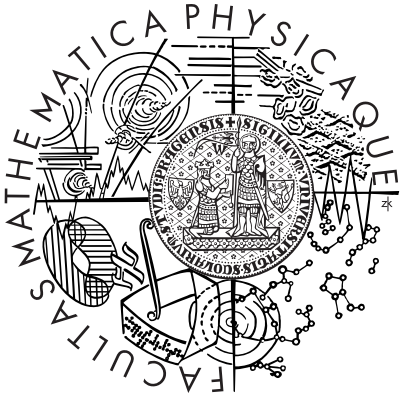
Charles University in Prague
Faculty of Mathematics and Physics

MASTER THESIS



Ondřej Plátek

Automatic speech recognition using Kaldi

Institute of Formal and Applied Linguistics
Supervisor: Ing. Mgr. Filip Jurčíček, Ph.D.
Study branch: Theoretical Computer Science

Prague 2013

TODO: Insert the task ../images/zadani[1−2]*.png for the printed version*

I would like to thank my supervisor, Ing. Mgr. Filip Jurčíček, Ph.D., for his advice, guidence and for keeping me motivated. I would like to thank namely, Matěj Korvas for HTK scripts results and advices, Lukáš Žilka, David Marek and Ondřej Dušek for hacks in Vim, Bash and Perl, Marek Vašut for advices with shared library linking, Tomáš Martinec for C++ advices and Pavel Mencl for proofreading. I am also very grateful to the Kaldi team, which was very responsive and helpful. Expecially, Daniel Povey and Vassil Panayotov. Last but not least, I would like to thank my parents and Adéla Čiháková for all the help and support.

I declare that I wrote my master thesis independently and exclusively with the use of the cited sources. I agree with lending and publishing this thesis.

I understand that my work relates to the rights and obligations under the Act No. 121/2000 Coll., the Copyright Act, as amended, in particular the fact that the Charles University in Prague has the right to conclude a license agreement on the use of this work as a school work pursuant to Section 60 paragraph 1 of the Copyright Act.

Prague, August 1, 2013 Ondřej Plátek

Název práce: Rozpoznávání řeči pomocí Kaldi
Autor: Ondřej Plátek
Katedra: Ústav formální a aplikované lingvistiky
Vedoucí diplomové práce: Ing. Mgr. Filip Jurčíček, Ph.D.
E-mail vedoucího: jurcicek@ufal.mff.cuni.cz

Abstrakt: Tématem této práce je implementace výkonného rozpoznávače v open-source systému trénování ASR Kaldi (http://kaldi.sourceforge.net/) pro dialogové systémy. Kaldi již obsahuje ASR dekodéry, které však nejsou vhodné pro dialogové systémy. Hlavními důvody jsou jejich malá optimalizace na rychlost a jejich velké zpoždění v generování výsledku po ukončení promluvy. Cílem této práce je proto vyvinutí real-time rozpoznávače pro dialogové systémy optimalizovaného na rychlost a minimalizujícího zpoždění. Zrychlení může být realizováno například pomocí multi-vláknového dekódování nebo s využitím grafických karet pro obecné výpočty. Součástí práce je také příprava akustického modelu a testování ve vyvíjeném dialogovém systému "Vystadial".

Klíčová slova: ASR,rozpoznávání mluvené řeči, dekodér

Title: Automatic speech recognition using Kaldi
Author: Ondřej Plátek
Department: Ústav formální a aplikované lingvistiky
Supervisor: Ing. Mgr. Filip Jurčíček, Ph.D.
Supervisor's e-mail address: jurcicek@ufal.mff.cuni.cz

Abstract: The topic of this thesis is to implement efficient decoder for speech recognition training system ASR Kaldi (http://kaldi.sourceforge.net/). Kaldi is already deployed with decoders, but they are not convenient for dialog systems. The main goal of this thesis to develop a real-time decoder for a dialog system, which minimize latency and optimize speed. Methods used for speeding up the decoder are not limited to multi-threading decoding or usage of GPU cards for general computations. Part of this work is devoted to training an acoustic model and also testing it in the "Vystadial" dialog system.

Keywords: ASR,speech recognition, decoder

Contents

TODO: UNIT words with multiple spelling: real-time vs real time, online vs on-line, offine vs off-line

TODO: training: partial data evaluation

TODO: rewrite/deleteBatch interface vs oop on-line interface

TODO: cha:decoder: specify beam, max-active

TODO: add ml over whole thesis

TODO: dialog vs dialogue

TODO: Fix "See Section XY". Tell what one can See in Section XY. "Explain why one should look there".

1. Introduction

A spoken dialog is the most intuitive way of communication among people. The quality of a dialog largely depends on the quality of speech recognition because the reasoning and the answer is based on the recognised speech.

In this work, we build Automatic Speech Recognition for a dialog system called Alex. We use Kaldi speech recognition toolkit[?] for acoustic modeling as well as real-time recognition the textual representation from speech. We see the added value of this thesis in a new training scripts deployed with open acoustic data[?], but also in new C++ interface for speech recognition and its Python wrapper which were integrated into Alex dialogue system. Training scripts, which use free publicly available data, evaluate the quality of trained Acoustic Models. The newly developed speech recogniser is deployed in the dialogue system Alex available at a public toll-free line 800 899 998.

1.1 The problem

The speech recognition in a dialog system closely interacts with a Spoken Language Understanding unit. The Spoken Language Understanding (SLU) unit typically classifies the speech better if the speech recogniser outputs more than one hypothesis for one utterance. A word lattice effectively represents multiple hypothesis, so it is convenient for passing the hypothesis between Automatic Speech Recognition (ASR) and SLU unit.

The Alex dialog system has been using the HTK toolkit[?] and OpenJulius[?] lattice decoderto in order to train acoustic models respectively to decode lattices in real time. Unfortunately, our project members were experiencing crashes of OpenJulius during extracting lattices.

Bearing in mind the OpenJulius's complicated code base and relatively slow development of both HTK and OpenJulius, we were looking for another open source toolkit with a real-time decoder.

We chose the Kaldi toolkit because the Kaldi toolkit deploys modern training recipes, and is actively maintained. Moreover, Kaldi is distributed under the permissive Apache 2.0 license[1]. On the other hand, the kaldi toolkit had no lattice decoder with interface convenient for a dialog system which can process audio stream incrementally.

The Kaldi community focused on improving acoustic model training. In August 2012[2] Kaldi team published a demo version of an on-line one best hypothesis decoder. The on-line decoder inspired us in designing the on-line interface suitable for real-time use of a lattice decoder, and we rewrote its feature preprocessing functionality to fit our interface.

1.2 The goals of the thesis

The goals of the thesis are presented in order as were implemented:

[1] `http://www.apache.org/licenses/LICENSE-2.0`

[2] The changes were introduced by svn commit 1259.

1. Acoustic Models (AMs) are trained to evaluate the new recogniser.
2. The new recogniser is developed so its Python wrapper can be deployed into our dialogue system Alex.
3. Finally, we integrate the recogniser into our Alex Spoken Dialog System (SDS) written in Python and evaluate its performance.

1.2.1 Training acoustic models

A Continuous Speech Recognition decoder requires two pre trained components, an Acoustic Model and a Language Model. We focus on finding the best Acoustic Model for the Kaldi toolkit. The Language Model is changed based on targeted domain.

We aim at developing acoustic model training scripts using the Kaldi toolkit with such quality, that resulting AMs could be compared with the AMs trained with previously used HTK toolkit. The scripts are developed for Czech and English transcribed acoustic data.

1.2.2 Development real-time speech recogniser

We modify Kaldi *LatticeFasterDecoder* decoder in order to allow incremental speech recognition. The resulting incremental interface should be as simple as possible yet allow state-of-the art performance. In addition, we implement such speech parametrisation and feature transformations preprocessing, that high-quality acoustic models can be used. Finally, we compute the posterior probabilities of the word lattice representing multiple ASR hypotheses.

The process of incremental speech recognition is represented by single instance of *OnlineLatgenRecogniser* class, which provides simple interface to speech recognition.

In addition, we suggest potential speed improvements e.g. approximations, use of Graphics Processing Unit (GPU) or Deep Neural Networks (DNN) for speech processing[**?**].

1.2.3 Integration into Alex Spoken Dialog System framework

We interface *OnlineLatgenRecogniser* via its Python wrapper since the dialogue system is written in Python. The *PyOnlineLatgenRecogniser* is a thin wrapper which efficiently exposes the speech recognition interfaces. In addition, we make sure that the lattices can be accessed from Python. The *PyOnlineLatgenRecogniser* is integrated into Alex SDS and the decoding parameters are tuned to obtain best performance. The evaluation of speech recognition setup is important part of the integration.

Thesis outline

In Chapter 2 we introduce a fundamental theory of speech recognition for related areas to our work. In Sections 2.2 and 2.3 we describe alternatives

to Kaldi speech recognition toolkit. At the end of the chapter, we present OpenFST framework which allows the Kaldi library effectively implement many standard speech recognition operations. To obtain high-quality Acoustic Models, we develop training scripts for Czech and English data described in Chapter **??**. In addition, we compare acoustic models trained by Kaldi and previously used HTK toolkit. Chapter 4 presents in detail the new Kaldi real-time recogniser and discuss its on-line properties. We distinguish the original work done by the Kaldi team and our improvements. Then in Chapter 5, we describe deployment of the real-time recogniser into dialogue system Alex, we suggest evaluation criteria and also evaluate the integrated recogniser accordingly. Finally, Chapter 6 summarises the thesis and concludes with future research directions.

2. Background

The statistical methods for continuous speech recognition were established more than 30 years ago. The most popular statistical methods are based on Hidden Markov Model (HMM) Acoustic Models and n-grams Language Models (LMs), which are also used in Kaldi, the toolkit of our choice.

Section 2.1 introduces speech preprocessing, AM and LM training, and explains the principle of speech decoding.

Next sections describe decoding or training models for a particular software. The Kaldi toolkit is described in Section 2.4, the HTK toolkit in Section 2.2 and the Julius decoder in Section 2.3.

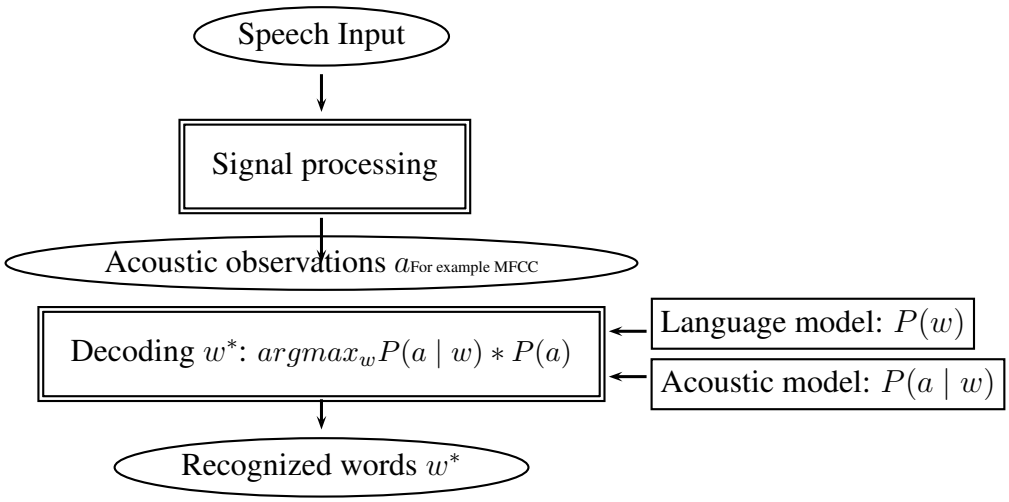


Figure 2.1: Architecture of statistical speech recognizer[?]

2.1 Automatic speech recognition

The goal of statistical ASR is to decode the most likely word sequence given speech. The term *decoding* finds its origin in HMM terminology. In speech recognition it is equivalent to *recognizing* the word sequence from the speech. The speech HMM decoders recognise phones or triphones sequences from which the word sequence is extracted as explained in Subsection 2.1.4. Formally, we search for the most probable sequence of words w^* given the acoustic observations a. See Equation 2.1, which can be simplified to Equation 2.1, because we want decode w^* for fixed speech a.

$$\begin{aligned} w^* = argmax_w\{P(w \mid a)\} &= argmax_w\{\frac{P(a \mid w) * P(w)}{P(a)}\} \\ &= argmax_w\{P(a \mid w) * P(w)\} \end{aligned} \tag{2.1}$$

The task of acoustic modeling is to estimate the parameters θ of a model so the probability $P(a \mid w; \theta)$ is as accurate as possible, which is described in Section 2.1.2. Similarly, the LM represents the probability $P(w)$. We describe language modeling in Section 2.1.3.

frames

The Figure 2.1 illustrate the process of decoding the most probable word hypotheses w^* for given speech utterance. First, the sampled audio signal is processed by speech parametrisation and feature transformations, so the decoding itself takes acoustic features a as input. The signal processing is computed on small overlapping windows of audio signal, typically denoted as frames. The decoding itself is performed time synchronously for each frame using beam search. The beam search expands hypotheses from previous step by taking account the new frame features, and it computes probabilities of the expanded hypotheses using AM and LM. If the number of hypotheses exceeds the beam, the low probable hypotheses are pruned. Pruning the least probable hypotheses and the described simple beam search may lead to discarding globally optimal hypotheses. After the decoding the last audio frame, all hypotheses represents whole utterance. The word labels w^* are extracted from the most probably hypothesis which survived the beam search.

Improving the accuracy of speech recognition engine depends mainly on improving AM and LM and also on parameters of the beam search such as threshold how many hypotheses are allowed at maximum.

2.1.1 Speech parameterisation

The goal of speech parameterization is to reduce the negative environmental influences on speech recognition. The speech varies in a number of aspects. Some of them are listed below:

- Differences among speakers pronunciation depends on gender, dialect, voice, etc.
- Environmental noises. In the application for a dialogue system the typical speech is recorded on a noisy street.
- The recorded channel. For example the telephone signal is reduced to frequency band between 300 to 3000Hz. The quality of mobile phone signal also influences the quality of the audio signal.

Different speech parametrisation may improve robustness of speech recognition in different recording conditions.

acoustic features

Speech parametrisation extracts speech-distinctive acoustic features from raw waveform. The two most successful methods for speech parametrisation in last decades are Mel Frequency Cepstral Coefficients (MFCC)[**?**] and Perceptual Linear Prediction (PLP)[**?**]. The methods are very computationally effective and significantly improve the quality of recognised speech.

The toolkits used in our dialogue system, Kaldi and HTK toolkit, compute MFCC coefficients for given audio input similarly.[1] We choose MFCC

[1]The subtle differences are caused by implementation approaches, but does not effect the quality of MFCC coefficients in significant way.

as speech parametrisation technique for both toolkits, so we can compare them.

Both MFCC and PLP transformations are applied on a sampled and quantized audio signal. The sampling frequency is typically 8000, 16000 or 32000 Hz. Each sample is usually encoded to 8, 16 or 32 bits. In our experiments we use 16 kHz sampling frequency for 16 bit samples.

feature window

The MFCC or PLP statistics are computed from overlapping windows of quantized audio signal. In Figure 2.2 there are 7 windows for audio of length 85 ms and window shift and length of 10 ms and 25 ms.

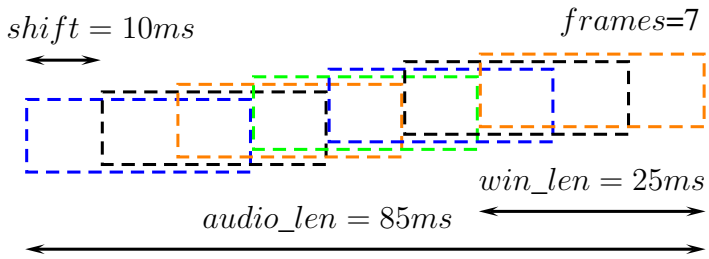


Figure 2.2: PLP or MFCC features are computed every 10 ms seconds in 25 ms windows. Audio length is $(frames - 1) * shift + win\_len = 85ms$

For each window MFCC or PLP efficiently computes statistics with a reduced dimension. Let us describe the MFCC computation for 25 ms window shifted by 10 ms and 16kHz audio sampling frequency. The $16000 * 0.025 = 400$ samples in one window are reduced to 13 static cepstral coefficients.

The MFCC or PLP static features are usually extended by time derivatives $\Delta + \Delta\Delta$ features [**?**], so it results to $13 + 13 + 13 = 39$ MFCC $\Delta + \Delta\Delta$ features per 400 samples in one window.

The MFCC features are computed by the following steps:

1. The audio samples are transformed into *frequency domain* by Discrete Fourier Transformation (DFT) in the window.

2. The frequency spectrum from the previous step is transformed onto the mel scale, using triangular overlapping filters.

3. From the mel frequencies the logs of the powers are taken from each of the mel frequencies.

4. At the end the discrete cosine transform is applied on the list of mel log powers.

5. The MFCC coefficients are the amplitudes of the resulting spectrum.

6. The $\Delta + \Delta\Delta$ coefficients are computed from the current and previous static features. See Figure 2.3.

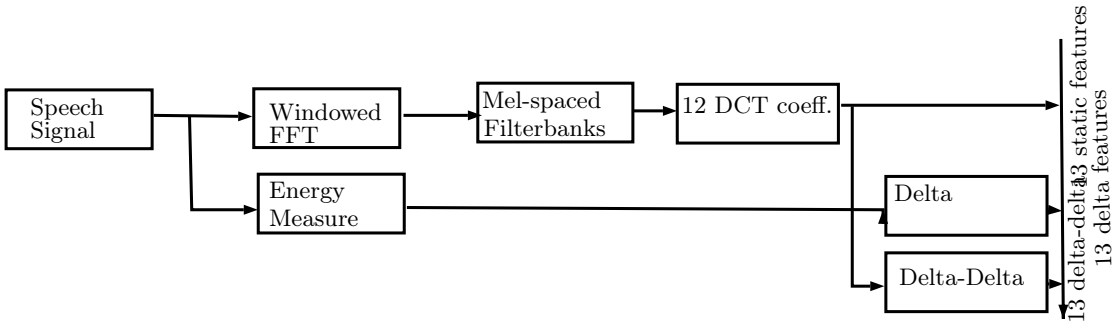


Figure 2.3: Typical setup with 39 features using MFCC.

Feature space transformations

Feature space transformations are usually applied in addition to MFCC or PLP parametrisation. The feature space transformations are typically applied per frame. Each feature vector extracted from MFCC or PLP window - frame, is projected to another feature space. Transformations usually take into account context of several preceding (left context) and consecutive frames (right context). *frame*

The linear or affine transforms are expressed by matrix multiplications Ax respectively Ax^+. The matrix A represents the transformation. The x is the input vector and Ax are the transformed features. The affine transformations uses extended vector $(x^+)^T = (x_1, \ldots, x_n, 1)$ and matrix $A : (n+1) * (n+1)$.

There is a large variety of available transforms and dependent on acoustic data one should choose the most appropriate one. Some transforms are estimated discriminatively, some use generative models. Some are speaker dependent, some speaker independent.

We list some of Kaldi transforms in order to illustrate rich choice of feature transforms in Kaldi toolkit.

- Heteroscedastic Linear Discriminant Analysis (HLDA)[**?**].
- Linear Discriminant Analysis (LDA)[**?**] is typically used with MLLT for speaker independent training.
- Maximum Likelihood Linear Transform (MLLT) also known as Semi-Tied Covariance (STC)[**?**]
- Exponential Transform (ET)[**?**]. It uses small number of speaker specific parameters for adaptation on speaker.
- Cepstral Mean and Variance Normalisation (CMVN)[**?**]. Typically normalise the cepstrum mean and variance per speaker.

In our acoustic modelling scripts, see Chapter 3, we use two non-speaker adaptive feature transformations, which can be computed with very small context. The first transformation, $\Delta + \Delta\Delta$ for MFCC coefficients, was already introduced. The second transformation, LDA and MLLT, is described briefly below, and we use it also with MFCC coefficients.

Linear Discriminant Analysis and MLLT feature transformation

The LDA+MLLT is an alternative setup to $\Delta+\Delta\Delta$ transformation in our training scripts. Using several spliced MFCC vectors the LDA+MLLT searches for the best dynamic transformation, replacing the $\Delta + \Delta\Delta$ coefficients.

The combination of LDA and MLLT applies the feature transformation in two steps: LDA reduces the feature dimension and MLLT rotates the feature space so the classes after LDA has more diagonal covariance in two steps.[?] Whereas, the HLDA performs dimension reduction and space transformation in one step.[?] The combination LDA and MLLT performs very similar feature transformation to HLDA and gains significant improvements over $\Delta + \Delta\Delta$ transformation as HLDA[?][?].

2.1.2 Acoustic modeling

Acoustic modeling is arguably the heart of the speech recognition. More realistic modeling of acoustic features directly affects the speech recognition quality as seen in Equation 2.1. The AM estimates the probability $P(a|w;\theta)$ of generating acoustic features a for a given word w.

The acoustic modeling has only partial information available for training AM parameters θ. Let us mention that acoustic features a are collected every 10 ms. However, the corresponding textual transcription is time-unaligned. The hidden information of words time alignment in a sentence makes acoustic model training more challenging.

Modern speech recognition toolkits use Hidden Markov Model for modeling uncertainty between acoustic features and the corresponding transcription.

Choice of training units

The most successful acoustic modeling methods do not estimate the $P(a|w)$ directly, but estimate probability $P(a|f_1f_2f_3)$ of generating acoustic features a for triphone $f_1f_2f_3$.

Phone is the smallest contrastive unit of speech. Let us see few examples of words and their phonetic transcriptions according CMU dictionary[?]. *phoneme*

- *youngest* & *Y AH1 NG G AH0 S T*
- *youngman* & *Y AH1 NG M AE2 N*
- *earned* & *ER1 N D*
- *ear* with two transcribed pronunciations *IY1 R* and *IH1 R*

The CMU dictionary distinguishes among several variations for each vowel e.g. *AH1* and *AH0*. It also stores two possible pronunciations for the word *ear*.

The acoustic features for a phone significantly depend on the context. The previous and the following phone strongly influence the sound of the phone in the middle.

The triphone is a sequence of three phonemes and captures the context of single phone. See 2.4. As a result, acoustic properties of the triphones vary much less according to the context than phonemes. Let us note that certain combinations of prefixes have the same effect on the central phoneme, e.g. q and k has the same effect on i. In order to reduce the number of triphones, these triphones are clustered together. *triphone*

Hidden Markov Models (HMMs)

The HMMs is a very powerful statistical method for characterizing observed data samples of a discrete-time series with an unknown state. [**?**]. In case of speech recognition the hidden states typically represent monophones or triphones and we observe samples of the acoustic features.

transition probability

Hidden Markov Models have two type of parameters *transition probabilities among states* and *probabilistic distribution for generating observation in given state.* The transition probability is a probability of changing state q to state u. Each transition can be represented as arc $e = qu$ between the states q and u, see 2.4. The probability is often represented as the weight w_e of arc e. For every node in a Markov model must hold that the sum weights of outgoing arcs is one.

Gaussian HMM

The Markov model emits an observation during traversal over its arcs. The Hidden Markov Model emits the observation stochastically based on the probabilistic distribution related to the visited state. In speech recognition, a multivariate Gaussian distribution is typically used to model observation probabilities of HMM states.

Training HMM

The Kaldi uses Viterbi training and the HTK toolkit uses Expectation Maximization algorithms to train HMM Acoustic Model. The toolkits models the observation probabilities using multivariate Gaussian distribution with dimension of the acoustic features a. Both Viterbi training and the Expectation Maximization (EM) algorithm starts with HMM with initial values.

Typically, the transition probabilities follow uniform distribution. The observation probabilities are usually initialized by multivariate Gaussian distribution with μ and Σ set to global mean and global covariance matrix estimated on all training acoustic data.

EM

Let us describe how the EM algorithm operates for one pair of training data consisting of acoustic features a extracted from speech and corresponding text speech transcription t. We create HMM t', where each state represents one triphone. The triphones are extracted from transcription t using pronunciation dictionary. In Figure **??** the transcription *how do you do* was the expanded HMM model representing triphone sequence. It should be obvious that only parameters of HMM states occurring in t' can be updated from pair (a, t).

The EM algorithm iterates following steps in order to update parameters of transition and observation probabilities:

- The observation probabilities are computed using HMM t'.
- **E-step**: Based on the observation probabilities the observation are assigned to states of HMM t'.
- **M-step**: Based on assignment of observation to states the t' parameters are re-estimated.

Baum-Welsch

The **E-step** finds the most probable distribution as a weighted combination of distributions which map acoustic observation to HMM states using Maximum Likelihood Estimation (MLE)[**?**]. The Baum-Welsch equations can be derived from the fact, that the MLE criterion is used for finding the most probable distribution in **M-step**.[**?**]

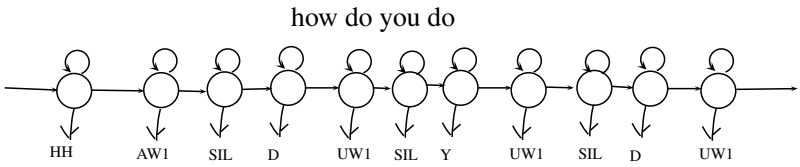


Figure 2.4: Markov monophone model for three words. Such an HMM is constructed for monophone Viterbi training and reference transcriptions *how do you do*. The parameters of the HMM model are updated according Equation 2.9, 2.10 and 2.11.

Maximum Likelihood Estimation method

The MLE is general approach to setting statistical model parameters. The maximum likelihood estimation search for best parameters θ^* in order to maximize the likelihood function f for Independent and Identically Distributed (IID) training data illustrated in Equation 2.4. For IID training data holds Equation 2.2 describing data joint probability.

$$f(x_1, x_2, x_3, \ldots, x_n|\theta) = f(x_1|\theta) * f(x_2|\theta) * \ldots * f(x_n|\theta) \tag{2.2}$$

The likelihood function can be derived from Equation 2.2 assuming training data fixed and parameter θ free as described in Equation 2.3.

$$\mathcal{L}(\theta \,|\, x_1, \ldots, x_n) = \sum_{i=1}^{n} log(f(x_i|\theta)) \tag{2.3}$$

$$\theta^* = argmax_\theta \mathcal{L}(\theta \,|\, x_1, \ldots, x_n) \tag{2.4}$$

Viterbi training of acoustic models

On the other, hand the Kaldi toolkit applies the Viterbi criterion in assigning the acoustic observation to HMM states. The Viterbi training approximates EM AM training by choosing single best alignment and maximizing the posterior probability for the chosen alignment. Latest work suggest that Viterbi training is just as effective for continuous speech recognition as Baum-Welch algorithm [**?**]. Moreover, Viterbi training needs much less computational resources.

We details the Viterbi training since it is used in the Kaldi toolkit for acoustic model training. *TODO: In addition, with minor modifications a Viterbi algorithm is used also for decoding in Kaldi.*

Given set of training observations $O^r, 1 \leq r \leq R$ and HMM state sequence $1 < j < N$ the observation sequence is aligned to the state sequence via Viterbi alignment.[**?**] The best alignment T results from maximising Equation 2.5 for $1 < i < N$.

$$\theta_N(T) = max_i[\theta_i(T)a_{iN}] \tag{2.5}$$

TODO: describe T and T versus o_t *TODO: Should I cite every equation?*

The $\theta_i(o_t)$ from Equation 2.5 is computed recursively according Equation 2.6

$$\theta_i(o_t) = b_j(o_t)max \begin{cases} \theta_j(t-1)ajj \\ \theta_{j-1}(t-1)aj-1j \end{cases} \tag{2.6}$$

The initial conditions are $\theta_1(1) = 1$ and $\theta_j(1) = a_{1j}b_j(o_1)$, for $1 < j < N$. In our case the likelihoods are modeled as mixture Gaussian densities, so the output probability $b_j(o_t)$ is defined as in Equation 2.7.

$$b_j(o_t) = \sum_{m=1}^{M_j} c_{jm}\mathcal{N}(o_t; \mu_{jm}, \Sigma_{jm}) \tag{2.7}$$

The M_j represents number of mixture components in state j, c_{jm} is the weight of m^{th} component and $\mathcal{N}(o_t; \mu_{jm}, \Sigma_{jm})$ is multivariate Gausian with mean vector μ and covariance Σ.

Firstly, model parameters are updated based on the single-best alignment of individual observation to states and Gaussian components within states. Secondly, transition probabilities are estimated from the relative frequencies, Equation 2.8 where A_{ij} denotes the number of transitions from state i to state j.

$$_{ij} = \frac{A_{ij}}{\sum_{k=2}^{N} A_{ik}} \tag{2.8}$$

The indicator function $\theta^r_{jm}(t)$ is used for updating means and covariance matrix from statistics. It returns one if o^r_t is *TODO: associated* with mixture component m of state j and is zero otherwise. The mean vector and covariance matrix is updated according Equations 2.9 and 2.10.

$$todo \tag{2.9}$$

$$todo \tag{2.10}$$

Finally, the mixture weights are computed based on the number of observations allocated to each component.

$$todo \tag{2.11}$$

generative training

To conclude, AM are trained using MLE or Viterbi training, which approximates the theoretically optimal MLE Baum-Welsh training; however, in practice Viterbi training performs as well as MLE modelling. Baum-Welsch or Viterbi training aim at modelling the probabilities of spoken utterance and perform so called generative training. However, in practice, discriminative methods, which optimise its objective function, perform better.

Discriminative training

discriminative training

The discriminative training uses its objective function and posterior probability to discriminate – select the best hypothesis. The discriminative training is typically initialised by acoustic model by a generative model from Baum-Welsch or Viterbi training. In training scripts described in Chapter 3 following objective functions and accordingly named discriminative training are used:

- Maximum Mutual Information[**?**]
- Boosted Maximum Mutual Information[**?**]
- Minimum Phone Error[**?**]

2.1.3 Language modeling

LM effectively reduces and more importantly priorities the AM hypothesis. The statistical LM assigns a given word sequence its probability. The probability of a word transcription from AM is combined with the probability of the word transcription from LM.

The invalid words sequences or the sequences not frequent in training data are estimated with a low probability. On the other hand, the frequent sequences are assigned with high probability.

The word sequence "*are few born new york*" should be rewarded with a lower probability from LM than the alternative with similar phonetic representation "*are you from new york*".

Note that in our settings the AM already limits the possible triphone sequences to sequences present in lexicon words - the words in training data. So the mapping from acoustic features a to triphone sequences are restricted to sequences, which form probably word sequences.

In practice it is unfeasible to compute the probability of k word sequence W according Equation 2.12. In addition, it would require enormous amount of data for estimating probability for $k > 3$.

$$P(W) = P(w_i, w_{i-1}, w_{i-2}, ..., w_1) = \prod_{i=1}^{k} P(w_i | w_{i-1}, w_{i-2}, w_1) \tag{2.12}$$

LM order

We call the number k order of LM. In speech recognition we typically use the order of LM limited to $k < 4$. The Equation 2.13 describes how the trigram LM should be estimated. The n-gram is a sequence of n consecutive words. The probability of n-gram can be estimated by language models of order n and greater.

The LM estimates the probability of an n-gram t by counting the frequency in text corpus of t and also (n-1), (n-2), .. n-grams, which are subsequences of t. The text corpus is typically chosen, so it corresponds to a domain, in which the ASR system with the LM should be used.

LM smoothing

We call the *smoothing* technique estimating the probability of unseen n-grams t based on n-grams with lower order which form the original sequence t. The technique is widely used, because there is much greater chance that lower order n-grams are present in training data than n-gram with high order. For example, in our training scripts if no external LM is supplied, we train the LM only on text transcriptions from the training data and we use Witten-Bell smoothing.[**?**]

$$P(w_i \mid w_{i-1}, w_{i-2}, ..., w_1) = P(w_i \mid w_{i-1}, w_{i-2}) * P(w_{i-1} \mid w_{i-2}) * P(w_{i-2}) \tag{2.13}$$

2.1.4 Speech decoding

The Continuous Speech Recognition (CSR) is a pattern recognition task as well as a search problem. In speech recognition, making a search decision is also referred to as decoding.[**?**]

For isolated word recognition the HMMs are trained for each word. Using the forward algorithm for each HMM h representing a word w, we are able to compute the probability of every w given the acoustic observations a. The isolated word recognition becomes a simple recognition problem, where we select the most probable HMM h^* from a finite set of HMMs, which parameters were trained either by EM algorithm as described in Subsection 2.1.2.

Note that the HMM training is identical for CSR and isolated word recognition, but the decoding is more complicated for CSR.

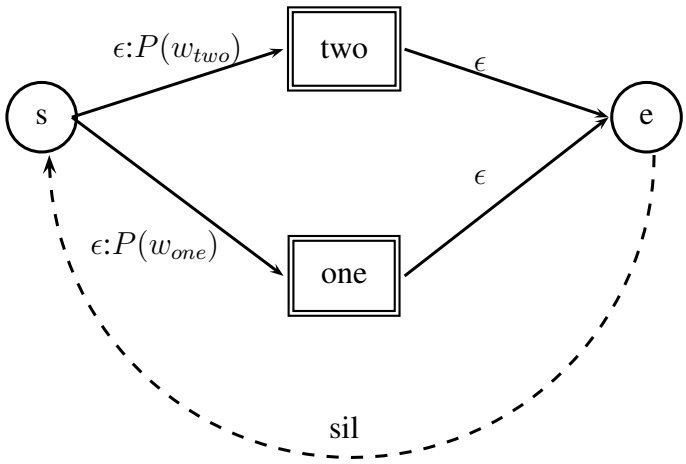


Figure 2.5: Diagram of how LM is combined with HMMs.

Imagine we have a LM order 1 of only two words - *one* and *two* each uniformly distributed[2]. We want to decode any possible sequence of words *one*, *two*, so we add additional ϵ transition at the end of words HMMs. The words are chained using silence HMM model as illustrated in Figure 2.5. Figure 2.6 shows expanded HMMs, which have ϵ transitions at the beginning and the end of the words. Note that LM weight $P(w)$ can be stored on the in the ϵ transition at the beginning.

Even the simple HMM network in Figure 2.6 can become large search problem partially because the search space of words grows exponentially in number of words in utterance and partially because the word boundaries are unknown. Each word can begin in any moment and last with decreasing probability ad infinity, so the search space explode. In our trivial case we skip problems with higher order LMs, silence modeling and large lexicon.

One can see that the search space of speech recognition problem with large vocabulary is enormous and has to solved in very short time.

Natural choice for one-best hypothesis is Viterbi beam search[**?**]. We already described Viterbi algorithm in Subsection 2.1.2. The beam is used to limit search space in each iteration of Viterbi algorithm. The Viterbi algorithm is breadth-first search algorithm and the beam limits number of nodes, which are expanded from current set of nodes to next iteration.

We list few alternatives how to set up the beam for speech decoding.

- *Fixed beam* guarantees maximum size of memory footprint and fast decoding.
- *One-best hypothesis comparison* effectively discards most of the improbably hypothesis if the one-best hypothesis is significantly better than alternatives and keeps lot of alternatives if one-best hypothesis is weak. The relative one-best hypothesis comparison naturally broaden the beam in uncertain region, but does not guarantee no hard limits.
- *Combination* of methods applies the strictest criteria on beam in each iteration.

[2]If a LM of order 1 assigns to every word equal probability, we say it has order 0

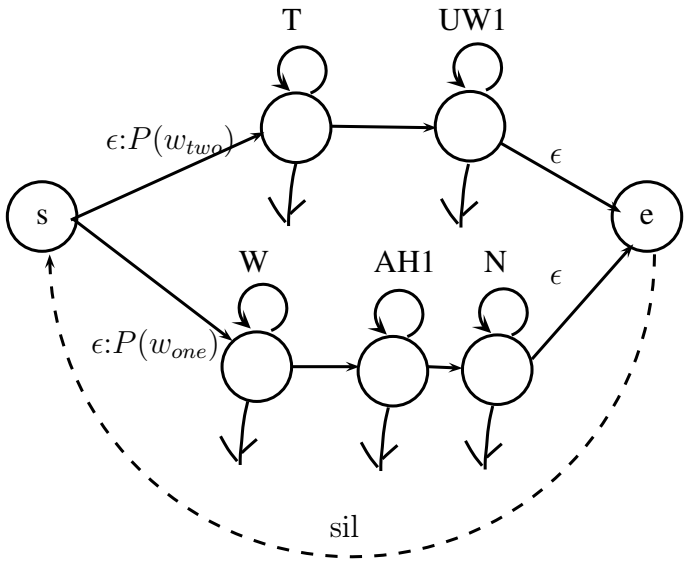


Figure 2.6: Expanded HMMs for words *one* and *two*. The arrows at HMM states illustrate that every observation of acoustic features can be generated according to the statistical distribution. Note that if a speaker says *two* a HMM model with well trained parameters should output higher probability for HMM representing *two*. For longer speaker sequences e.g. *two one one two ...* the HMMs are connected over the ϵ transitions, and a search is used for selecting the most probable sequence.

So far we focused on combination of HMM AM and LM and its graph representation, but also mixing the probabilities of AM and LM according Equation 2.1 is not straightforward in practice. Due to numerous assumption violation for computing AM as well as LM it proves to efficient to use weighted product of the two models. Typically, language model weight is used w_{lm}.

$$w^* = argmax_w\{P(w \mid a)\} = argmax_w\{P(a \mid w) * P(w)^{w_{lm}}\} \tag{2.14}$$

As you can imagine the path of phones in the search graph become rather large easily and even with a beam search lot of hypothesis are assign with tiny probabilities. In order to keep the numeric stability, the probabilities are computed in log scale. We are searching for the most probable path in a graph, however the typical search task for is shortest path problem. In order to use the shortest distance measure to find the most probably path and avoid multiplication we use formula **??** derived from equation 2.15. The $C(a \mid w)$ and $C(w)$ are costs with range between zero and one, where cost of one corresponds to zero probability $C(1) \cong P(0)$ and cost of infinity corresponds to one probability $C(\infty) \cong P(1)$.

$$\begin{aligned} w^* &= argmin_w\{log(\frac{1}{C(a \mid w) * C(w)^{w_{lm}}})\} \\ &= argmin_w\{-log(C(a \mid w) * C(w)^{w_{lm}})\} \\ &= argmin_w\{-log(C(a \mid w)) - w_{lm} * log(C(w))\} \end{aligned} \tag{2.15}$$

Decoding formats

Formats which can represent also alternative hypothesis provide better results for further processing than one-best hypothesis because the alternatives contain missing information.

n-best list

We present n-best list and lattice formats, which both are able to represent alternative hypothesis.

N-best list is an extension to the one-best hypothesis. In n-best list is included apart from the most probable word sequence, also the second, third, ..., n-th most probable hypothesis.

```
0.5 hi how are you
0.2 hi where are you
0.1 bey how are you
```

Figure 2.7: Toy example of 3-best list output with posterior probability for each path.

The n-best list and lattice are extracted from the beam history. Note that the Kaldi toolkit always generates state level lattices [**?**][3]. The word lattice and later word n-best list are extracted from the state level lattice where the states roughly corresponds to triphones.

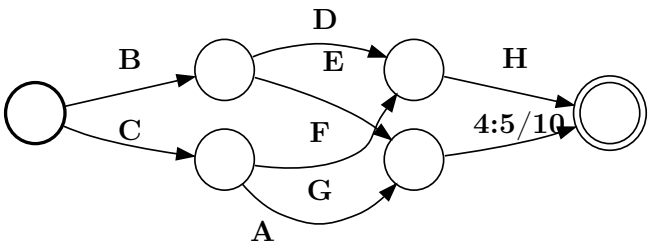


Figure 2.8: Small lattice. *TODO: TODO draw it with Ipython*

It is convenient to output with each hypothesis a measure so one can see how good the hypothesis is in contrast to its alternatives or in absolute measure. Typically likelihood and posterior probability is associated with each word sequence. The likelihood measure can be used only for relative comparison, whereas posterior probability is an absolute measure since because it is normalised to one. For some applications the likelihood measure is sufficient, other applications for example dialogue systems prefer posterior probabilities. Note that the posterior probabilities of 3-best list on Figure 2.7 do not sum to one, because n-best list omits some hypothesis which are used to compute the posterior probability.

2.1.5 Evaluating Automatic Speech Recognition quality

WER

The quality of speech recognizer is typically measured using Word Error Rate. The Word Error Rate (WER) measure is computed on pairs of ASR hypotheses $(decoded(a), t)$ and their human transcriptions t. The expression $decoded(a)$ in this section denotes ASR hypothesis in one-best format as described in Subsection 2.1.4.

[3]State level lattices uses states very similar to triphone states

The pair (a, t) is not supposed to be used in AM training because we are testing the ability to decode unknown speech. We should also measure the ASR quality on speech from a speaker, who does not appear in speech data used for AM training because we usually want to decode speech of an unheard speaker.

The WER is computed as a minimum edit distance on words between pair $decoded(a), t$ typically using edit operations *substitution, deletion, insertion* as described in 2.16. The effective implementation for computing WER uses dynamic programing and is not computationally intensive, because ASR hypotheses are typically quite short.

$$WER = 100 * \frac{min\_dist(decoded_{AM,LM}(a), t, edit\_operation = \{Subs, Del, Ins\})}{\#\ words\ in\ t} \tag{2.16}$$

Note that WER is an error function so the ideal value is zero, because for $WER = 0$ the hypothesis $decoded(a)$ and the reference transcription t are identical. The WER 100 for $decoded(a)$ and reference t with equal number of words show that every single word is different between $decoded(a)$ and reference t. Despite the fact that WER resembles percentage format, it can be bigger than 100. See Figure 2.9.

reference

```
decoded(a) = 'hi hi hi hi'
t='hello'
WER = 100 * ( 4 / 1) = 400
```

Figure 2.9: The WER measure can be greater than 100.

Alternative measures

The Sentence Error Rate measures how many decoded utterances $decoded(a)$ match exactly its reference t for all pairs (a, t) in test set T.

SER

$$SER = \frac{\sum_{\{(a,t) \in T; decoded(a) = t\}} 1}{|T|} \tag{2.17}$$

If we are using an alternative output format to one best hypothesis as n-best list or lattice, we are extracting one-one best hypothesis format for measuring WER or Sentence Error Rate (SER). On the other hand, we are using n-best lists or lattice, because the one-best hypothesis might be wrong and the alternative hypothesis may be closer to the reference.

We do not evaluate any other measurement except for WER and SER, but note that richness and correctness of alternatives are desired qualities. The alternatives may contain additional information and are for example used in a dialogue system Spoken Language Understanding unit which parses ASR output.

Measuring speed

In this thesis we are especially concerned about the speed of speech decoding, because the implemented decoder is used in a Spoken Dialog System.

A very natural measure of the speed for speech decoding is Real Time Factor, which expresses how much the recognizer decodes faster than the user speaks. We

Real Time Factor

measure the Real Time Factor (RTF) for each recording as described in Equation 2.18.

$$RTF = \frac{time(decode(a))}{length(a)} \tag{2.18}$$

For real-time decoding in a dialogue system we need $RTF < 1.0$ for all tested utterances a. In other words, the decoding of each utterance a should take less time than playing the utterance in a music player. With $RTF < 1.0$ we can decode faster than the user speaks, so the hypothesis is ready immediately after the user finishes the speech.

Note that in our decoder we have two phases of decoding. The forward decoding is performed as user speaks, but the backward decoding is triggered at the very end of the speech as described in Subsection 2.1.4. The users have to wait at least for the time when backward decoding is performed, so we measure Forward Real Time Factor (FWRTF) for forward decoding.

latency In real-time SDS the critical measure is a delay how long the user has to wait for its answer. The latency measures the time between the end of the user speech and the time when a decoder returns the hypothesis, which is the most important speed measure for ASR component in SDS. Note if $FWRTF < 1.0$ in our decoder then the latency is the time of backward decoding.

2.2 HTK

The HTK toolkit is a set of command line tools, sample scripts and library for training and decoding HMM focused on speech recognition. HTK uses Baum-Welch algorithm for HMM parameters estimation. With the toolkit are distributed two decoders *HVite* and *HDecode*, which are not designed for real-time applications. *HVite* can be used only with unigram or bigram LM. The *HDecode* decoder handles also a trigram LM.

The functionality of the library is wrapped by command line programs. The programs are typically combined in training scripts to train acoustic and language models. In Figure 2.10 the acoustic models are labelled as "HMMs" and the language models in HTK are represented in "Networks". The trained models are used in one of HTK decoders e.g. *HVite* for decoding transcriptions, which can be evaluated using *HResults*.

The HTK use Baum-Welch algorithm to train acoustic models. The *HVite* decoder uses token passing algorithm and Viterbi criterion. [**?**] Only bigram LM can be used with *HVite*. The
termHDecode decoder can handle bigram or trigram language models.

Let us stress that we use high quality Bash and Perl scripts for training HTK AM from Vertanen improved by Matěj Korvas.[**?**][**?**]

The HTK toolkit is licensed under a special license[4]. The *HDecode* has very similar license condition but can be only used for research purposes.[5]

[4] `http://htk.eng.cam.ac.uk/docs/license.shtml`

[5] You need to register even to see the license: `http://htk.eng.cam.ac.uk/prot-docs/hdecode_register.shtml`

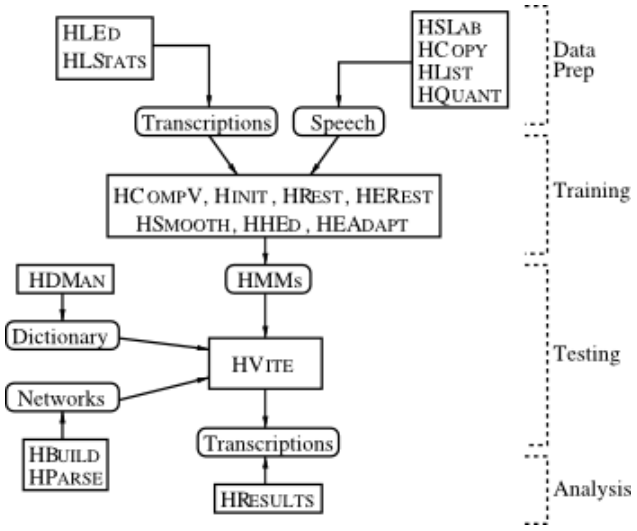


Figure 2.10: Figure 2.2 from HTK Book 3.4[?]

2.3 Julius decoding engine

Julius is a large vocabulary continuous speech engine, which can use AM in HTK format for decoding.[?] Julius is BSD licensed[6] and performs almost real-time decoding.

Julius is a two pass decoder. In the first pass, the forward decoding is performed using time synchronous beam search. The second pass reranks and further prunes the extracted hypothesis from thethe pass one. The second pass is also known as backward decoding. Bigram LM is used for forward decoding and more complex trigram LM is used for backward decoding.

Before the implementation of this thesis was finished the team of Alex had been interested in Julius, because its ability of real-time decoding and confusion network[7] output format.

The Alex team abandoned the Julius decoder for software issues with crashes of the decoder. The crashes appeared during backward decoding and extracting the confusion network from Julius. In addition, the crashes were hard to detect, because Julius used to run in a separate process.

2.4 Kaldi

Kaldi is a speech recognition toolkit consisting of a library, command line programs, scripts for focus on acoustic modeling. Kaldi deploys several decoders for evaluation Kaldi AM. Kaldi uses Viterbi training for estimating AM. Only in special cases of speaker adaptive discriminative training is also used extended Baum-Welsch algorithm[?].

The architecture of the Kaldi toolkit could be separated to Kaldi library and training scripts. The scripts access the functionality of Kaldi library through command line programs. The C++ Kaldi library is based on the *OpenFST*[?] library and it uses optimized libraries for linear algebra such as BLAS and LAPACK. Related functionality is usually grouped in one namespace in C++ code, which

[6] `http://www.linfo.org/bsdlicense.html`

[7] Confusion network is approximation of lattice described in Subsection **??**

corresponds to one directory on file system. The examples of the namespaces or directories can be seen in Figure 2.11

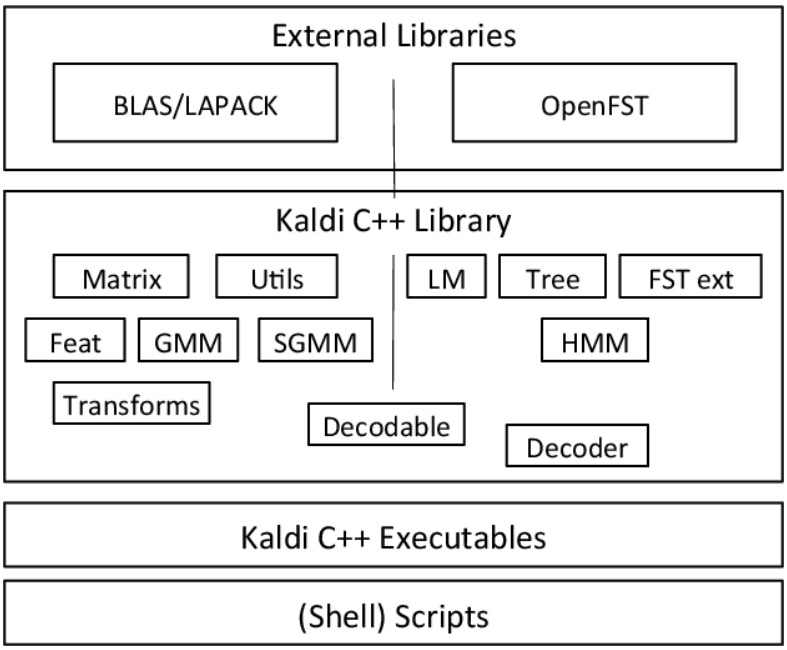


Figure 2.11: Kaldi toolkit architecture[?]

Similarly Kaldi uses binaries which load its input from files and typically store results again to files. Alternatively, the output of one command can be feed into next command using system pipes. In contrast to HTK, there are many alternatives as seen in list of executables below:

1. Speech parametrisation
   - *apply-mfcc*
   - *compute-mfcc-feats*
   - *compute-plp-feats*
   - ...
2. Feature transformation
   - *apply-cmvn*
   - *compute-cmvn-stats*
   - *acc-lda*
   - *fmpe-apply-transform*
   - ...
3. Decoders
   - *gmm-latgen-faster*
   - *gmm-latgen-faster-parallel*
   - *gmm-latgen-biglm-faster*

- ...

4. Evaluation and utilities
   - compute-wer
   - show-alignments
   - ...

In addition, Kaldi provides very useful standardized scripts which wrap Kaldi executables or add a useful functionality. The scripts are located in *utils* and *steps* directories and are used in many training script recipes for different corpus data. In this thesis we created a new training recipe using the Kaldi infrastructure and Czech and English training corpus [**?**]. The recipe, the data and acoustic modeling scripts are described in Chapter 3.

2.4.1 Finite State Transducers

The Finite State Transducer framework and its implementation OpenFST determine the shape of the Kaldi data structures. Kaldi uses Finite State Transducer (FST) as underlaying representation for LM, partially for AM, lexicon and also for representing transformation between text, pronunciation and triphones.

The FST framework provides well studied graph operations[**?**], which can be effectively used for acoustic modeling. Using the FST framework the speech decoding task is expressed as a beam search in a graph, which is well studied problem.

The OpenFST library implements memory efficient representation of FST and provides standardized efficient operations. As stated in [**?**] and [**?**] the operations can be effectively used for speech decoding.

The FST graphs used for AM model training and speech decoding can be constructed as sequence of standardized OpenFST operations.[**?**]. Decoding is performed using a final result of training, so called *decoding graph* $HCLG$.

$$HCLG = H \circ C \circ L \circ G \tag{2.19}$$

.

The symbol $\circ$ represents an associative binary operation of composition on FSTs. Namely, the transducers appearing in Equation 2.19 are:

1. G is an acceptor that encodes the grammar or language model.
2. L is the lexicon. Its input symbols are phones. Its output symbols are words.
3. C represents the relationship between context-dependent phones on input and phones on output.
4. H contains the HMM definitions, that take as input id number of Probability Density Functions (PDFs) and return context-dependent phones.

Following one liner illustrates how Kaldi decoding graph is created using standard FST operations[8].[**?**]

$$HCLG = asl(min(rds(det(H'omin(det(Comin(det(LoG))))))))) \tag{2.20}$$

Most of the operation used in speech recognition operates on paths in the *Semiring*

[8]Kaldi tutorial on building $HCLG$: `http://kaldi.sourceforge.net/graph_recipe_test.html`

decoding graph. Path is a sequence of edges which has weight and an input and an output label. Based on the weight type and weight path operations we distinguish several semirigns.

Formally, a *semiring* $(\mathcal{K}, \oplus, \otimes, \bar{0}, \bar{1})$ is an algebraic structure on set $\mathcal{K}$ with operations $\oplus$ and $\otimes$. The binary operations multiplication $\oplus$ and addition $\otimes$ have identity element $\bar{0}$ respectively $\bar{1}$. The $(\mathcal{K}, \oplus)$ forms commutative monoid and $(\mathcal{K}, \otimes)$ forms just a monoid. The multiplication is left and right distributive over addition. Moreover, multiplication by $\bar{0}$ annihilates any member of $\mathcal{K}$ to *zero*. Table 2.4.1 shows useful semirings in OpenFST.

Name	$\mathcal{K}$	$\oplus$	$\otimes$	$\bar{0}$	$\bar{1}$
Real	$[0, \infty)$	$+$	$*$	0	1
Log	$(-\infty, \infty)$	$-log(e^{-x} + e^{-y})$	$+$	∞	0
Tropical	$(-\infty, \infty)$	min	$+$	∞	0

Table 2.1: Semirings used in speech recognition.[**?**]

3. Acoustic model training

This chapter presents new Kaldi acoustic modeling recipe for free Czech and English "*Vystadial*" data. The recipe scripts were developed as part of this thesis, they are licensed under the Apache 2.0 license and are publicly available in Kaldi repository[1]. The AM trained using these scripts can be used for both batch speech recognition with common Kaldi decoders and for our *OnlineLatgenRecognizer*, which performs on-line decoding described in Chapter 4.

The first Section 3.1 describes used data. The chapter continues by presenting the AMs training in Section 3.2. Later, in Section 3.3 we evaluate trained AMs and also compare them to generative HTK AMs which are trained using state of art HTK scripts. Note, that the details about launching the scripts and file system organisation can be found in Appendix **??**. The description and references to methods which setup we describe below can be found in Section 2.1.

3.1 Vystadial acoustic data

The data were collected in Vystadial project[2] and are released under the Creative Commons Share-alike (CC-BY-SA 3.0) license. The Czech[3]and English[4] data are available online in Lindat repository[5][6].

The English acoustic data consists of recorded phone calls among humans and the Spoken Dialog System, which was designed to provide the user with information on a suitable dining venue in the town. Most of the data was spoken in American English. The typical sentences recorded from users were queries for the dialogue system e.g.,

```
I NEED A CHINESE TAKE AWAY RESTAURANT IN THE CHEAP PRICE RANGE
I'M LOOKING FOR AN INTERNATIONAL RESTAURANT
I NEED TO FIND A PUB IT SHOULD ALLOW CHILDREN AND HAVE A TELEVISION
```

On the other hand, the Czech recordings were collected in three different ways[**?**]:

1. using a free Call Friend phone service
2. using the Repeat After Me speech data collecting process,
3. from the telephone interactions with the Alex SDS in a public transport domain.

In the Call Friend service native Czech speakers were invited to make free calls. In Repeat After Me process volunteers called a number where they were asked to repeat sentences synthesized by a Text to Speech (TTS).

The user language differs significantly in dialogues with Alex system and the other two settings. The sentences in Alex public transport domain, as seen in the

[1] http://sourceforge.net/p/kaldi/code/HEAD/tree/sandbox/oplatek2/egs/vystadial/
[2] http://ufal.mff.cuni.cz/grants/vystadial
[3] Czech data: `http://hdl.handle.net/11858/00-097C-0000-0023-4670-6`
[4] English data: `http://hdl.handle.net/11858/00-097C-0000-0023-4671-4`
[5] http://lindat.mff.cuni.cz/repository/
[6] A previous version of our training scripts is published with the data in Lindat repository and described in work [**?**].

first paragraph, are shorter and contain noises. The speech is spontaneous and proper names are frequently used. On the other hand, the other two recording tasks, as seen in the second paragraph, have much broader vocabulary with less named entities, and the ideas are expressed in longer sentences.

```
A DALŠÍ
_NOISE_
JO DĚKUJU MOC TO JSEM CHTĚL VĚDĚT
ZE ZASTÁVKY DEJVICKÁ

PRYČ S TYRANY A ZRÁDCI VŠEMI
UTRHNE SI KVĚT Z KYTICE A ODCHÁZÍ
DYŤ TO JE HROŠÍ NEŽ ZVÍŘE
O LIBERALIZMU TEHDY NEBYLO ŘEČI
CO BY TAM S TEBOU DĚLALI
```

The AMs for Czech are trained on acoustic data from all three very different domains, because there is only 2 hours of in-domain data available in the Alex's public transport domain. The evaluation for Czech data in Section 5.3 is performed on a test set combined from all three domains. The Czech test set is mixed according to the proportion of the domains in training and development set. The English AMs are trained and tested on the data collected from the Venue domain using SDS. The summary of audio sizes in training, development and test set are presented in Table 3.1. Both Czech and English orthographic speech transcriptions were transcribed by humans.

dataset	audio	# sents	# words
English			
train	41:30	47,463	178,110
dev	01:45	2,000	7,376
test	01:46	2,000	7,772
Czech			
train	15:25	22,567	126,333
dev	01:23	2,000	11,478
test	01:22	2,000	11,204

Table 3.1: Size of the data: length of the audio (hours:minutes), number of sentences (which is the same as the number of recordings), number of words in the transcriptions.[**?**]

3.2 Acoustic modelling recipe

In the recipe we search for the best non-speaker adaptive AMs. In this section, the explored methods and their settings are described, and the Section 5.3 presents the results for both Czech and English datasets.

The acoustic modelling techniques focus on modelling the speech to word mapping, so the test utterances are decoded with the least error possible. For correctness the testing uses previously unseen utterances in training or development set, so the real conditions are well simulated. The Figure 3.1 lists all acoustic models

trained in our recipe. The advanced AM is always initiated by audio alignments (respectively acoustic features alignments) obtained using simpler AM.

In paragraphs below the organisation of acoustic model training is described. The used methods are listed in Figure 3.1 together with the their hierarchy. The hierarchy shows that a more advanced method the typically reuses initial values from previously trained simpler AM.

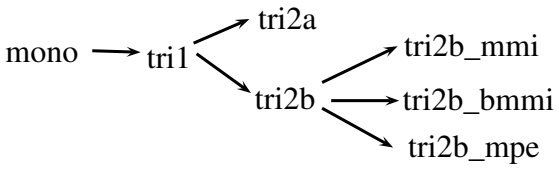
At first, a mono-phone model is trained from flat start using the MFCCs, Δ and $\Delta\Delta$ features. We force-align the feature vectors to HMM states for phones in the corresponding transcriptions. Secondly, we retrain the triphone AM (*tri1a*). One branch of experiments finishes by training MFCC $\Delta + \Delta\Delta$ triphone AM (*tri2a*).

On the other hand, the second branch instead of $\Delta + \Delta\Delta$ transformation uses LDA+MLLT to train AM (*tri2b*). Using the AM *tri2b* three AMs are discriminatively trained using following objective functions:

1. Maximum Mutual Information[**?**][7]. The model *tri2b_ mmi*is train in four loops.

2. Boosted Maximum Mutual Information[**?**]. The model *tri2b_ bmmi* is train in four loops with parameter 0.05.

3. Minimum Phone Error[**?**]. The model *tri2b_ mpe* is also retrained in four loops.



Training method name	Script shortcut
Monophone	mono
Triphone	tri1
$\Delta + \Delta\Delta$	tri2a
LDA+MLLT	tri2b
LDA+MLLT+MMI	tri2b_mmi
LDA+MLLT+bMMI	tri2b_bmmi
MPE	tri2b_mpe

Figure 3.1: Training partial order among AM in our training recipe

The acoustic models *mono*, *tri1*, *tri2a* and *tri2b* are trained generatively. The discriminative models *tri2b_ mmi*, *tri2b_ bmmi* and *tri2b_ mpe* yield better results than generative models if enough data is available. See Figure 3.2. Note that the discriminative may over-fit to train data, so models from the second or third retraining loop may provide better results in general case. However, we have not experienced such behaviour. The discriminative methods use a LM for improving its results over generative models by discriminating according their objective function. In our setup an unigram LM estimated on train set transcriptions was used for discriminative training methods.

The training scripts for Czech and English data differ only in using a different phonetic dictionary and preprocessing the data using the dictionary, but the

[7]Note the Maximum Mutual Information (MMI) function is implemented as bMMI with boosted parameter set to 0.

training itself remains exactly the same. The default bigram and zerogram LMs for testing are built from orthographic transcriptions. The bigram LM is estimated from the training data transcriptions. Consequently, in a test set Out of Vocabulary Word may appear words. The zerogram is extracted from a test set transcriptions. The zerogram is a list of words with probabilities uniformly distributed, so it helps decoding just by limiting the vocabulary size.

In this work, we present results with fixed datasets as described in Table 3.1. The bigram LM contains 17433 unigrams and 79333 bigrams. The zerogram LM is limited to 2944 words.

Furthermore, in Chapter 5, we evaluate the trained Czech AMs on Public Transport Domain on a different test set with a fine tuned LM and the best AM from list in Figure 3.1. The best AM is selected based on the results in Section 3.3.

Setup for feature transformations

We explore not only AM training methods, but we also experiment with two feature transformation techniques. First, the $\Delta + \Delta\Delta$ triples the number of 13 MFCC features by computing also the first and the second derivatives from MFCC coefficients, resulting in 39 features per frame.

Second, the combination of LDA and MLLT is computed from 9 spliced frames consisting of 13 MFCC features. The default context window of 9 frames takes four frames from the left context and four frames from the right context. The LDA and MLLT feature transformation gains substantial improvement over $\Delta + \Delta\Delta$ transformation. See Figure 3.2.

Decoding setup

We described the training setup in sections above, and now we describe the setup for testing and evaluating the trained AMs. For each trained AM we used the same speech parametrisation and feature transformation as was used for the given AM at training time. We experiment with all trained AMs with both zerogram and bigram LM.

The AM, LMs, phonetic dictionary and few helper files are used to build *HCLG* decoding graph for each AM and feature transformation combination. The decoding graph is used for the speech recognition on development and test set. The parameters are set to default values; the exceptions are *beam=12.0, lattice-beam=6.0, max-active-states=14000* and Language Model Weight. The Language Model Weight (LMW) is estimated on the development set and the best value is used for testing. The details on the parameters can be found in Section 4.3.

The *gmm-latgen-faster* decoder is used for the evaluation on testing data. It generates a word level lattice for each utterance and the one best hypothesis is extracted from the decoded lattice and evaluated by WER and SER metrics against the reference transcription.

Note, we are able to exactly reproduce the results of *gmm-latgen-faster* decoder with our *OnlineLatgenRecogniser*, but the *gmm-latgen-faster* was used for evaluation in the scripts, so the Kaldi users do not have to install our extension.

3.3 Evaluation

The experiments focus on comparing the quality of ASR hypothesis measured by WER on AMs trained by different methods. We are not interested in absolute

numbers since we model the language using a weak LM focusing on the acoustic modeling. By training only simple bigram LM we let the AM influence the recognition quality more significantly. The same motivation lead us to use zerogram LM which just limits vocabulary in the decoding task, and does not advise the decoding search more probable phrases as higher order LM does. Consequently, the best words are chosen among all hypotheses only by acoustic similarity.

We concentrate on acoustic modelling since we believe that; if two AMs am_1, am_2 are trained with the same weak lm_{weak} and the first AM gains lower WER than the second one ($wer_1^{weak} < wer_2^{weak}$), then in the same experiment just with a richer LM the first AM will still gain lower WER ($wer_1^{rich} < wer_2^{rich}$).

First, we show how the data size influence the quality of AMs measured by WER. Second, the best results on full data is presented. Finally in Subsection 3.3.2, the best Kaldi results are compared against the results obtained by well-written HTK scripts by Keith Vertanen and improved by Matěj Korvas [**?**] on the same Vystadial dataset.

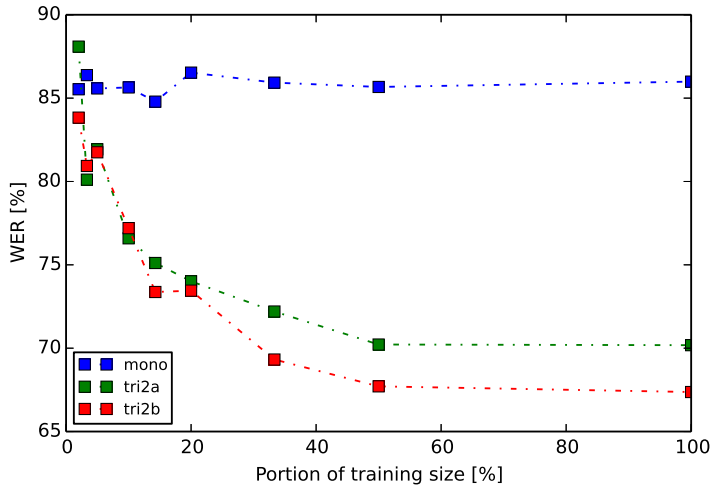


Figure 3.2: Performance of Czech generative AMs based on training size for acoustic models. The zerogram LM results in a high WER, but allows evaluating only acoustic modelling.

The Figure **??** describes how the amount of acoustic data influences the WER. We illustrate that even with small datasets like Vystadial the high quality AM can be trained. The WER decreases significantly if new data are added to small dataset, but WER reduction becomes very small between 50% of data and the full data. One can also see that the $\Delta+\Delta\Delta$ feature transformation is clearly outperformed by LDA+MLLT setup on full data. Note also that the monophone AM is typically used for the initialisation of triphone models and requires small portion of data to reach its limit. The WER is rather high due to the use of zerogram LM. We evaluate only generative LMs since we would have to have a fixed LM for discriminative methods and we do not have any obvious choice how to build one.

It may seem that more acoustic data is not needed for this domain, but discriminative training methods require more training data, and with more transcribed data a better LM adaptation can be achieved. The Figure 3.3 shows the effect of in-domain data size for LM on quality of speech decoding. The AM *tri2b_bmmi* and decoding parameters were fixed and the experiments were performed with different

LMs which differ only in the training size used for their estimation. Note that this experiment was run by Ondřej Dušek and not as a part of the training scripts[8]. The experiment was run on different test set from Public Transport Domain and the LMs were built also from that in-domain data.

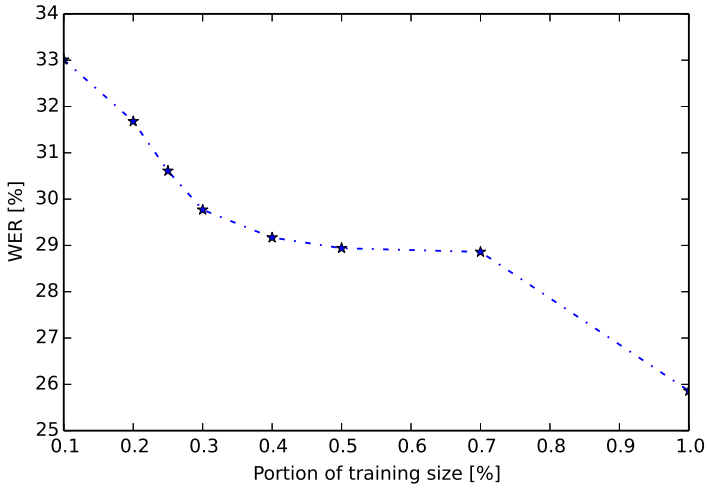


Figure 3.3: Influence of in-domain text size of LM on speech recognition quality. The AM *tri2b_ bmmi* and parameters are fixed and only LM training size varies.

To conclude the first section, we continue to collect new acoustic data through the dialogue system Alex because

- the domain still changes due to new features and we need to update the LM,
- we can still improve the best discriminatively trained AM,
- and the speech recogniser is presumably more robust to new speakers.

3.3.1 Results

In this section we present the results of different acoustic training methods and we choose the best non-speaker adaptive setup. The Table 3.2 presents AMs results.

The complexity of the Czech data is clearly much larger than the complexity of English data. The low WER on the Czech dataset may be explained be following reasons:

- The mix of a very different domain and recording conditions is difficult to model by both AM and LM.
- The *Call Friend* and *Repeat After Me* collections task have a really broad domain which affect language modelling.
- The flective languages such as Czech have larger vocabulary and higher Out of Vocabulary Words (OOVs) since one word may have several variations.

[8]Ondřej Dušek used the scripts developed by us for Alex dialogue system for Public Transport Domain

language/method	zerogram	bigram	RTF
Czech			
tri $\Delta + \Delta\Delta$	70.7	56.6	-1.0
tri LDA+MLLT	68.2	53.9	-1.0
tri LDA+MLLT+MMI	65.3	49.5	-1.0
tri LDA+MLLT+bMMI	65.3	49.3	-1.0
tri LDA+MLLT+MPE	63.8	49.2	-1.0
English			
tri $\Delta + \Delta\Delta$	35.7	16.2	-1.0
tri LDA+MLLT	33.28	15.8	-1.0
tri LDA+MLLT+MMI	25.01	10.4	-1.0
tri LDA+MLLT+bMMI	23.9	10.2	-1.0
tri LDA+MLLT+MPE	22.41	11.1	-1.0

Table 3.2: Word error rates for zerogram and bigram LM for different training triphone methods. The RTF was measured for bigram LM. The 'tri $\Delta+\Delta\Delta$' row shows results for a generative model which is comparable to the model trained using the HTK scripts.

Nevertheless, the training scripts for the Czech data are very important since there is no other Czech acoustic data available[9].

The WER on the Vystadial English data is lower than 20% for discriminative methods, which is reasonable, given the broad but limited domain.

The discriminative training methods clearly outperformed the generative AMs, and also the LDA+MLLT is more effective feature transformation than using $\Delta + \Delta\Delta$ features. On the other hand, there are subtle differences among the three discriminatively trained AM in terms of performance, *TODO: but Boosted Maximum Mutual Information (bMMI) has lower RTF than Minimum Phone Error (MPE) method* . As a result, we choose AM (*tri2b_bmmi*) discriminatively trained by bMMI with MFCC, LDA+MLLT preprocessing.

3.3.2 Kaldi and previous HTK results comparison

We want to roughly compare Kaldi and HTK on the Vystadial Czech and English datasets, since we are not familiar with the Kaldi toolkit. In addition, by using state of the art HTK scripts which were used on ther domains we can see the complexity of collected Vystadial acoustic data.

We present results for triphone AM estimated using Baum-Welsch iterative training on zero-gram and bi-gram LMs. The *HVite* HTK decoder was used to perform *TODO: the decoding with the same LMs as used in Kaldi scripts.* The training procedure is further described in work [**?**].

The results suggest that Kaldi achieves similar WER compared to HTK when using standard generative training methods and bigram LMs. Using more advanced discriminative training methods, one can obtain a substantial decrease in WER.

The experiment using MFCC, LDA & MLLT and bMMI discriminative training is a state of the art set up for speaker independent speech recognition[**?**] and outperforms HTK models.

[9] According our knowledge.

language/method	zerogram	bigram
Czech		
tri $\Delta + \Delta\Delta$	64.5	60.4
English		
tri $\Delta + \Delta\Delta$	50.0	17.5

Table 3.3: Word error rates on test set obtained using HTK and either a zerogram or a bigram LM. [**?**]

4. Real time recogniser

We implemented a lightweight modification of the *LatticeFasterDecoder* from the Kaldi toolkit, improved on-line speech parametrisation and feature processing in order to create an *OnlineLatgenRecogniser*. The Kaldi *OnlineLatgenRecogniser* implements on-line interface which allows incremental speech processing which is able to process the incoming speech in small chunks incrementally. As a result, the speech decoding is performed as user speaks and the ASR output is obtained with minimal latency.

The implementation of the recogniser was motivated by the lack of an on-line recognition support in Kaldi toolkit. Therefore, the toolkit decoders could not be used in applications such as spoken dialogue systems. Although Kaldi included an on-line recognition application; but hard-wired timeout exceptions, audio source fixed to a sound card, and a specialised 1-best decoder limit its use to demonstration of Kaldi recognition capabilities only.

Our on-line recogniser may use acoustic models trained using the state-of-the-art techniques, such as Linear Discriminant Analysis (LDA), Maximum Likelihood Linear Transform (MLLT), Boosted Maximum Mutual Information (BMMI), Minimum Phone Error (MPE). It produces word posterior lattices which can be easily converted into high quality n-best lists. The recogniser's speed and latency can be effectively controlled off-line by optimising a language model. At runtime the speed of decoding is controlled by a beam threshold. The latency which corresponds to the amount of time spent on word posterior lattice extraction depends on level of approximations used.

Section 4.1 describes the implementation of the *OnlineLatgenRecogniser*. *OnlLatticeFasterDecoder* is the core component of the speech recogniser and it greatly influences the interface of *OnlineLatgenRecogniser*. First, we describe the *OnlineLatgenRecogniser* in Subsection 4.1.1. Later, we present the new interface of audio buffering, speech parametrisation and feature transformations in Subsection 4.1.3, and Subsection **??** discuss word posterior lattice extraction. Next Section 4.2 describes, *PyOnlineLatgenRecogniser*, an extension of *OnlineLatgenRecogniser* into Python. Finally, Section **??** summarize how we extended the Kaldi library.

4.1 OnlineLatgenRecogniser

The standard Kaldi interface between the components of the toolkit is based on a batch processing paradigm, where the components assume that whole audio signal is available when recognition starts. However, when performing on-line recognition, one would like to take advantage of the fact that the signal appears in small chunks and can be processed incrementally. When properly implemented, this significantly reduces recogniser output latency.

4.1.1 *OnlLatticeFasterDecoder*

We did not implement no new functionality *OnlLatticeFasterDecoder*, but we only reorganised the code of base class *LatticeFasterDecoder*. We splited the LatticeFasterDecoder::Decode which performed two separate tasks into two other functions. The *LatticeFasterDecoder::Decode* function runs a beam search from frame 0 to

the end of each utterance. In addition, a pruning is triggered periodically in the function. Namely, we control the beam search by following functions:

- *Decode* – decoding a fixed number of audio frames instead of decoding whole utterance, pruning is triggered periodically,
- *PruneFinal* – run final pruning and so prepare the internal data structures for lattice extraction,
- *Reset* – preparing the recogniser for a new utterance.

In the *PruneFinal* function, called at the end of an utterance, the states selected by beam search are pruned with the knowledge that no further search will be performed so more states can be safely discarded.

Extracting the ASR is performed by method of *LatticeFasterDecoder*, namely:

- *GetRawLattice* returns state-level lattice,
- *GetLattice* extracts from state-level lattice word lattice which is returned,
- *GetBestPath* finally return just one-best bath.

The state-level lattice returned from *GetRawLattice* method can be understand as lattice on triphone level. Single word hypothesis is typically represented as multiple state-level hypotheses since the words are aligned differently, which means that the same words were pronounced at different time according the state-level hypotheses.

furthprocessed states pruning and later marking the last frame of the utterance as final. Namely, we create orward decoding, marking current frame as final and extracting lattice from frame 0 to final frame into three function

4.1.2 *OnlineLatgenRecogniser* interface

To achieve this, we implemented Kaldi's *DecodableInterface* supporting incremental speech pre-processing, which includes speech parameterisation, feature transformations, and likelihood estimation. In addition, we subclassed *LatticeFasterDecoder* and split the original batch processing interface.

The *OnlineLatgenRecogniser* makes use of the new incremental speech pre-processing and modified *LatticeFasterDecoder*. It implements the following interface:

- *AudioIn* – queueing new audio for pre-processing,
- *Decode* – decoding a fixed number of audio frames,
- *PruneFinal* – preparing internal data structures for lattice extraction,
- *GetLattice* – extracting a word posterior lattice and returning log likelihood of processed audio,
- *Reset* – preparing the recogniser for a new utterance,

The `C++` example in Listing 4.1 shows a typical use of the *OnlineLatgenRecogniser* interface. When audio data becomes available, it is queued into the recogniser's buffer (line 11) and immediately decoded (lines 12-14). If the audio data is supplied in small enough chunks, the decoding of queued data is finished before new data arrives. When the recognition is finished, the recogniser prepares for lattice extraction (line 16). Line 20 shows how to obtain word posterior lattice as an

Listing 4.1: Example of the decoder usage

```
OnlineLatgenRecogniser rec;
rec.Setup(...);

size_t decoded_now = 0;
size_t max_decoded = 10;
char *audio_array = NULL;

while (recognitionOn())
{
  size_t audio_len = getAudio(audio_array);
  rec.AudioIn(audio_array, audio_len);
  do {
    decoded_now = rec.Decode(max_decoded);
  } while(decoded_now > 0);
}
rec.PruneFinal();

double tot_lik;
fst::VectorFst<fst::LogArc> word_post_lat;
rec.GetLattice(&word_post_lat, &tot_lik);

rec.Reset();
```

OpenFST object. The *getAudio()* function represents a separate process supplying speech data. Please note that the recogniser's latency is mainly determined by the time spent in the *GetLattice* function.

The source code of the *OnlineLatgenRecogniser* is available in Kaldi repository[1].

4.1.3 On-line feature pre-processing

The pre-processing consist of audio signal buffering, MFCC feature extraction and applying feature transformation. The pre-processing and the decoder are bridged by the *decodable* interface as illustrated in Figure 4.1. We briefly describe each step:

- Audio buffering
- Computation of MFCC features on overlapping audio window.
- Applying feature transformation on top of MFCC features.
  - $\Delta + \Delta\Delta$ requires at least two previous frames.
  - The $LDA + MLLT$ is computed using context, which by default is set to four previous and four future frames.

  Note, that the $LDA + MLLT$ and the $\Delta + \Delta\Delta$ transformations are complementary.
- The *Decodable* interface in our case the its *OnlDecodableDiagGmmScaled* implementation queries the AM for the probability for acoustic features a and given state. Note that the acoustic features a are the output of the previous steps.
- The decoder itself performs the search in state level space having the probabilities from the *Decodable* interface. States represents triphones where for some thriphones the parameters are shared.

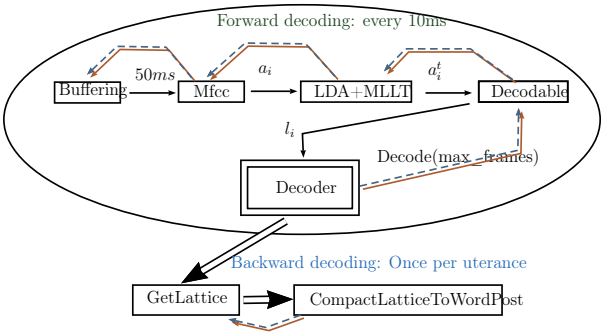


Figure 4.1: Components for on-line decoding

Each step in Figure 4.1 is implemented as a class. During decoding each class is instantiated only once at the beginning. The pre-processing classes are aggregated together.

The *OnlLatticeFasterDecoder* performs forward decoding frame by frame using the Viterbi beam search. The forward decoding is performed on request by calling the method *decode(int max_frames)*. It returns the number of frames which were actually decoded, which is always smaller than *max_frames*. In order to evaluate the probability of acoustic features for the new frame the decoder query the *Decodable* interface.

In case of *OnlDecodableDiagGmmScaled*, the on-line implementation, can trigger chain reaction. If *OnlFeatureMatrix* does not contain the acoustic features for the new frame, it asks the previous component to compute the features, namely *OnlDeltaInput* or *OnlLdaInput*. If the previous component can not provide the features, it return default empty value indicating, that no features are not available at the moment. It triggers the message that no data is available and the decoder's method *decode(..)*, returns zero, meaning that no frames were decoded.

We did not implement the classes *OnlDeltaInput, OnlLdaInput, OnlFeInput* and *OnlFeatureMatrix* from scratch. We started the work with Mathias Pawlik implementation and we implemented few buffering details in the classes, but mainly we removed the built in timeouts and changed the interface. In addition, we suggest that a higher program logic e.g. timeouts should not be embedded into speech recogniser. It slows the decoding and it limits the usage of such decoder. Note that we also added the *OnlBuffSource* class, which just allows buffer the raw Pulse-Code Modulation (PCM) audio.

4.1.4 Post-processing the state lattice

The *CompactLattice* determinised at state level still may contain multiple paths for each word sequence encoded in the lattice. In fact, typically there is number of state level alternatives for each word sequence. We need to realize that the phones are not represented individually at the state level and we does not concatenate the phone labels on path in order to obtain words and sentences. The words label is located on the first arc of the word HMM. The rest of the arcs in the graph are ϵ transition.

alignment

Note that HMM states are time synchronous, so traversing one arc represent a fixed time slot. The time slot corresponds to frame shift as introduced in Subsec-

[1] `https://sourceforge.net/p/kaldi/code/HEAD/tree/sandbox/oplatek2/src/dec-wrap/`

tion 2.1.1. The mapping between each word and the number of arcs from beginning of the utterance is called alignment.

The *CompactLattice* distinguishes each path not only according word labels on the path, but also according the alignments. In order to obtain only the word lattice we discard the alignments. In next steps we convert the *CompactLattice* to standard OpenFST object.[2]

The following steps convert the weights on the lattice from representing likelihood to posterior probabilities. The posterior probabilities are of course approximation of true posterior probabilities for individual paths in lattice.

The first approximation has a very small impact. The first approximation is that we are using only the lattice constructed from the active states in beam search and not the complete search space. The discarded hypotheses have very low posterior probabilities, so it does a little harm.

More serious problem is violating the assumption that the AM and LM models are computing the true likelihood. In generative models we are training and improving the models so the likelihood match the reality as much as possible. On contrary, the discriminative AM models deliberately favor the most probable hypothesis, so they are not computing the true likelihood. The discriminative behaviour can be compensated, but we do not solve the problem in the decoder, because the decoder has no information, which AM was used.

Listing 4.2: Converting *CompactLattice* to posterior word lattice

```
double CompactLatticeToWordsPost(CompactLattice &clat,
                                 fst::VectorFst<fst::LogArc> *pst) {
  {
    Lattice lat;
    fst::VectorFst<fst::StdArc> t_std;
    RemoveAlignmentsFromCompactLattice(&clat); // remove the alignments
    ConvertLattice(clat, &lat); // convert to non-compact form.. no new
        ↪states
    ConvertLattice(lat, &t_std); // this adds up the (lm,acoustic) costs
    fst::Cast(t_std, pst);  // reinterpret the inner implementations
  }
  fst::Project(pst, fst::PROJECT_OUTPUT);
  fst::Minimize(pst);
  fst::ArcMap(pst, fst::SuperFinalMapper<fst::LogArc>());
  fst::TopSort(pst);
  std::vector<double> alpha, beta;
  double tot_lik = ComputeLatticeAlphasAndBetas(*pst, &alpha, &beta);
  MovePostToArcs(pst, alpha, beta);
  return tot_lik;
}
```

The computing of the posterior probabilities is done through standard forward-backward algorithm. We reused Kaldi function for computing helper α and β data structures, but we implemented the *MovePostToArcs* function for updating the lattice weights from likelihood to posterior probabilities based on α and β.

4.2 PyOnlineLatgenRecogniser

In addition, we developed a Python extension, *PyOnlineLatgenRecogniser*, exporting the *OnlineLatgenRecogniser* C++ interface. This can be used as an example of bringing Kaldi's on-line speech recognition functionality to higher-level programming languages. This Python extension is used in the Alex Dialogue Systems Framework [**?**].

[2] Kaldi implements a special semiring for *CompactLattice*[**?**].

PyOnlineLatgenRecogniser is a thin wrapper around *OnlineLatgenRecogniser* implemented using Cython[**?**]. The Cython is well known for its speed when interfacing Python and C++. In addition, we extended PyFST library[**?**] which interfaces via Cython the core parts of OpenFST library into Python. The output of the *OnlineLatgenRecogniser* is represented as OpenFST lattices and is input is raw binary data. Consequently, the recogniser as well as its input and output can be seamlessly used from Python.

We also implemented conversion of the word posterior lattices which are returned by *PyOnlineLatgenRecogniser* to an n-best list. The implementation is efficient since the OpenFST shortest path algorithm is used on small lattices.

The minimalistic Python example in Listing 4.3 shows usage of the *PyOnlineLatgenRecogniser* and the decoding of a single utterance.

Listing 4.3: Fully functional example of the *PyOnlineLatgenRecogniser* interface

```
d = PyGmmLatgenWrapper()
d.setup(argv)
while audio_to_process():
    d.frame_in(get_raw_pcm_audio())
    dec_t = d.decode(max_frames=10)
    while dec_t > 0:
        decoded_frames += dec_t
        dec_t = d.decode(max_frames=10)
d.prune_final()
lik, lat = d.get_lattice()
```

4.3 Decoder parameters

In this section we focus on parameters of the *OnlineLatgenRecogniser*, which affect speed of decoding. The parameters of the speech recognizer are passed to speech parametrisation, feature transformations or *OnlLatticeFasterDecoder*. Most of the parameters for speech parametrisation and feature transformations does not effect speed of decoding. The frame width (set to 25 ms), the frame shift (set to 10 ms) and the frame splicing (nine frames are spliced) are the only "preprocessing" parameters which could significantly affect the forward decoding speed. We did not experiment with setting those and we used the recommended values.

On the other hand, the *beam*, *lattice-beam* and *max-active-states* parameters directly affect the speed of speech recognition. The *beam* and *max-active-states* parameters affect speed of beam search, whereas *lattice-beam* influence speed of lattice extraction.

4.4 Summary

The *OnlLatticeFasterDecoder* is able to perform on-line speech recognition. Its parameters for real-time decoding, can be setup based on its reference batch decoder *LatticeFasterDecoder* used through *gmm-latgen-faster* executable.

The minimal interface works very well for supported MFCC speech parametrisation, $\Delta - \Delta\Delta$ feature transformation or LDA+MLLT and both generative training and discriminative training using bMMI and MPE. The setup yields the best results for non-speaker adaptive methods.

5. Kaldi ASR in Alex SDS

This chapter discuss the details of deploying *OnlineLatgenRecogniser* into Alex dialogue system written in Python. The *OnlineLatgenRecogniser* is used in Alex dialogue system for Czech Public Transport Domain available on public toll-free (+420) 800 899 998 line.

First, the architecture of Alex Spoken Dialog System (SDS) is described. Second, Section 5.2 presents how the Python wrapper *PyOnlineLatgenRecogniser* was integrated into SDS Alex. Finally, Section 5.3 evaluates the decoder in Alex dialogue system on Czech Public Transport Information (PTI) domain.

5.1 Alex dialogue system architecture

The Alex dialogue system is developed in Python programming language and consist of six major components.

1. Voice Activity Detection (VAD)
2. Automatic Speech Recognition (ASR)
3. Spoken Language Understanding (SLU)
4. Dialog Manager (DM)
5. Natural Language Generation (NLG)
6. Text To Speech (TTS)

The Alex dialogue system has a speech to speech user interface. The system interacts with the user in *turns*. During a single turn the dialogue system waits for a user spoken input, processes the speech and generates the spoken response. The data flow in a single turn is depicted in Figure 5.1.

Each of the Alex's component runs in separate process in order parallelize the input data processing and output data generation. The components communicates among themselves through system pipes.

The Alex's framework is organised into several logical parts:

- The core library is located at *alex/components/*. The library is domain and language independent. All components in Figure 5.1 are implemented in this core library.
- Settings and scripts for specific domain applications are located in *alex/applications/*. For example, there is a bash scripts *alex/applications/PublicTransportInfoCS/vhub_live_kaldi* which starts the Alex SDS service which provides Public Transport Information at the 800 899 998 toll free line.
- The general scripts which use external tools or data can be found in:
  - *alex/corpustools/* directory which focuses on formatting and organising the collected data,
  - and alex/tools/ directory which stores code for modelling Voice Activity Detection (VAD), ASR. It also stores code for evaluation and *SIP* client.

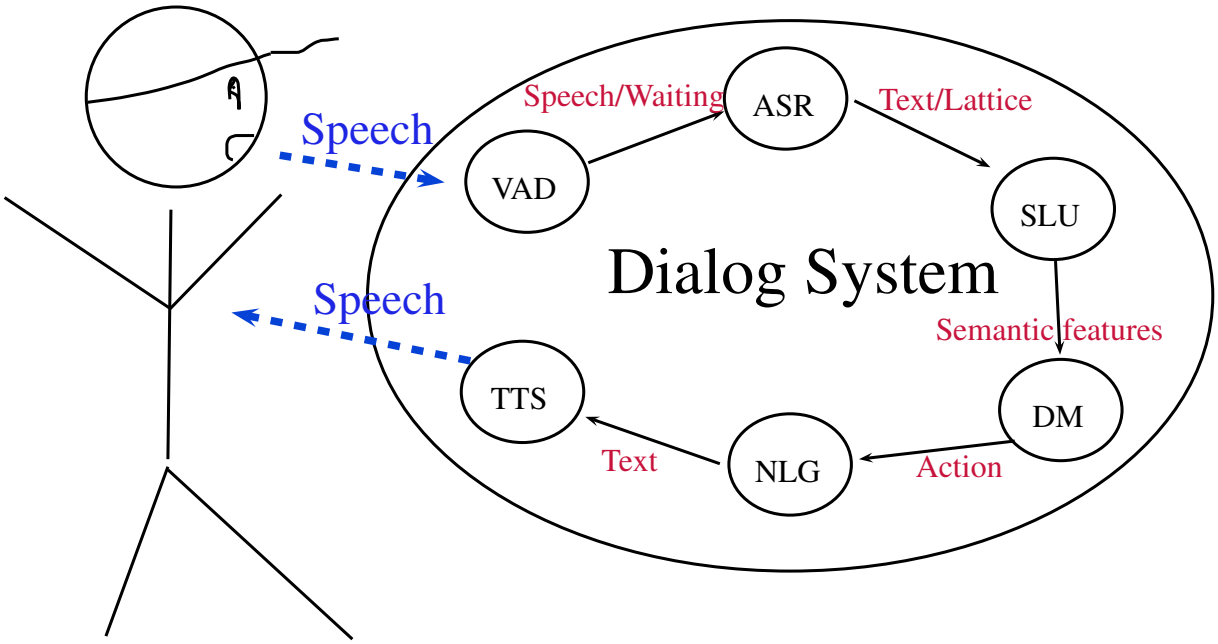


Figure 5.1: Single turn in Alex dialogue system

- Integration tests are stored in *alex/tests/*.
- The *alex/utils/* directory contains simple utilities for various purposes.

The components depicted in Figure 5.1 are represented as modules under *alex/components/* e.g. *alex/components/asr/*, *alex/components/tts/*, etc.. The source code of the components is very modular. For example the ASR component currently supports several ASR engines wrapped in modules, which implement the base class *ASRInterface* methods. See Listing 5.1. The supported engines are:

- OpenJulius *alex/components/asr/julius.py* interfaces OpenJulius decoder through sockets for on-line recognition.
- Google *alex/components/asr/google.py* uses cloud service for batch decoding.
- Kaldi *alex/components/asr/kaldi.py* imports *PyOnlineLatgenRecogniser* class and uses its methods and the utilities for on-line decoding.

Implementations of components can be easily change using simple configuration file. For example, by specifying the ASR tool, AM and other parameters one can choose between ASR cloud based service, local OpenJulius or Kaldi speech recognition.

In order to prepare a specific application one need to train appropriate models for classifiers or create setup for cloud services, which implements Alex components functionality. As an example, a public phone call service[1], which provides PTI and weather information in Czech language, is developed in directory *alex/applications/PublicTransportInfoCS/*. The Alex's framework provides source code for domain dependent training for models in VAD, ASR, SLU and also Dialogue Manager (DM).

[1]On telephone number (+420) 800 899 998

5.2 Kaldi integration into Alex's Spoken Dialog System framework

Integration of the Kaldi real-time recognizer into Alex's framework requires implementing new features:

1. The *kaldi.py* module is implemented so the Alex's ASR component can use *PyOnlineLatgenRecogniser* decoder through *ASRInterface*. See Subsection 5.2.1.

2. The training scripts for AMs were prepared as described in Chapter 3.

3. The scripts for building custom decoding graph *HCLG* and evaluation of Alex's ASR unit were developed. See Subsection 5.2.2 and Section 5.3.

The decoding graph *HCLG* is necessary in order to run any Kaldi decoder, and the evaluation scripts are used to find out decoding parameters which balance quality and speed of speech recognition.

5.2.1 *PyOnlineLatgenRecogniser* in Alex

The ASR component in the Alex dialogue system runs as separate process, and the speech recognition is triggered based on VAD decisions.

If VAD detects starts of speech in input audio stream, it passes the audio data to ASR component and *rec_in* method is called. See Listing 5.1. The audio data are passed to the ASR component as instance of class *Frame* in *rec_in* method. The audio is forward decoded using beam search as user speaks. The method *rec_in* adds gradually the new audio in *PyOnlineLatgenRecogniser* buffer and calls forward decoding on new buffered audio. The buffering and forward decoding could have been split into two steps, but the current setup works well.[2]

If VAD recognises end of speech, no more data are sent to ASR engine and *hyp_out* method is called in order to extracted word lattice. Extracting the word lattice is also referred as backward decoding since the data structures are traversed in time-synchronous backward. In case of *PyOnlineLatgenRecogniser* word posterior probabilities are computed using forward-backward algorithm. Then, the word posterior lattice in *PyFst* format is converted to an n-best list represented as proprietary *UtteranceNBlist* n-best list.[3] The conversion from a lattice to an n-best list is very fast because the *OpenFST* shortest path algorithm is used on lattices which usually contain only tens of words.

The *flush* method is used only if the speech recogniser wants to throw away the buffered audio input and reset the decoding.

The method *rec_wav_file* nicely illustrates how the two methods *rec_in* and *hyp_out* are used for decoding. Since the method is used only for testing purposes, it sends all input audio to the speech recogniser at once. However, in real-time application the audio is passed to *PyOnlineLatgenRecogniser* in small chunks, so the forward decoding can run as a user speaks.

In the Kaldi real-time settings, latency of ASR unit depends mostly on the time spent in *hyp_out* method. In the *hyp_out* method a word posterior lattice

[2] If the ASR component is busy with forward decoding and is not able to accept the audio from VAD component the audio just waits in VAD buffer instead of in *PyOnlineLatgenRecogniser*'s buffer.

[3] In the future, we would like to implement keyword spotting using directly *pyfst* lattices in our SLU unit.

Listing 5.1: ASRInterface

```
class ASRInterface(object):

    ...

    def rec_in(self, frame):
        raise ASRException("Abstract class: Not implemented")

    def flush(self):
        raise ASRException("Abstract class: Not implemented")

    def hyp_out(self):
        raise ASRException("Abstract class: Not implemented")

    ...


    def rec_wav_file(self, wav_path):
        pcm = load_wav(self.cfg, wav_path)
        frame = Frame(pcm)
        res = self.rec_wave(frame)
        self.flush()
        return res
```

is extracted using the *PyOnlineLatgenRecogniser::GetLattice* method as described in Subsection 4.2. For most cases the latency is well below 200 ms for our settings as illustrated in Figure 5.3.

TODO: SIMPLE HIGH LEVEL POINT OF VIEW: The integration of PyOnlineLatgenRecogniser is simple yet effective. However, be aware of triggering backward decoding too often. It may seem that a user should wait to response of the Alex dialogue system before he speaks again and consequently new audio is buffered to recognition. However in practice, users speak spontaneously; quite often an user finishes utterance and immediately starts speaking in order to change his request. The spontaneous speech may cause delays of ASR unit since the forward and backward decoding run in single process. In case of very short consecutive utterances, the processor time which is meant for backward decoding should also be used to forward decoding.

We notices the problem for chains of noises detected in VAD components as multiple short utterances. The backward decoding (*rec_in* method) was extracting hypothesis while the audio input was still buffering the noisy speech and want to call *rec_in*. The backward decoding (*hyp_out* method) was called so often that almost no forward decoding was performed. The problem was solved by improving VAD so the *hyp_out* method is not triggered so often. It is useful to target settings for a forward decoding RTF smaller than 1.0 e.g. 0.6, because with reduced RTF the forward decoding can catch up the delay caused by backward decoding the previous utterance. See Section 5.3 for RTF definition.

5.2.2 Building in-domain decoding graph

We developed scripts for building a decoding graph and other necessary files for decoding with *PyOnlineLatgenRecogniser*.[4] In order to build the decoding graph the best AM obtained by training scripts and an in-domain LM are used. The scripts are based on standard Kaldi utilities. If new LM or AM needs to be tested, new *HCLG* graph has to be built. The decoding graph also needs to specified in

[4]The evaluation scripts and *HCLG* scripts are located in *alex/applications/PublicTransportInfoCS/hclg*.

configuration file before launching Alex dialogue system with Kaldi ASR engine.

Acoustic and language models used

The *OnlineLatgenRecogniser* is evaluated on a corpus of audio data from the Public Transport Information (PTI) domain. In PTI, users can interact in Czech language with a telephone-based dialogue system to find public transport connections [**?**]. The PTI corpus consist of approximately 12k user utterances with a length varying between 0.4 s and 18 s with median around 3 s. The data were divided into training, development, and test data where the corresponding data sizes were 9496, 1188, 1188 respectively. For evaluation, a domain specific class-based language model with a vocabulary size of approximately 52k and 559k n-grams was estimated from the training data. Named entities e.g., cities or bus stops, in class-based language model are expanded before building a decoding graph. The perplexity of the resulting language model evaluated on the development data is about 48.

Since the PTI acoustic data amounts to less then 5 hours, the acoustic training data was extended by additional 15 hours of telephone out-of-domain data from VYSTADIAL 2013 - Czech corpus [**?**]. The acoustic models were obtained by BMMI discriminative training with LDA and MLLT feature transformations. A detailed description of the training procedure is given in Chapter 3.

5.3 Evaluation of *PyOnlineLatgenRecogniser* in Alex

We focus on evaluating the speed of the *OnlineLatgenRecogniser* and its relationship with the accuracy of the decoder. We evaluate following measures:

- Real Time Factor (RTF) of decoding – the ratio of the recognition time to the duration of the audio input,
- Latency – the delay between utterance end and the availability of the recognition results,
- Word Error Rate (WER).

Accuracy and speed of the *OnlineLatgenRecogniser* are controlled by the *max-active-states*, *beam*, and *lattice-beam* parameters [**?**]. *Max-active-states* limits the maximum number of active tokens during decoding. *Beam* is used during graph search to prune ASR hypotheses at the state level. *Lattice-beam* is used when producing word level lattices after the decoding is finished. It is crucial to tune these parameters optimally to obtain good results.

In general, one aims for a RTF smaller than 1.0. Moreover, in practice, it is useful if the RTF is even smaller because other processes running on the machine can influence the amount of available computational resources. Therefore, we target the RTF of 0.6 in our setup.

We used grid search on the test set to identify optimal parameters. Figure 5.4 (a) shows the impact of the *beam* on the WER and RTF measures. In this case, we set *max-active-states* to 2000 in order to limit the worst case RTF to 0.6. Observing Figure 5.4 (a), we set *beam* to 13 as this setting balances the WER.Figure 5.4 (b) shows the impact of the *lattice-beam* on WER and latency when *beam* is fixed to 13. We set *lattice-beam* to 5 based on Figure 5.4 (b) to obtain the 95th latency percentile of 200 ms, which is considered natural in a dialogue [**?**]. *Lattice-beam*

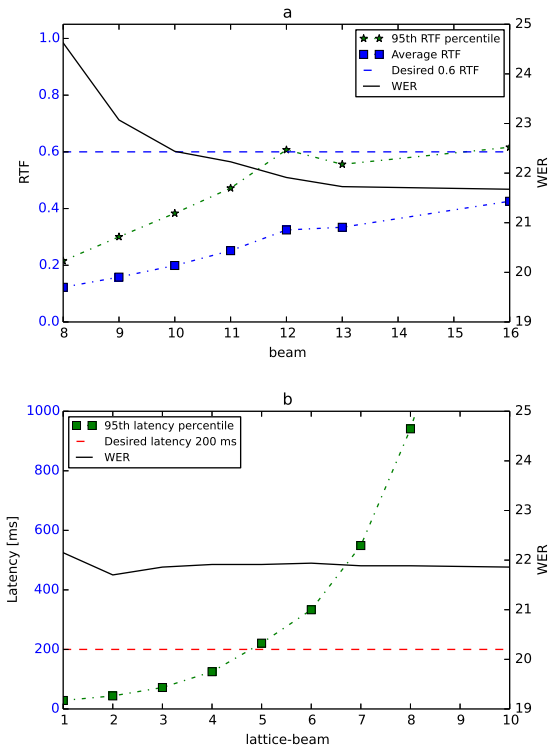


Figure 5.2: The upper graph (a) shows that WER decreases with increasing *beam* and the average RTF linearly grows with the beam. Setting the maximum number of active states to 2000 stops the growth of the 95th RTF percentile at 0.6, indicating that even in the worst case, we can guarantee an RTF around 0.6. The lower graph (b) shows how latency grows in response to increasing *lattice-beam*.

does not affect WER, but larger *lattice-beam* improves the oracle WER of generated lattices [**?**]. Richer lattices may improve SLU performance.

Figure 5.3 shows the percentile graph of the RTF and latency measures over the test set. For example, the 95th percentile is the value of a measure such that 95% of the data has the measure below that value. One can see from Figure 5.3 that 95% of test utterances is decoded with RTF under 0.6 and latency under 200 ms. The extreme values are typically caused by decoding long noisy utterances where uncertainty in decoding slows down the recogniser. Using this setting, the *OnlineLatgenRecogniser* decodes the test utterances with a WER of about 21%.

In addition, we have also evaluated Google ASR service as we used it previously in Alex SDS. The Google ASR service decoded the test utterances from the PTI domain with 95% latency percentile of 1900ms and it reached WER about 48%. The high latency is presumably caused by the batch processing of audio data and network latency, and the high WER is likely caused by a mismatch between Google's acoustic and language models and the test data.

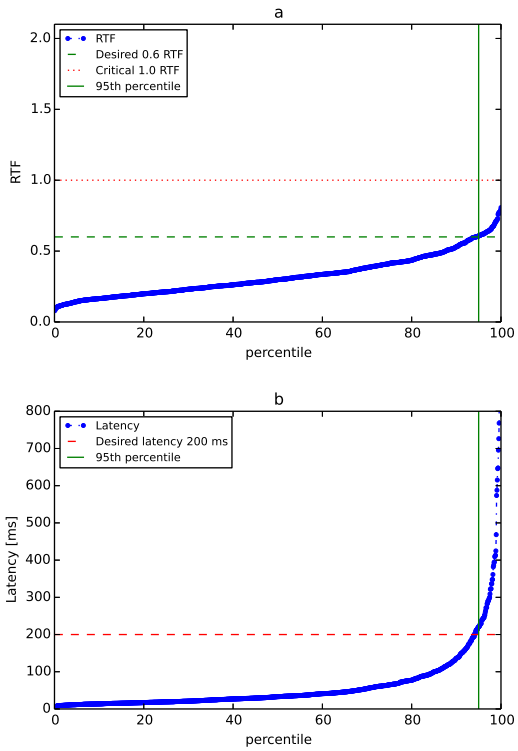


Figure 5.3: The percentile graphs show RTF and Latency scores for test data for *max-active-sates*=2000, *beam*=13, *lattice-beam*=5. Note that 95 % of utterances were decoded with the latency lower that 200ms.

Results

Based on evaluation we selected the best setup[5] for ASR component in Alex Dialogue System Framework with WER under 22 %, latency less tha 200 ms and RTF under 0.6 on PTI domain. To conclude, the *OnlineLatgenRecogniser* performs significantly better than our previous ASR engines: OpenJulius decoder or queries to cloud Google ASR service.

[5]Setup: *beam* 12, *lattice-beam* 5, *max-active-states* 2000.

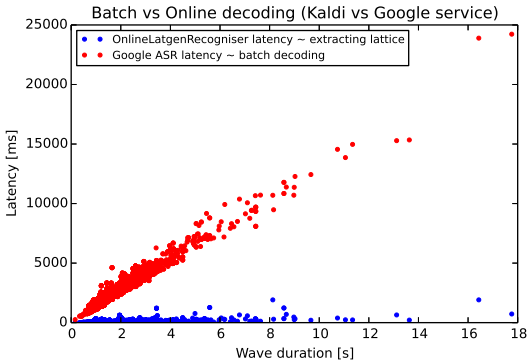


Figure 5.4: Shorter latency of custom on-line decoder (OnlineLatgenRecogniser) over batch decoding with cloud service (Google ASR service).

6. Conclusion

The Kaldi toolkit is a speech recognition toolkit distributed under a free license [**?**]. The toolkit is based on Finite State Transducers, implements state-of-the-art acoustic modelling techniques, is computationally efficient, and is already widely adapted among research groups. Its only major drawback was the lack of on-line recognition support. Therefore, it could not be used directly in applications such as spoken dialogue systems.

This work presented the *OnlineLatgenRecogniser*, an extension of the Kaldi automatic speech recognition toolkit. The *OnlineLatgenRecogniser* is distributed under the Apache 2.0 license, and therefore it is freely available for both research and commercial applications. The recogniser and its Python extension is stable and intensively used in a publicly available spoken dialogue system [**?**]. Thanks to the use of a standard Kaldi lattice decoder, the recogniser produces high quality word posterior lattices.

The training script as well as the source code of the *OnlineLatgenRecogniser* is currently merging into Kaldi repository[1][2]. The Alex dialogue system and the integration of *OnlineLatgenRecogniser* is Apache, 2.0 licensed and freely available on Github[3]. The training scripts, the *OnlineLatgenRecogniser* and its Python wrapper *PyOnlineLatgenRecogniser* was developed also under Apache, 2.0 license on Github[4]

The work specified by goals in introduction varies in many aspects. We successfully trained acoustic models, design and also implement decoders interface, improve real-time decoder and prepared experiments for evaluating results.

Future planned improvements include implementing more sophisticated speech parameterisation interface and feature transformations.

Acknowledgments

This research was partly funded by the MEYS of the Czech Republic under the grant agreement LK11221 and core research funding of Charles University in Prague. The work described herein uses language resources hosted by the LINDAT/CLARIN repository, funded by the project LM2010013 of the MEYS of the Czech Republic. We would also like to thank Daniel Povey, Vassil Panayotov, Pavel Mencl, Ondřej Dušek, Matěj Korvas, Lukáš Žilka, David Marek and Tomáš Martinec for their useful comments and discussions.

[1] `http://sourceforge.net/p/kaldi/code/HEAD/tree/sandbox/oplatek2/src/dec-wrap/`
[2] `http://sourceforge.net/p/kaldi/code/HEAD/tree/sandbox/oplatek2/egs/vystadial/`
[3] `https://github.com/UFAL-DSG/alex`
[4] `https://github.com/UFAL-DSG/pykaldi`, `https://github.com/UFAL-DSG/pyfst`

A. Acronyms

DNN Deep Neural Networks . . . 4
SLU Spoken Language Understanding . . . 3
CSR Continuous Speech Recognition . . . 15
ASR Automatic Speech Recognition . . . 3
FST Finite State Transducer . . . 23
DFT Discrete Fourier Transformation . . . 9
GPU Graphics Processing Unit . . . 4
HTK Hidden Markov Model Toolkit . . . 1
EM Expectation Maximization . . . 12
PDF Probability Density Function . . . 23
OOV Out of Vocabulary Word . . . 30
RTF Real Time Factor . . . 20
PCM Pulse-Code Modulation . . . 36
FWRTF Forward Real Time Factor . . . 20
HLDA Heteroscedastic Linear Discriminant Analysis . . . 10
HMM Hidden Markov Model . . . 7
LDA Linear Discriminant Analysis . . . 10
LM Language Model . . . 7
AM Acoustic Model . . . 4
MFCC Mel Frequency Cepstral Coefficients . . . 8
PLP Perceptual Linear Prediction . . . 8
PTI Public Transport Information . . . 39
IID Independent and Identically Distributed . . . 13
MLE Maximum Likelihood Estimation . . . 12
MLLT Maximum Likelihood Linear Transform . . . 10
CMVN Cepstral Mean and Variance Normalisation . . . 10
STC Semi-Tied Covariance . . . 10
ET Exponential Transform . . . 10
MMI Maximum Mutual Information . . . 27
bMMI Boosted Maximum Mutual Information . . . 31
PDF Probability Density Function . . . 23
MPE Minimum Phone Error . . . 31
PLP Perceptual Linear Prediction . . . 8
SER Sentence Error Rate . . . 19
SDS Spoken Dialog System . . . 4
WER Word Error Rate . . . 18
LMW Language Model Weight . . . 28

VAD Voice Activity Detection 39
DM Dialogue Manager 40
TTS Text to Speech 25

B. CD content

- Alex - Alex Dialogue System Framework which I extended by work in following files and directories:
  - *alex/components/asr/kaldi.py* - component interfacing *PyOnlineLatgenFaster*
  - *alex/tools/kaldi/* - training Kaldi scripts very similar to scripts in Kaldi/egs/vystadial
  - *alex/applications/PublicTransportInfoCs/hclg/* - scripts for building decoding graph *HCLG* and running evaluation of ASR engines for Public Transport domain
- Kaldi toolkit - Speech recognition toolkit which I extended by implementing work in following directories:
  - *src/onl-rec* - C++ code implementing *PyOnlineLatgenFaster*
  - *src/pykaldi* - Python wrapper *PyOnlineLatgenFaster*
  - *egs/vystadial/online_demo* - Demos using pre-trained acoustic models and LMs on small test set. The demos run *OnlineLatgenFaster*, *PyOnlineLatgenFaster* and *gmm-latgen-faster*(original batch processing Kaldi executable).

  egs/vystadial/s5 - training scripts for VYSTADIAL data published under TODO license and partially collected on Alex Public Transport domain
- Pyfst - Fork of Python wrapper of OpenFST, where I improved installation and add several simple functions.
- pykaldi-eval - TODO
- thesis.pdf
- Related Published Papers - papers where I am main author or co-author and are related to this work.
- Reference documentation for Kaldi. Code in *kaldi/src/onl-rec* is documented on pages todo in todo pdf.
- Reference documentation for *kaldi/src/pykaldi*. See pages todo in todo pdf
- Reference documentation for Alex. See pages todo in todo pdf.