000
	output/lm/idify-audio/lmFileList.for_ngrammer.part0000
	vocab (the vocabulary)
Output:	output/lm/ngrammer/countFileFwd*
	output/lm/ngrammer-audio/countFileFwd*
Tools:	ngrammer

The *ngrammer* counts the occurrence of each word bi- and trigram and stores it in binary format in multiple count files.

A.3.6 Merging Multiple Count Files

Target:	output/lm/countFileFwd
Input:	output/lm/ngrammer/countFileFwd*
	output/lm/ngrammer-audio/countFileFwd*
Output:	output/lm/countFileFwd
Tools:	quickngrammerge

The *countFileFwd* files are merged together with different weights. In general the transcripts of the audio get higher weights (e.g. 3 to 5) than the text corpora. The weighting can be used to decrease the word error rate during recognition for a specific domain (e.g. a single channel such as Aljazeera).

A.3.7 Creating the Forward Tree

The pronunciations are organized into a common-prefix tree for optimized access during recognition.

Target:	output/lm/fm_tree_sorted
Input:	output/master.dict-mat
	output/vocab
	output/phones-mat
Output:	output/lm/fm_tree_sorted
	output/lm/word_classes
	output/lm/matador.htk.shared.model.list
Tools:	makeFwdTree

master.dict-mat is the dictionary in Sail Labs' internal format containing all pronunciations of the recognition vocabulary *vocab*. The *phones-mat* contains the phoneme-inventory excluding *sil* and *sp*. *word_classes* represents all words with a common-prefix which can be reached from a given node in the tree. *matador.htk.shared.model.list* is a list of triphone models and composite triphone models corresponding to all nodes in the tree.

A.3.8 Creating the Language Model

The language model contains bigram probabilities for the above created classes of words (according to the forward tree) given a preceding word as context.

Target:	output/lm/matLm2gFwdClasses
Input:	output/countFileFwd
	output/word_classes
	output/vocab
Output:	output/lm/matLm2gFwdClasses
Tools:	makeNgr

matLm2gFwdClasses is the language model used by Sail Labs' recognizer. The recognition vocabulary is found in *vocab*. *countFileFwd* contains the number of occurrences of each word bi- and trigram.

A.4 Exporting HTK Models

Sail Labs' recognizer (internally named Matador) uses a different storage format and composite models, which are not supported by HTK. Composite models are a combination of triphone models, sharing the left context. e.g. the composite model A-B+C,D models the combination of A-B+C and A-B+D.

Therefore the following steps are performed

1. *matador.htk.shared.model.list* holds the list of required HMMs (monophone, biphone, triphone and composite models) in Sail Labs internal format. It is converted to an HTK style list found in *composite.models*.
2. Decompose composite models found in *composite.models* into individual triphone models (*composite.models.flat*).
3. Merge the decomposed models with the phoneme inventory *tiedlist* used during acoustic training (*composite.models.flat.merged*).
4. Extract the list of composite models (*composite.models.only*).

A.4.1 Resolving Unseen HMMs

matador.htk.shared.model.list most likely contains HMMs not found in the trainings data. The so-called "unseen" HMMs are modeled by a similar model based on linguistic criteria.

Input:	create.missing.triphones.using.trees.hed
	output/trees
	output/composite.models.flat.merged
Output:	output/composite.models.flat.merged.compact
	output/lm/matador.htk.shared.model.list
	output/mixture/12/model.pre-matador
Tools:	HHEd

create.missing.triphones.using.trees.hed contains the commands for *HHEd*

```
LT output/trees
AU output/composite.models.flat.merged
CO output/composite.models.flat.merged.compact
```

1. **LT** loads the tree of models categorized by linguistic criteria (see chapter A.2.6).
2. **AU** commands *HHEd* to link each model to its corresponding model residing in the leaf found in the decision tree (*trees*).

3. **CO** compresses the model representation and stores it in *composite.models.flat.merged.compact*.

A.4.2 Matador Model Creation

The conversion of the HTK model to a Matador model is done with a modified version of *HHEd*.

The command file *htk2matador.hed* for *HHEd* contains

```
LS output/mixture/12/4/stats.no-sp
CC output/composite.models.only
SM output/composite.models
MA 0
```

and does the following

1. **LS**: Load occupation statistics (see the definition of γ in chapter 3.4.2).
2. **CC**: Creates composite models by mixing together the individual triphones using *MixDown(...)* of *HHEd*.
3. **SM**: Sorts the models for Sail Labs internal representation corresponding to the model IDs in the nodes of the forward tree.
4. **MA**: Writes the model in Matador format for the first pass of Sail Labs' recognizer.

A.4.3 Fast Gaussian Tree

To pre-select among the Gaussian of all models, a "fast Gaussian tree" is generated. This reduces the number of Gaussian evaluations to be performed for a given input vector.

A.5 MMI Training

To generate the required lattices for MMI estimation, the model produced in the previous chapters is required. Initial experiments were done using HTK's recognizer *HVite*.

A recognition pass for 3 minutes of audio took over 10 hours on a 3.4 GHz Pentium4 machine, so Sail Labs' recognizer was adapted to produce the lattice. The recognizer normally runs in real-time on a comparable machine. Generation of lattice for the complete trainings set of 96 hours took approx. 2-3 days on 3 machines ranging from 2 to 3.4GHz.

The recognizer output does not produce the standard HTK format for lattices and needs the following post processing:

1. *lat2slf.pl* maps global pronunciation IDs as used by Sail Labs to the corresponding relative IDs of HTK.

2. *compress_lattice.pl* is used to remove duplicate paths in the lattice.

3. *strip.s.from.lattice.pl* converts initial `<s>` and final `</s>` tags into *sil*.

4. *validate.lattice.pl* does a rough validation of the lattices.

An important part of MMI estimation is the inclusion of language model scores. To reduce the amount of software development, the generalized version of the forward/backward algorithm in *HERest* was used to calculate γ^{num}, that is the occupation counts based on the transcript including language model scores. Therefore the transcripts were converted to degenerated lattices including language model scores:

1. *trans2all.pl* concatenates all *.trans* files into a single file *trans.all*.

2. *evallm.exe* processes *trans.all* and retrieves the language model scores for all transcripts. The transcript is concatenated so that the time-consuming loading of the language model only has to be done once.

3. *trans.all.2.slf.pl* creates the degenerated lattices in HTK standard lattice format.

4. *validate.lattice.pl* is used again to validate the lattices.

It is required that the lattices from the recognition run include the correct transcript, thus *merge.transcript.into.slf.all.pl* is used to merge the transcript with the corresponding lattice. The current implementation of the merging process creates a new path starting from the longest prefix found in the lattice to the longest postfix.

Now the actual MMI estimation using the generalized version of *HERest* can begin.

References

[1] http://www.arts.gla.ac.uk/ipa/ipa.html

[2] L. Rabiner and B. H. Juang: Fundamentals of Speech Recognition. New Jersey: Prentice Hall (1993)

[3] A. V. Oppenheim and R. W. Schafter: Zeitdiskrete Signalverarbeitung. München, Wien: Oldenbourg Verlag (1992).

[4] B. Logan: Mel Frequency Cepstral Coefficients for Music Modeling. International Symposium on Music Information Retrieval (2000).

[5] S. Mallat and W. L. Hwang: Singularity detection and processing with wavelets. IEEE Trans. Inform. Theory, vol. 38, pp. 617-643 (1992).

[6] J. P. Olive, A. Greenwood and J. Coleman: Acoustics of American English Speech - A Dynamic Approach. Springer Verlag (1993).

[7] S. Young, G. Evermann, T. Hain et al: The HTK Book. Cambridge University Engineering Department (2002).

[8] L. E. Baum: An inequality and associated maximization technique in statistical estimation for probabilistic functions of Markov processes. Inequalities, 3 (1972).

[9] M. Kumar, N. Rajput, and A. Verma: A large-vocabulary continuous speech recognition system for Hindi. IBM Journal of Research and Development, Volume 48, Number 5/6, (2004).

[10] L.R. Bahl, S. Balakrishnan-Aiyer, M. Franz, P.S. Gopalakrishnan, R. Gopinath, R. Novak, M. Padmanabhan, S. Roukos. The IBM Large Vocabulary Continuous Speech Recognition System for the ARPA NAB News Task. In Proceedings of ARPA Spoken Language System Technology Workshop, pp 121-126, (1995).

[11] P. Beyerlein, X. Aubert, R. Haeb-Umbach, M. Harris, D. Klakow, A. Wendemuth, S. Molau, H. Ney, M. Pitz and A. Sixtus. Large vocabulary continuous speech recognition of Broadcast News - The Philips/RWTH approach. In Speech Communication, Volume 37, pp 109-131, (2002).

[12] M.Y. Hwang. Subphonetic Acoustic Modeling for Speaker-Independent Continuous Speech Recognition. Ph.D. thesis, Tech Report No. CMU-CS-93-230, Computer Science Department, Carnegie Mellon University, (1993).

[13] P. Lamere, P. Kwok, E.B. Gouvêa, B. Raj, R. Singh, W. Walker, P. Wolf. The CMU Sphinx-4 Speech Recognition System. Proc. of the ICASSP 2003, (2003).

[14] R. Schwartz, T. Colthurst, N. Duta, H. Gish, R. Iyer, C.-L. Kao, D. Liu, O. Kimball, J. Ma, J. Makhoul, S. Matsoukas, L. Nguyen, M. Noamany, R. Prasad, B. Xiang, D.-X. Xu, J.-L. Gauvain, L. Lamel, H. Schwenk, G. Adda, L. Chen. Speech recognition in multiple languages and domains: the 2003 BBN/LIMSI EARS system. Acoustics, Speech, and Signal Processing, 2004. Proceedings. (ICASSP '04). (2004).

[15] L. Nguyen, T. Anastasakos, F. Kubala, C. LaPre, J. Makhoul, R. Schwartz, N. Yuan, G. Zavaliagkos, Y. Zhao. The 1994 BBN/BYBLOS Speech Recognition System. In Proceedings of ARPA Spoken Language Systems Technology Workshop, pp. 77-81, (1995).

[16] G. Evermann, H.Y. Chan, M.J.F. Gales, T. Hain, X. Liu, D. Mrva, L. Wang, P.C. Woodland. Development of the 2003 CU-HTK Conversational Telephone Speech Transcription System. Proceedings ICASSP 2004, Montreal, (2004).

[17] I. Kirschinger, H. Tomabechi and J.-I. Aoe. Recent Advances in Continuous Speech Recognition Using the Time-Sliced Paradigm. International Workshop on Soft Computing in Industry, Muroran, Japan, (1996).

[18] E. Trentin, M. Gori. Robust combination of neural networks and hidden Markov models for speech recognition. Neural Networks, IEEE Transactions on , vol.14, no.6pp. 1519- 1531, (2003).

[19] W. Macherey, L. Haferkamp, R. Schlüter and H. Ney. Investigations on Error Minimizing Training Criteria for Discriminative Training in Automatic Speech Recognition. In the 9th European Conference on Speech Communication and Technology (Interspeech 2005), Lisbon, Portugal, (2005).

[20] V. Valtchev. Discriminative Methods in HMM-based Speech Recognition. PhD Thesis, Cambridge University, Department of Electrical Engineering, (1995).

[21] S. Kapadia. Discriminative Training of Hidden Markov Models. PhD Thesis, Cambridge University, Department of Electrical Engineering, (1998).

[22] P.C. Woodland and D. Povey. Large scale discriminative training for speech recognition. Proc. ISCA ITRW ASR2000, (2000).

[23] N. Marhav and C.-H. Lee: On the asymptotic statistical behavior of empirical cepstral coefficients. IEEE Transactions and Signal Processing 41, (1990-1993).

[24] Y. Normandin: Maximum Mutual Information Estimation of Hidden Markov Models. Automatic Speech and Speaker Recognition: Kluwer Academic Publishers (1996).

[25] http://en.wikipedia.org/wiki/Place_of_articulation

[26] http://en.wikipedia.org/wiki/Manner_of_articulation

[27] A.P. Dempster, N.M. Laird and D.B. Rubin. Maximum-likelihood from incomplete data via the em algorithm. J. Royal Statist. Soc. Ser.B., 39, (1977).

[28] R. Redner and H. Walker. Mixture densities, maximum likelihood and the em algorithm. SIAM Review, 26(2), (1984).

[29] Z. Ghahramami and M. Jordan. Learning from incomplete data. Technical Report AI Lab Memo No. 1509, CBCL Paper No. 108, MIT AI Lab, (1995).

[30] M. Jordan and R. Jacobs. Hierarchical mixtures of experts and the em algorithm. Neural Computation, 6:181-214, (1994).

[31] C.F.J. Wu. On the convergence properties of the em algorithm. The Annals of Statistics, 11(1):95-103, (1983).