A decoder for large-vocabulary continuous speech recognition

Master's thesis

Teemu Hirsimäki

Helsinki University of Technology
Department of Computer Science and Engineering
Laboratory of Computer and Information Science
Otaniemi 2002

Preface

This work was done in the Laboratory of Computer and Science of Helsinki University of Technology during years 2001 and 2002. I am grateful to my supervisor and instructor Prof. Mikko Kurimo for giving the possibility to work in the speech group and suggesting this interesting topic for my thesis. I am also grateful to Vesa Siivola for the numerous discussions which gave me many ideas and helped me to concentrate on the relevant things. He also provided the acoustic and language models used in the experiments. Finally, I wish to thank my family for encouraging and supporting me in my studies, and my love, Kristel, for being there for me.

Teemu Hirsimäki
Otaniemi, September 18, 2002

TEKNILLINEN KORKEAKOULU **DIPLOMITYÖN TIIVISTELMÄ**

Tekijä: Teemu Hirsimäki
Työn nimi: A decoder for large-vocabulary continuous speech recognition
Suomenkielinen nimi: Suurten sanastojen dekooderi jatkuvan puheen tunnistukseen

Päivämäärä: 18. syyskuuta 2002 **Sivumäärä:** 46

Osasto: Tietotekniikka
Professuuri: Informaatiotekniikka

Valvoja: Prof. Mikko Kurimo **Ohjaaja:** Prof. Mikko Kurimo

Yksi automaattisten puheentunnistusjärjestelmien keskeisistä osista on dekooderi, jonka tehtävänä on etsiä akustisten mallien ja kielimallien perusteella parhaiten puheeseen sopiva sanajono.

Tässä diplomityössä on toteutettu dekooderi C++-kielellä tutkimuskäyttöön. Erityisesti huomiota on kiinnitetty siihen, että dekooderin kanssa voidaan käyttää suuria sanastoja ja laajan kontekstin kielimalleja. Järjestelmän perustana on niin kutsuttu pinomenetelmä, jossa tunnistushypoteeseja pidetään ajan mukaan järjestetyissä pinoissa. Lisäksi akustinen mallinnus on erotettu kielimallinnuksesta, minkä vuoksi laajojenkin kielimallien liittäminen järjestelmään on mahdollista.

Dekooderin toteutuksen lisäksi tässä työssä kuvataan myös puheentunnistusjärjestelmän muiden tärkeimpien osien toimintaa sekä oleellisimpia ongelmia, joita täytyy ratkaista, kun foneemipohjaisen tunnistuksen sijaan tehdäänkin sanastoihin ja kielimalleihin perustuvaa tunnistusta. Järjestelmän toimivuutta on testattu suomenkielisellä puheaineistolla, ja testeissä on tutkittu sanaston koon ja dekooderin parametrien vaikutusta tunnistustuloksiin.

Avainsanat: dekooderi, puheentunnistus, jatkuva puhe, suuri sanasto, kielimallit, kätketyt Markov-mallit

HELSINKI UNIVERSITY OF TECHNOLOGY

ABSTRACT OF THE MASTER'S THESIS

Author: Teemu Hirsimäki

Title: A decoder for large-vocabulary continuous speech recognition

Title in Finnish: Suurten sanastojen dekooderi jatkuvan puheen tunnistukseen

Date: September 18, 2002	**Pages:** 46

Department: Computer Science and Engineering
Professorship: Computer and Information Science

Supervisor: Prof. Mikko Kurimo	**Instructor:** Prof. Mikko Kurimo

A decoder is one of the central parts of a speech recognition system. Its purpose is to search the word sequence which matches best with the speech given the acoustic and language models.

In this master's thesis, a decoder was implemented for research purposes using the C++ language. The decoder is based on the stack decoding approach, in which the recognition hypotheses are organised in time-ordered stacks separating the acoustic and language modeling, which makes it possible to use large vocabularies and long-range language models.

In addition to the decoder implementation, the thesis covers the other main parts of typical speech recognition systems. The problems arising in large-vocabulary decoding with language models are discussed in contrast to more traditional phoneme-based decoding. The performance of the system is experimented with Finnish speech data, and the effects of the lexicon size and the parameters of the decoder are studied.

Keywords: decoder, speech recognition, continuous speech, large vocabulary, language models, hidden Markov models

Contents

1 Introduction **8**
- 1.1 Speech recognition . . . 8
- 1.2 Typical speech recognition system . . . 9
- 1.3 Background . . . 9
- 1.4 Purpose of the thesis . . . 10

2 Acoustic modeling **11**
- 2.1 Speech signal . . . 11
- 2.2 Extracting feature vectors . . . 11
  - 2.2.1 Short-time power spectrum . . . 13
  - 2.2.2 Mel-frequency cepstral coefficients . . . 13
  - 2.2.3 Delta features . . . 15
  - 2.2.4 Other possibilities . . . 15
- 2.3 Acoustic modeling . . . 15
  - 2.3.1 Hidden Markov model . . . 16
  - 2.3.2 Finding the optimal path—Viterbi search . . . 17
  - 2.3.3 Estimating the model parameters . . . 18
  - 2.3.4 Phoneme-based recognition . . . 18
  - 2.3.5 Context-sensitive models . . . 19

3 Language modeling **20**
- 3.1 Probability of a word sequence . . . 20
- 3.2 N-gram models . . . 21
- 3.3 Topic models . . . 21
- 3.4 Modeling inflections . . . 21
- 3.5 Using language models in decoding . . . 22
  - 3.5.1 Static network expansion . . . 22
  - 3.5.2 Dynamic network expansion . . . 23
  - 3.5.3 Multi-pass decoders . . . 23
- 3.6 Other related work . . . 24

4 Stack decoder implementation **25**
- 4.1 Stack decoding . . . 25
- 4.2 Choosing word candidates . . . 26

4.2.1 Lexicon . . . 26
4.2.2 Comparing the words . . . 27
4.2.3 Pruning the Viterbi search . . . 27
4.2.4 About Viterbi search implementation . . . 28
4.3 Managing stacks and hypotheses . . . 29
4.3.1 Stacks and hypotheses . . . 29
4.3.2 Pruning improbable hypotheses . . . 29
4.3.3 Keeping track of word sequences . . . 30
4.4 N-gram implementation . . . 30

5 Experiments **33**
5.1 Experimental setup . . . 33
5.1.1 Speech data . . . 33
5.1.2 Acoustic models . . . 33
5.1.3 Language model . . . 33
5.1.4 Lexicons . . . 34
5.1.5 Performance measures . . . 34
5.2 Confidence intervals . . . 34
5.3 Experiments on segmented words . . . 35
5.4 Experiments on continuous speech . . . 37
5.4.1 Lexicon and language model test . . . 37
5.4.2 Hypothesis pruning test . . . 39
5.4.3 Acoustic pruning test . . . 40

6 Conclusion **41**

Symbols and abbreviations

$A = \{a_{ij}\}$	Transition probability matrix
$c_i(t)$	ith feature for frame t
$\Delta c_i(t)$	Delta features
$\delta(t, i)$	Probability of the best path ending in state i at frame t
$\gamma(t, i)$	Previous state on the best path to state i at frame t
$f_i(o)$	Observation probability density function of HMM state i
$h(i)$	Hamming window
$\lambda = (A, f, \pi)$	HMM parameters
$\pi(i)$	Initial probability of state i
$s(i)$	Samples of the speech waveform
$S_t(i)$	Short-time Fourier transform of frame t
$O = o_1, \ldots, o_t$	Observation sequence
$Q = q_1, \ldots, q_t$	State sequence
$W = w_1, \ldots, w_n$	Word sequence

HMM	Hidden Markov model
LP	Linear prediction
LPC	Linear prediction coefficients
MFCC	Mel-frequency cepstral coefficients
PLP	Perceptual linear prediction

Chapter 1

Introduction

1.1 Speech recognition

Understanding or transcribing speech is one of the tasks that are easy for humans but appear to be very hard to teach to computers. It is amusing how often the most natural and effortless tasks for humans, like understanding or producing speech, reading hand-written characters, or interpreting other visual images, are the hardest tasks for computers, even if the computational power of computers has grown enormously during the past decades, and computers are able to perform other things far beyond human capabilities.

When people think of speech recognition systems, the most typical application coming to mind is perhaps a dictating machine which converts speech directly to text. However, in many applications, it is not necessary to obtain the exact transcription of the speech. For instance, speech provides a convenient interface for controlling machines if it is not possible to use hands for some reason. Speech recognition can also be used in simple automated services like ticket vending machines or information query systems (see e.g. [1, 10, 12]).

The difficulty of the different speech recognition tasks depends on the following factors:

- **Vocabulary size.** The number of different words the system has to recognise varies typically from a few words up to million words, and affects the difficulty of the task considerably.
- **Type of speech.** It is much easier to recognise single words spoken separately than *continuous speech*. In addition, continuous speech can be *planned* or *spontaneous*, the latter being harder to recognise.
- **Number of speakers.** Building a *speaker independent* system, which can recognise any voice, is much more difficult than building a *speaker dependent* system.
- **Amount of noise.** All kind of noise from traffic, machinery, or other sources makes the recognition task more difficult.

For example, it is a relatively easy task to build a system which recognises digits spoken clearly by a single user in a silent environment (small vocabulary, single words, one speaker, no noise), but devising a system which can transcribe any kind of conversation between several persons in a rush-hour bus would be extremely difficult (open vocabulary, spontaneous continuous speech, many speakers, noise).

1.2 Typical speech recognition system

Figure 1.1 illustrates a typical speech recognition system. The incoming speech signal is first preprocessed and relevant features are extracted from the signal. In the next stage, acoustic models are used to find out how well different phonemes match with the given speech. Last, in the core of the system, lies a decoder, which examines different word sequences and uses the acoustic and language model probabilities to find the overall best word sequence.

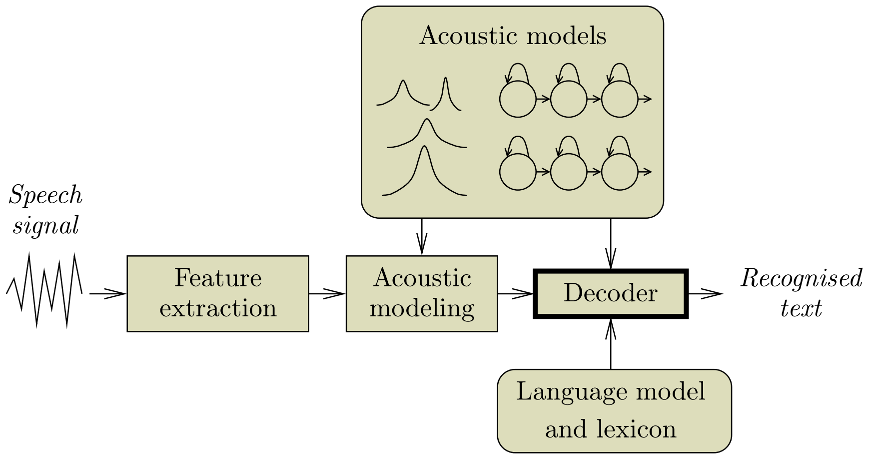


Figure 1.1: An overview of a typical speech recognition system. First, useful features are extracted from the raw audio signal. Then acoustic models are used to compute acoustic probabilities. Finally, the decoder searches the most probable word sequence using the acoustic and language models.

1.3 Background

The speech recognition research in our laboratory started in the late 70's. In [16], one of the first Finnish recognition systems is described. It was based on *Learning Subspace Method* and *Redundant Hash Addressing*, and it could recognise up to 1000 discrete words uttered by one speaker. In the 80's the system was developed further, and hidden Markov models were brought in [34].

In the 90's, the speech recognition research shifted more towards hidden Markov models. Kurimo developed methods to estimate mixture density models using Self

Organizing Maps and Learning Vector Quantization in [19, 20]. During the recent years there has also been active research on developing language models for Finnish speech recognition [33, 22, 21].

1.4 Purpose of the thesis

Since the interest in developing language models for Finnish has grown in our laboratory, and because the old decoders have been mostly phoneme-based, there has been a need for a new speech recognition decoder which could be used to do large-vocabulary speech recognition with the new language models. The purpose of this work was to implement such a decoder and explain what kind of difficulties are involved with large-vocabulary decoding. In addition to allow arbitrary complex language models, special care was taken to keep the system modular in order to make it possible to add further enhancements easily.

Because the system is meant mainly for a research tool instead of a commercial product, there has not been heavy effort to tune the system for real-time use or to implement a complete user interface for desktop use, although there are not any fundamental obstacles for developing the system further in that direction. As such, the current system is more suitable for testing different acoustic or language models, or transcribing large collections of speech off-line.

The emphasis of the thesis is on the decoding process, and although the acoustic and language modeling are crucial parts of a complete speech recognition system, they are covered here only to the extent needed to understand the working of the decoder. How to build and optimise good acoustic or language models are not presented in this work. Also, the models used in the decoding experiments in Chapter 5 can be thought as given data, and instead of focusing on the absolute recognition accuracy, the experiments explore how the settings of the decoder affect the recognition results. For readers who are interested in modeling language and acoustics, references to more detailed descriptions are given in the text.

The thesis is structured as follows. Chapter 2 deals with the acoustic properties of speech and how statistical models can be used with signal processing techniques to model speech signals. Chapter 3 describes statistical language models commonly used in speech recognition and discusses problems arising when we are moving from traditional phoneme-based recognition to word-based recognition with language models. In Chapter 4, the implementation of the stack-based decoder is described, and Chapter 5 presents some experiments to demonstrate that the system works. Finally, Chapter 6 concludes the work.

Chapter 2

Acoustic modeling

In speech recognition, statistical modeling has been found to be a flexible and efficient method for modeling the acoustic and linguistic aspects of speech. For example, instead of asking an expert linguist to write specific rules for recognising different phonemes in a speech signal, we define probabilistic models for phonemes and use a large amount of speech data to mould our models to cover the phonemes in the data as well as possible.

In the following sections, we first take a look at some properties of speech signals and what kind of signal processing methods can be used to dig out features which are essential for recognising speech. Then the hidden Markov model (HMM) and its use in acoustic modeling are explained.

2.1 Speech signal

When speech or any other sound is heard, the ear measures the small changes in the air pressure and converts them into electric signals for the brain. Similarly, a microphone can be used to convert sound signals into electric signals, which then can be processed further with computers.

Figure 2.1 illustrates a speech signal in which the Finnish word OSAKEYHTIÖ is uttered. At the top of the figure, we see a *waveform*, which shows the amplitude of the signal with respect to time. At the bottom of the figure, we see a *spectrogram*, which shows how the power of the signal is distributed over different frequencies. As can be seen from the figure, the distribution varies for different phonemes. For example, /S/ is concentrated on the higher frequencies while the vowels have most of their energy on lower frequencies. These differences are caused by the changes in the shape of the vocal tract when different phonemes are pronounced, which can be used to distinguish phonemes from each other [32].

2.2 Extracting feature vectors

Computers store speech waveforms typically in signals that contain 16 000 samples per second and 16 bits per sample. This kind of signal has clearly all relevant

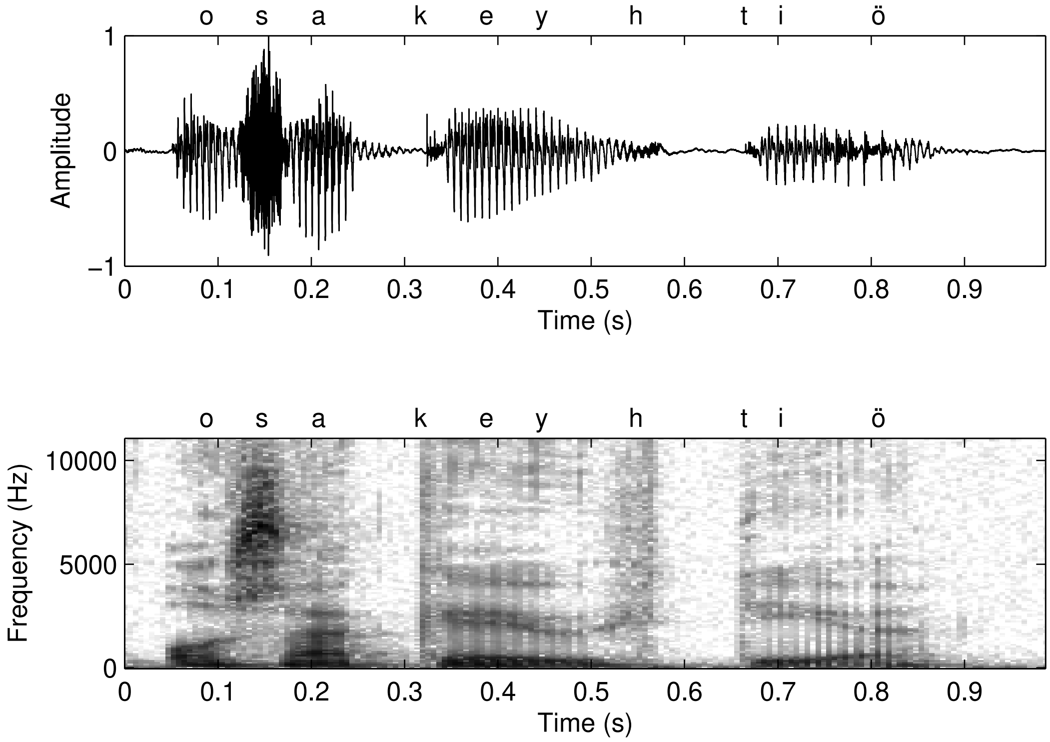


Figure 2.1: An utterance of the Finnish word OSAKEYHTIÖ. The topmost figure shows the waveform and where the phonemes are approximately pronounced. Below, the spectrogram of the same speech signal is shown. Darker color indicates higher power.

information about the spoken words because humans can easily understand the words when the signal is replayed, but it has also a great deal of other information. For example, the gender, the personal voice, and even the mood of the speaker can be heard in the signal, but this information is quite useless for speech recognition. It also takes quite much memory to store speech waveforms as such. For example, one second of speech (16 000 Hz, 16 bits) takes 256 000 bits in total, and the same amount of bits could be used to store over 30 000 characters of raw text if the usual 8 bits were used per character.

Thus, it is necessary to reduce first the amount of the data by extracting essential features of the speech. Ideally, the extracted features should be *invariant* to the irrelevant information in the signal, so that changing the speaker or adding noise to the signal should not change the features considerably if the underlying words are the same. However, it is important to notice that the usefulness of the features depend heavily on the application at hand. In a speaker identification system, the features should naturally retain the information about the speaker's voice, while the spoken words might not be so important.

Nowadays, most speech recognition systems use features based on the power spectrum of the signal [14]. Because the human vocal tract can change shape only at limited speed, the features are normally produced at considerably slower rate compared to the sample rate of the original waveform—typically around 10–20 ms [28], i.e. 50–100 Hz compared to the 16 000 Hz of the original waveform.

Also in this work, the power spectrum is used to compute the basic features, namely the short-time power, 12 mel-frequency cepstrum coefficients, and their derivates. Because these features have been used traditionally in our speech group and they had been implemented already, it was a natural choice to use them in this work. In the next sections, the feature extraction process is described in greater detail. Figure 2.2 shows the overview of the process.

2.2.1 Short-time power spectrum

First, the speech signal $s(n)$ is filtered by a first-order high-pass filter given by

$$\hat{s}(i) = s(i) - 0.95s(i-1). \tag{2.1}$$

The purpose of the filter is to flatten the power spectrum of the signal, which typically decreases towards the higher frequencies [28]. Then, the signal is divided in overlapping frames of 256 samples (16 ms) separated by 128 samples (8 ms) as illustrated in Figure 2.2. Before computing the power spectrum, each frame is scaled with a Hamming window

$$h(i) = 0.54 - 0.46\cos(2\pi i/N), \tag{2.2}$$

where N is the length of the window (256 in our case). This is a standard technique for reducing the distortions caused by the frame borders. Then, the short-time power spectrum can be obtained by computing first the Fourier transform

$$S_t(i) = \sum_{k=0}^{N-1} \hat{s}(k+tN)h(k)e^{-j2\pi ik/N} \tag{2.3}$$

for each frame t and then the short-time power

$$c_0 = \sum_{i=0}^{N/2} |S_t(i)|^2. \tag{2.4}$$

2.2.2 Mel-frequency cepstral coefficients

The power spectrum computed above is also used to obtain the mel-frequency cepstral coefficients. The Mel-scale is a non-linear mapping of frequencies which tries to approximate the critical bands of the human ear and it can be approximated by the following equation [28]:

$$f_{\mathrm{mel}} = 2595\log_{10}\big(1 + \frac{f}{700}\big), \tag{2.5}$$

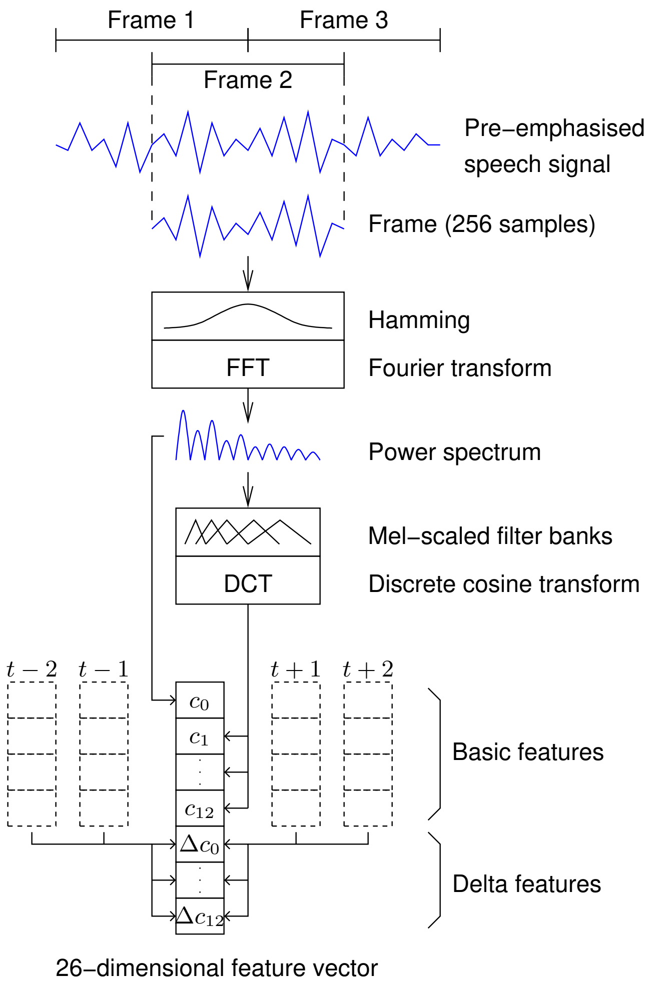


Figure 2.2: The signal is divided in frames, and a 26-dimensional feature vector is computed for each frame. The basic feature set consists of the average power and 12 MFCCs. Delta features are computed from the neighbouring features.

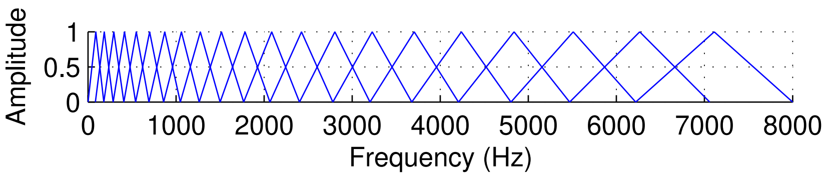


Figure 2.3: The filter bands are located linearly in mel-scaled frequencies, and non-linearly in normal frequencies. The mapping tries to approximate the critical bands of the human ear.

where f is the linear frequency. To compute the mel-frequency cepstral coefficients, 21 triangular bandpass filters are placed linearly to the mel-frequency scale (see Fig 2.3), and the average energy is computed for each band. Finally, a discrete cosine transform is applied to convert the log-energies into 12 cepstral coefficients [28]:

$$c_i = \frac{2}{N} \sum_{k=0}^{N-1} \log |P(k)| \cos\left(\frac{2\pi}{N} ik\right), \tag{2.6}$$

for $i = 1, \ldots, 12$. Above, $P(k)$ denotes the average log-power of the filter band k. In this work, $N = 21$, and the first cepstral coefficient ($i = 0$) was discarded because it is close to the short-time power already computed in c_0.

2.2.3 Delta features

In order to add more dynamic information to the feature vector, the derivates of the 13 basic features are also computed. These rates are traditionally called as *delta features* and they can be computed as follows [28]:

$$\Delta c_i(t) = \sum_{k=-M}^{M} k c_i(t+k), \tag{2.7}$$

where M is the number of neighbouring frames used in delta computation. In this work, the value 2 was used for M.

2.2.4 Other possibilities

Of course, the presented features are not the only possibility, and several other features have been proposed in the literature. For example, using *linear prediction* (LP) to model the speech signal as a linear combination of its previous samples leads to *linear prediction coefficients* (LPC), which can be used to derive *linear prediction cepstral coefficients* (LPCC) [28].

Perceptual linear predictive (PLP) analysis introduced in [13] develops LP further by taking some properties of hearing into account. In [15], so called *RASTA* filtering is added to PLP analysis. Its main idea is to band-pass filter each frequency channel so that slow changes are suppressed making the features more robust.

2.3 Acoustic modeling

For several decades, the hidden Markov model (HMM) [4, 3] has been the most widely used acoustic model in speech recognition. It is a reasonably simple model to be computationally effective but it can capture important properties of speech signals quite well. It has also been studied intensively in many research areas, so its strengths and limitations are well known. The following sections are based on Rabiner's very good tutorial [29].

2.3.1 Hidden Markov model

The hidden Markov model assumes that observations are generated by a finite-state process, which can change state (or stay in the current state) with fixed transition probabilities. Each state has also a fixed observation probability distribution which is used to generate an observation before the process changes state. Figure 2.4 illustrates a 3-state HMM with two-dimensional observation distributions.

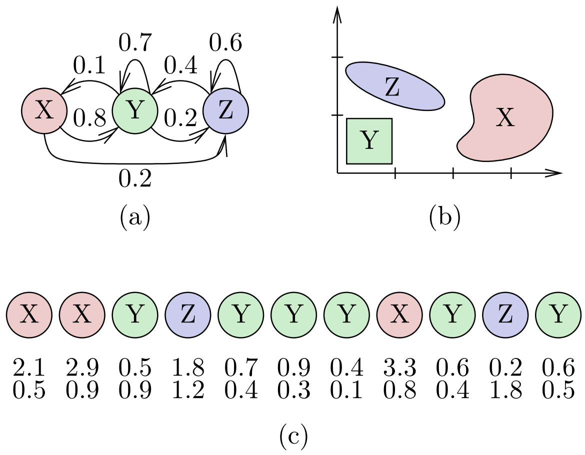


Figure 2.4: (a) A 3-state HMM and the transition probabilities. (b) 2-dimensional observation probability distributions for the states. (c) A randomly generated state sequence and a 2-dimensional observation sequence.

Formally, an N-state HMM is defined by a triple (A, f, π), where A is an $N \times N$ matrix, where an element a_{ij} defines the transition probability from ith state to jth state. The functions $f_i(o)$ define observation probability densities for each state, and $\pi(i)$ defines the initial state probabilities. The observation probability distributions are typically mixtures of Gaussian densities but also other distributions can be used. For example, in [31], a solution based on neural networks is used.

In the HMM framework, there are three fundamental problems, which are interesting in different applications:

1. Compute the probability of the observation sequence $P(O|\lambda)$, given the sequence $O = o_1 o_2 \ldots o_T$ and model $\lambda = (A, f, \pi)$. This can be used to compare different models and to select the model which maximises the probability of the observation sequence. However, we do not need the solution for this problem in this work.

2. Find the optimal state sequence $Q = q_1 q_2 \ldots q_T$, given the observation sequence O and model λ. This is an interesting problem in speech recognition tasks, since by finding the optimal state sequence, we get also the optimal phoneme or word sequence.

3. Given the observation sequence O, find the model $\lambda = (A, f, \pi)$ which maximises $P(O|\lambda)$. The solution for this problem can be used to estimate acoustic models for speech recognition.

2.3.2 Finding the optimal path—Viterbi search

Optimal can be defined in many ways. For example, we could maximise the probability of each state q_t separately but this could lead to state sequences which have invalid transitions. Thus, the most usual way is to find the most probable state sequence [29]:

$$\hat{Q} = \underset{Q}{\operatorname{argmax}}\, P(Q|O, \lambda) \tag{2.8}$$

$$= \underset{Q}{\operatorname{argmax}}\, \frac{P(Q, O|\lambda)}{P(O)} \tag{2.9}$$

$$= \underset{Q}{\operatorname{argmax}}\, P(Q, O|\lambda). \tag{2.10}$$

This can be computed efficiently in a lattice structure. Starting from the first observation, we compute *the probability of the best path ending in each state i, for each time instant t*:

$$\delta(t, i) = \max_{q_1 \ldots q_{t-1}} P(q_1 \ldots q_{t-1}, q_t = i, o_1 \ldots o_t|\lambda). \tag{2.11}$$

The computation can be done as follows.

1. At the first time instant (see Fig. 2.5a), the best path ending in a state i is simply:
$$\delta(1, i) = \pi(i) f_i(o_1), \tag{2.12}$$
where $\pi(i)$ is the probability of starting from state i, and $f_i(o_1)$ is the probability of state i generating the first observation.

2. At the subsequent time instants (see Figs. 2.5b-c), the best path ending in state i, is computed by continuing the previous best paths with a transition to the state i, and choosing the one with highest probability:
$$\delta(t+1, j) = \max_i \Big[\delta(t, i) a_{ij}\Big] f_j(o_{t+1}). \tag{2.13}$$
For each state, we also have to store a pointer to the previous state which produced the best probability:
$$\gamma(t+1, j) = \underset{i}{\operatorname{argmax}} \Big[\delta(t, i) a_{ij}\Big]. \tag{2.14}$$

3. When the last observation is reached (see. Fig 2.5d), the overall best state sequence $\hat{Q} = \hat{q}_1 \ldots \hat{q}_T$ can be computed backwards using the stored pointers. We start from the best state at the last time instant (drawn bold in the figure):

$$\hat{q}_T = \operatorname*{argmax}_i \delta(T, i) \tag{2.15}$$

and backtrack along the stored pointers, i.e. for each frame $t = (T-1), \ldots, 1$:

$$\hat{q}_t = \gamma(t+1, \hat{q}_{t+1}) \tag{2.16}$$

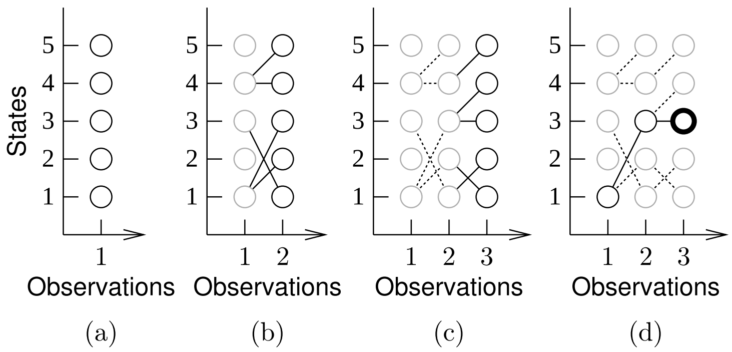


Figure 2.5: The optimal state sequence can be computed in a state-frame lattice, which contains all possible state sequences. (a)-(c) At each frame, the best path is computed and back-pointers are created to store the best path for each state. (d) In the end, the best path can be retrieved by starting from the best state and following the back-pointers.

2.3.3 Estimating the model parameters

The two most common methods to estimate parameters for HMMs given an observation sequence, are the *Baum-Welch* [4, 3] and *segmental k-means* [18, 29]. In both methods, the estimation procedure consist of 2 steps repeated iteratively. First, the model parameters are kept fixed while finding the state sequences which have most likely generated the observation sequence, and then new parameters are computed using the best sequences found in the previous step. The main difference between the two methods is that the Baum-Welch considers all possible paths through a state when the parameters of the state are updated, while the segmental k-means uses only the best path through the observations to tune the parameters. The latter method was used to create the models used in this work.

2.3.4 Phoneme-based recognition

A typical choice in speech recognition applications is to create a separate HMM for each phoneme and use, for example, 3 states per HMM (see Fig. 2.6a). The transitions are often limited to self transitions and transitions to next states. The idea behind this structure is that the first state could model the beginning of the phoneme, which is heavily affected by the previous phoneme. The second state could model some kind of steady state in the phoneme, and the last state could model the ending of the phoneme, which is distorted somewhat by the following phoneme.

Using this kind of models, we can now create a simple phoneme-based recogniser. First, we combine the separate models into a single big HMM by adding transitions between all phonemes as illustrated in Figure 2.6b. The transition probabilities can be estimated from a large collection of text documents. Now, given some training data, the solution to the third problem can be used to estimate the parameters of the HMM. Then we can use the solution of the second problem to recognise new observations.

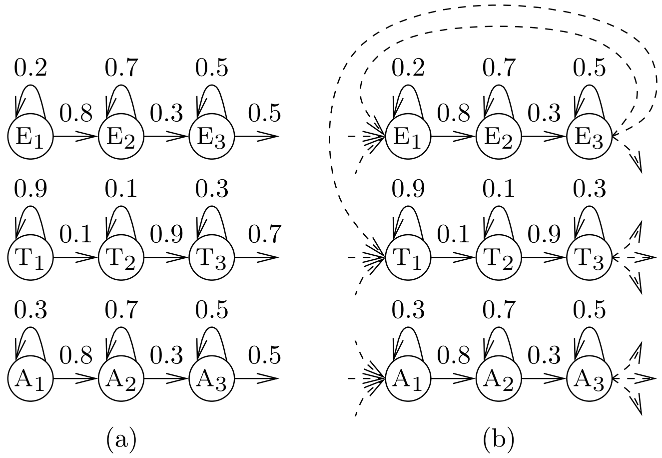


Figure 2.6: (a) 3-state models for three phonemes: /E/, /T/, and /A/. Typically, transitions are limited to self transitions and forward transitions. (b) Phoneme models can be combined into a single big HMM, which can be used to implement a simple phoneme-based recogniser. Transition probabilities between phonemes (dashed arrows) can be estimated from a large text collection.

2.3.5 Context-sensitive models

In normal speech, the pronunciation of a phoneme is often affected by the neighbouring phonemes and, thus, many speech recognition systems use so called *context-sensitive* or *context-dependent* phoneme models. Instead of creating one HMM for each phoneme, several models are created for different contexts. For an extensive survey of training this kind of models, see [26].

Chapter 3

Language modeling

Early speech recognisers were typically phoneme-based and did not take words into account. One might think if the acoustic phoneme models are good enough, it is not necessary to consider words because if the system can recognise phonemes correctly, the words are recognised correctly automatically. This might be possible if the words were pronounced separately and extremely clearly, but in practice, the acoustic models are far from perfect for the following reasons:

1. In continuous speech, the phonemes are heavily affected by the neighbouring phonemes, and it is not always possible to discriminate phonemes without the context.
2. The models are always created using a finite set of speakers, and thus cannot cover all kinds of voices perfectly.
3. In natural speech, people often pronounce words quite carelessly, especially if the context makes the meaning obvious.
4. There are no real inter-word silences in continuous speech, which makes it very hard or even impossible to find the word boundaries without understanding something about the words in the language.

Language models try to address these problems by bringing information about the words and their relations in the recognition process. They can be used to reduce the search space by discarding improbable words in an early stage, which can speed up the recognition process considerably, and if there are several words that are acoustically very similar, language models help to decide which word makes sense with the neighbouring words.

3.1 Probability of a word sequence

Statistical language models assign probabilities for word sequences. In natural languages, some word sequences are much more probable than others, and some sequences appear practically never in normal speech. In general, the probability of a word sequence $W = w_1 \dots w_n$ can be divided as follows:

$$P(W_1^n) = P(w_1)P(w_2|w_1)P(w_3|w_1w_2)\dots P(w_n|w_1\dots w_{n-1}). \tag{3.1}$$

In practice, however, the model above becomes too complex, because the probability of the next word depends on the exact sequence of preceding words, and it would be necessary to have a separate word distribution for every possible history. It would be impossible to estimate such a model well, or even store it efficiently. A common solution to this problem is to group some histories together, and estimate a common distribution for each group.

3.2 N-gram models

One idea to group histories is to take only a few preceding words into account, which leads to so called N-gram models. All histories which have the same last $n-1$ words are grouped together, even if they differ in the earlier words. For example, in the case of 3-gram, the probability of a single word becomes:

$$P(w_i) = P(w_i|w_{i-2}w_{i-1}). \tag{3.2}$$

2-gram and 3-gram models are surprisingly effective in languages like English, in which the word order is quite strict, and they have been the most widely used language model in speech recognition for several decades. A good introduction for N-gram models can be found in [23], and an exhaustive survey of techniques for training and optimising N-gram models can be found in [7].

3.3 Topic models

In N-gram models the complexity of the model was reduced by taking only a few preceding words into account and discarding the earlier words. Another approach is to take more preceding words into account and to discard the word order. The idea behind these, so called *topic models* or *bag-of-words* models, is to model the word probability according to the content words which have appeared earlier in the speech. For example, if the last sentence contained words *jungle* and *monkey*, it is not very likely that *biathlon* and *wrestling* would appear very soon. A few topic-based models can be found in [5, 11, 21].

3.4 Modeling inflections

In languages like Finnish, Czech or Turkish, which use inflections extensively, the traditional dictionary-based language modeling does not work so well. In these languages, many English words like *in*, *into*, *from*, or *without*, can be expressed with suffixes, which increases the amount of different word forms considerably. There has been some research on developing language models to overcome the inflection problem. In [6], for example, the words were divided in morphemes and a language model was created for morphemes treating them as word units in order to model Czech.

3.5 Using language models in decoding

Given an observation sequence, the decoder could, in principle, try all possible word sequences, and compute their probabilities using the acoustic and language models. However, this is not possible in practice because the number of possible word sequences is enormous even for short observation sequences when large vocabularies are used. In the next sections, two main search approaches according to [2] are described.

3.5.1 Static network expansion

One possibility to add language modeling to the decoding process is to extend the phoneme-based recogniser presented in Section 2.3.4. In the same way as the phoneme models were used to create a big HMM, language models can be build statically into the HMM, creating so called *word networks* [39]. For instance, in order to use a 2-gram language model, in which the probability of the word depends on the previous word, we can add new states to the network so that each word has an unique path (see Fig. 3.1), and it is possible to add language model probabilities to the transitions between words (dashed arrows in the figure). Even though the

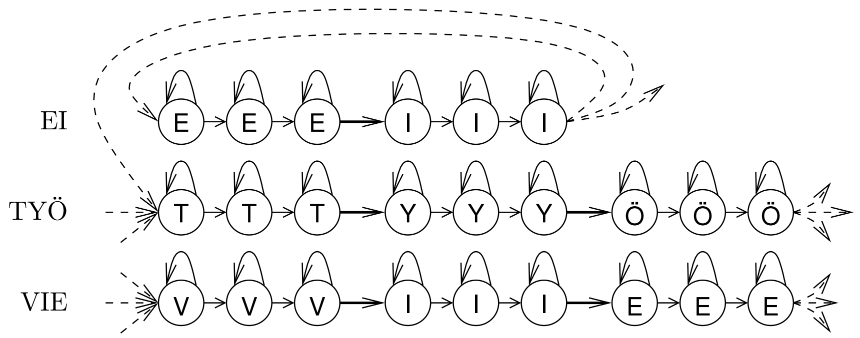


Figure 3.1: A 2-gram language model can be incorporated in the decoding process by adding states to the network so that each word has an unique path. Then the language model probabilities can be added in the transitions between words (dashed arrows).

network becomes much bigger, it is still possible to use the Viterbi search to find the best word sequence, and the word probabilities given by the 2-gram model are taken correctly into account in the transitions between the words.

The main advantage of building language models into the network is that the decoding process becomes quite straightforward, and all information is built neatly into a single framework. Also, the language model probabilities can be taken into account in an early stage, which makes it possible to prune the search more effectively.

However, as can be seen from the above, the more history we want to preserve the more states we need in our HMM. For example, in order to build a 3-gram model into the network, it is not enough to have a unique path for each word, as in the 2-gram case. Instead, each word must have a parallel copy for *every* possible predecessor word, and the number of states grows enormously. For this reason, it is quite hard to build long-range language models effectively into the network, which is the reason this approach is not used in this work. Nonetheless, there has been research to find efficient ways to store big networks. For instance, Mohri [25] has studied and developed methods to build and optimise static networks so that real-time speech recognition speed can be achieved.

A good example of a state-of-the-art recognition system using static networks is the well-known HTK [38, 39] (HMM Toolkit), which contains, in addition to the decoder, tools for acoustic and language models, different front-end parameterizations and transcriptions. HTK is designed primarily for speech recognition but it can also be used for other HMM applications.

3.5.2 Dynamic network expansion

Aubert [2] uses the term *dynamic network expansion* to describe approaches in which the language model is not built into the network prior to decoding. Instead, only the vocabulary without language model is built statically, and then the network and the Viterbi search can be used to find the acoustically best word candidates at different times. Because the small network can not preserve enough word histories to add language model probabilities, the decoder needs to keep track of the most promising word sequences, often called as *hypotheses*, until the language model probabilities can be added. This approach is also used in this work, as described in more detail in the next chapter.

The biggest benefit from separating language modeling from the network is that arbitrary complex language models can be used without affecting the Viterbi search. However, optimising word boundaries and using language models to prune acoustic matching becomes more difficult [2].

This approach is has been quite popular recently and is used, for instance, in Noway [31] and Ducoder [36]. The Noway system uses also neural networks, instead of traditional Gaussian mixtures, to model the observations densities.

3.5.3 Multi-pass decoders

There exist also so called *multi-pass* decoders which use different search methods on successive passes. For example, the Sphinx-II system, developed at Carnegie Mellon University, is a 3-pass decoder. In the first pass it uses Viterbi search with 2-gram or 3-gram language models to generate a word lattice which contains the acoustically best words. Then a backward Viterbi search is used to find multiple starting times for the words generated by the first pass. In the last pass, the best word sequence is searched without acoustic rescoring, and arbitrary complex language models can be used to choose between the words found in the first two passes. [30]

The main advantage in multi-pass decoders is that more complex models can be used efficiently on the most potential hypotheses, because a great deal of the search space is already discarded on the first passes. The main problem, however, is that the errors made in the first passes can not be recovered later.

3.6 Other related work

In [37], Woszczyna made an extensive study of speeding up the recognition process. Up to 200-fold speed-up was obtained using lookahead techniques to estimate acoustic and language model probabilities, optimising the computation of observation probabilities, and utilising frame skipping techniques among other things.

In addition to the recognition speed optimization, Ravishankar [30] studied methods for decreasing memory requirements in large vocabulary speech recognition tasks. The speed-up was achieved by improving the lexical tree search and using post-processing techniques to recover possible errors. The methods to obtain memory savings included language model caching, and compressing acoustic model presentations.

Chapter 4

Stack decoder implementation

4.1 Stack decoding

In this work, one of the main design principles of the decoder was the possibility to use complex long-range language models. Thus, instead of building the language model in the network, as described in the previous chapter, the approach of separating it from the Viterbi search was chosen along the lines of [36].

The main idea is illustrated in Figure 4.1. The decoder stores the most probable hypotheses in so called *stacks*, so that each time frame has a separate stack. The hypotheses in a stack can be expanded by single words by using the Viterbi search to find the best words starting at the corresponding frame. For example, in Figure 4.1a, the decoder expands the first stack containing the initial empty hypothesis. Viterbi search is used to find the acoustically best five words (EN, ON, ENSIN, ETSIN, ENSIMMÄINEN), which are then used to create five new hypotheses. The new hypotheses

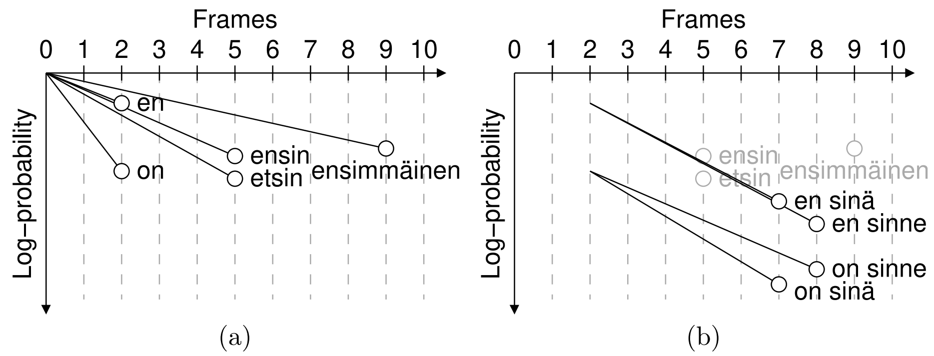


Figure 4.1: (a) The decoder uses Viterbi search to find acoustically best words. Each frame has a stack, and the best hypotheses are placed in the stacks according to the best ending times. (b) The decoder can expand all hypotheses in a stack by starting the Viterbi search at the corresponding frame, and expanding all hypotheses with the best words.

are placed in the stacks according to the best ending frames found by the Viterbi search, as illustrated in the figure. Note, that the word boundaries are not known in advance and because typical continuous speech does not contain actual inter-word silences, there are often several short and long words which match the acoustic data quite well. In this imaginary example, the speech actually contains the word ENSIMMÄINEN, but the other words found by the Viterbi search match also the speech data reasonably well.

In Figure 4.1b, the decoder has completed the first stack expansion and proceeds to frame 2, which contains currently two hypotheses. Again, the Viterbi search is used to match the lexicon to the speech, this time starting at frame 2, and all hypotheses in the stack are expanded with the acoustically best words. By repeating these two steps, the decoder processes the speech until the end is reached.

4.2 Choosing word candidates

4.2.1 Lexicon

In order to limit word expansions to valid words, the decoder needs a lexicon which contains the words and the corresponding phoneme sequences, i.e. the HMM state sequences. In Finnish, it is possible to deduce the phoneme sequence automatically from the written form of the word, but in English, manually created pronunciation dictionaries are needed, because similarly written words may have quite different pronunciations, while some differently written words are pronounced in the same way. Also, some words have several alternative pronunciations.

Figure 4.2 shows a simple lexicon of eight Finnish words. In order to save memory and make computations somewhat faster, the lexicon is usually stored in a tree structure, by merging the common prefixes of the words. As we remember from the previous sections, the Viterbi search maintains the best path up to the current frame

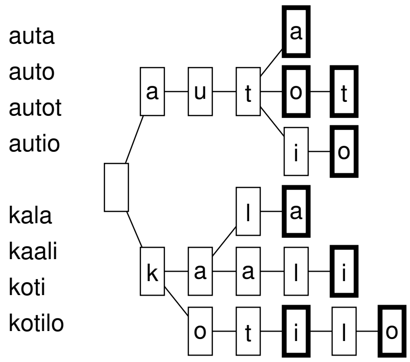


Figure 4.2: A simple eight-word lexicon. The words are organised in a tree structure by merging the common beginnings in order to make computations more efficient. The bold boxes mark the word endings.

for each state, so the best paths are obtained for each word as long as each word has a unique end state (marked with bold boxes in the figure). The tree-structured lexicon is a standard technique used in almost all modern decoders.

In the decoder, the lexicon tree is used to build a big HMM by replacing the phonemes in the tree by the corresponding HMM state sequences and taking the transition probabilities into account. Then the lexicon HMM can be fitted to the acoustic data starting from a given frame simply by using the Viterbi search.

4.2.2 Comparing the words

When the decoder chooses a stack for expansion, the Viterbi search is started at the frame corresponding to the stack. Because the best ending frames of the words are not known beforehand, and the ending frames will most likely differ for different words, the search must be performed so far that even the longest words in the lexicon can be fitted. One possible solution is to approximate the maximum duration of the longest word in the lexicon, and to search always the same fixed amount of frames.

After performing the Viterbi search, the decoder must select the best words and ending frames for expanding hypotheses. One possibility is to compute average log-probability for each frame and word, and choose an ending frame for each word, so that the average log-probability is minimised, as illustrated in Figure 4.3. To limit the size of the word candidate list, the following parameters were implemented in the decoder:

1. *Word limit*: Only N acoustically best words are used to expand hypotheses.
2. *Word beam*: Discard words whose average log-probability differs more than beam width B from the average log-probability of the best word.

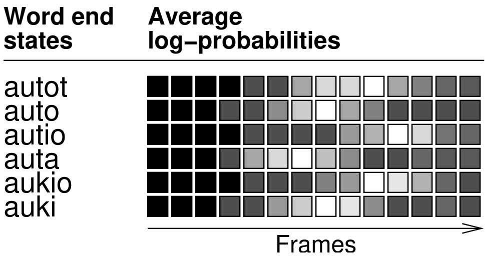


Figure 4.3: For each word, the ending time is chosen by finding the frame which produces the best log-probability averaged over the frames. Brighter color indicates higher probability.

4.2.3 Pruning the Viterbi search

When large vocabularies are used, the word expansion phase becomes easily the most computationally expensive part of the decoder. However, it is possible to omit significant parts of the computation without degrading the recognition results considerably. In this work, the following two standard pruning methods are used at each frame:

1. *State limit*: Keep only N best states, and forget the others.
2. *State beam*: Discard states whose log-probabilities differ more than beam width B from the best state.

It has been suggested that language model probabilities should be taken into account already in the word expansion phase so that the improbable words could be pruned as early as possible. However, due to the stack approach used in this work, it is quite impossible to compute language model probabilities in the word expansion phase, because the preceding word sequence is ambiguous. The best words found in the Viterbi search are used to expand several hypotheses in the current stack, and the hypotheses will most likely have different word histories. If the language model were to be utilised in the expansion phase, the Viterbi search should be done separately for each group of hypotheses which have different language model context [2], but this would be hard with complex language models.

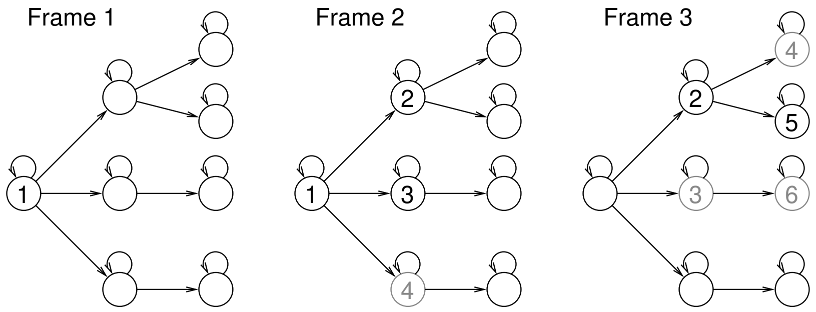
Also, even if the word history was fixed in the Viterbi search, using the language model probabilities in the Viterbi search would be far from trivial. Because of the tree-structured lexicon, the identity of the word is not known until the word end state is reached, which makes it hard to compute the language model probabilities during the search. Various look-ahead techniques have been proposed to overcome this problem (see e.g. [27]).

4.2.4 About Viterbi search implementation

The Viterbi search is usually described to be performed in a lattice structure [40]. This works well if the network is small enough, but when large vocabularies are used, the resulting network has so many states that the lattice would consume too much memory, and computing the whole lattice would be too slow. However, it is possible to be organised more effectively.

In this work, the *token passing* method [40] is used to implement the Viterbi search. Instead of doing the search in a lattice, each state of the search network is capable of holding a so called *token*, which contains the log-probability of the best path to its state (see Fig 4.4). Initially, only the first state of the network contains a token, and at every time instant, each token is copied along the possible transitions updating the probability of the token, and for each state, only the best token is stored. The decoder also keeps a sorted list of the tokens, which makes it possible to prune improbable tokens. For example, in the figure, the decoder stores at most three best tokens, and discards also tokens whose log-probability is more than 1.0 worse than the best token in list.

Note that in the normal form, the Viterbi search produces also the best state sequence. In that case, it is necessary to store information about the states through which the tokens have been passed, but in our case, the actual state sequence is



Token	Log−prob
1	−0.98

Token	Log−prob
2	−1.23
3	−1.34
1	−2.21
4	−2.87

Token	Log−prob
2	−1.56
5	−1.79
3	−2.89
4	−3.12
6	−3.19

Figure 4.4: The token passing method. At each frame, the tokens are copied along the transitions, and the probabilities are updated. Only the best token is stored for each state, and improbable tokens are pruned.

not needed. Because of the tree-structured lexicon, the previous states are unique whatever state we are in.

4.3 Managing stacks and hypotheses

4.3.1 Stacks and hypotheses

Because the word sequences are not managed by the acoustic matching part, which was the main idea of stack decoding, the decoder has to store the full-word hypotheses in separate data structures. In this work, the hypotheses are ordered in frame-wise stacks according to their ending times. This makes it possible to use a single Viterbi search to expand all hypotheses ending in the same frame, because the word expansion depends only on the ending frames of the hypotheses, and not on the word histories. The decoder chooses always the earliest stack for expansion but there are also other possibilities (see e.g. [36]).

4.3.2 Pruning improbable hypotheses

In theory, we would like to preserve all hypotheses to the end, but in practice, the more hypotheses the decoder has to expand, the slower the decoding process becomes, so it is necessary to prune the most improbable hypotheses regularly. In this work, the following prunings are applied before a hypothesis is expanded with the acoustically best words:

1. Discard the hypothesis if there exists a more probable hypothesis with the same language model context in the stack.

2. *Hypothesis limit*: Discard the hypothesis if it is not among the N best hypotheses in the stack. This keeps the number of hypotheses reasonable when there are many good hypotheses in the stack.

3. *Hypothesis beam*: If the log-probability of the hypothesis is too far from the best hypothesis, discard it. This is useful, when there are many bad hypotheses in the stack in addition to the good ones.

4. *Global beam*: Interpolate the globally best hypothesis to the current stack and discard hypotheses falling too far from the interpolated hypothesis. This prunes hypotheses from a stack which contains only bad hypotheses, when there is a much better stack in the later frames.

The pseudo code of the search algorithm and the location of the prunings are shown in Figure 4.5.

4.3.3 Keeping track of word sequences

In order to compute language model probabilities during the decoding process and to output the best word sequence, the decoder has to keep track of the word sequences of the hypotheses. Because the number of the active hypotheses can grow up to 10 000 or even 100 000, it is not efficient to store the word sequences separately for each hypothesis. Instead, the word sequences are stored in a tree structure in the same way as the state sequences are stored in [40].

Figure 4.6 illustrates the idea. Each node contains a word index, a pointer to the previous node, and a reference counter which makes it possible to detect and discard unused nodes when hypotheses are pruned. The counts can also be used to detect and output the recognition result during the search process as soon as the words get fixed in the root of the tree: starting from the root, all consecutive nodes pointed only by one node are known to be in the final hypothesis.

If more information about the decoding process is desired, the starting times of the words and accumulated log-probabilities can also be stored in the nodes.

4.4 N-gram implementation

Because high-order N-gram models are typically very sparse [7], it is a common practice to build the model in a tree structure, and in this work, a structure based on [35] is used. As shown in Figure 4.7, the 1-grams are first stored in an array containing the word, the 1-gram probability, the back-off weight, and an index of the first 2-gram starting with this word. Similarly, the 2-grams have the probability information and an index to the 3-gram table.

In [35], some additional optimizations were proposed. For instance, it is unnecessary to store the first column in the 1-gram table, and the last two columns in the

```
stack ← the first stack containing hypotheses
sort hypotheses in stack
retain only the best hypothesis for each language model context in stack
best ← interpolate the hypothesis with the best average log-probability
remove hypotheses worse than best.logprob + globalbeam from stack
best ← the best hypothesis in stack
remove hypotheses worse than best.logprob + hypobeam from stack
do the Viterbi search starting from frame stack.frame
wordlist ← the word candidates found in Viterbi search
for each hypo in stack do
   for each word in wordlist do
      newhypo ← copy of hypo
      append word to newhypo
      add language model probability to newhypo
      insert newhypo to stack at frame word.targetframe
   end for
end for
remove hypotheses from stack
go to step 1
```

Figure 4.5: An outline of the search algorithm.

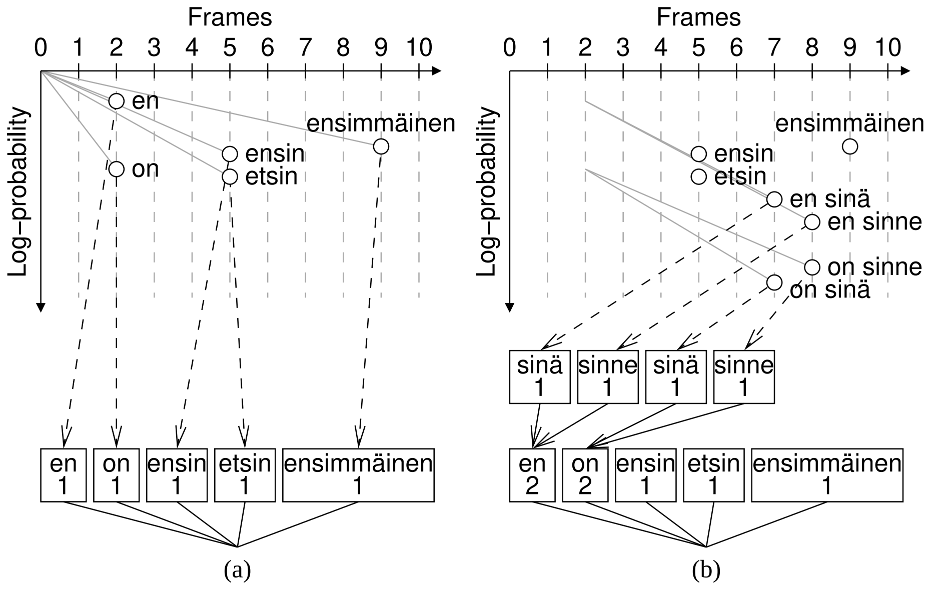


Figure 4.6: The word sequences are stored in a tree structure. When a hypothesis is expanded, a new leaf node is created for each new word. Each node contains also a pointer to the previous node and a reference counter which makes it possible to free unused nodes when hypotheses are pruned.

highest order table. Also, the probabilities can be quantized to save memory with no loss in performance. However, as memory requirements were not the primary concern in this thesis, these optimizations were not implemented.

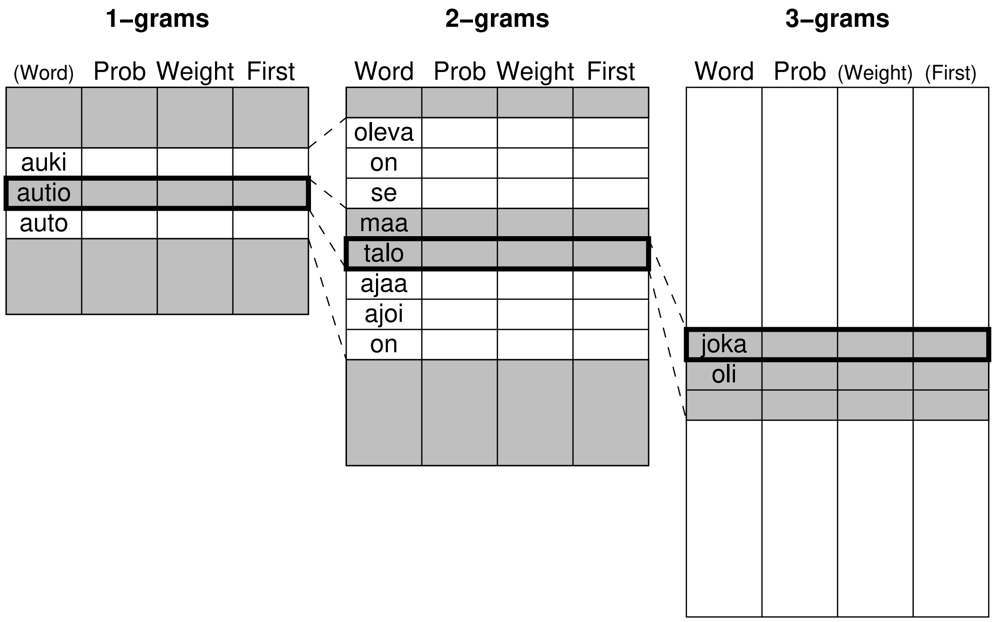


Figure 4.7: Fetching the 3-gram AUTIO TALO JOKA from a tree-structured model.

Chapter 5

Experiments

5.1 Experimental setup

5.1.1 Speech data

The speech data used in the experiments was a Finnish book read aloud by a female speaker. The first 11 hours and 50 minutes (49375 words) were used to teach the acoustic models, and the next 28 minutes (1929) words) were used as test data. The test data was divided in 20 segments, each containing 8 sentences.

5.1.2 Acoustic models

Context-sensitive HMMs were created for the following Finnish phonemes: /A/, /B/, /D/, /E/, /F/, /G/, /H/, /I/, /J/, /K/, /L/, /M/, /N/, /O/, /P/, /R/, /S/, /T/, /U/, /V/, /Y/, /Ä/, /Ö/, /NG/, and inter-word silence. In addition, a context-insensitive model model was created for longer silence. Each HMM had 3-states, except the inter-word silence, which had only one state.

Each state had a observation probability density consisting of a mixture of 4 Gaussian densities. The mixture densities were initialised using Self Organizing Maps as described in [20], and then the models were trained using the *segmental k-means* algorithm [29, 18].

5.1.3 Language model

The language model used in the experiments was a 3-gram model trained using a collection of Finnish books, journals and newspaper articles. The training data contained 40 million words and 1 million different word forms in total. The 64000 most frequent word forms were used in the language model, which was created using the SLM toolkit [8].

Even if the language model contains 64000 word forms, 22% of the word forms in the test data are not found in the language model. The biggest reason for this are the inflectional forms used in Finnish.

5.1.4 Lexicons

In order to study the effect of the vocabulary size to the overall performance, four lexicons of different sizes were created. The smallest lexicon contained all 17911 words that appeared in the speech data (training and test data). Then three lexicons of 20000, 40000, and 60000 words were created by adding the most frequent words from the language model until the desired size was reached.

5.1.5 Performance measures

The performance measures of speech recognition systems depend on the application at hand but the three most interesting factors are the *recognition accuracy*, *recognition speed*, and *memory requirements*. These three factors are typically in conflict so that one can be improved at the cost of the other two and it is not easy to improve two or three factors at the same time. [30]

In this work, the emphasis is on the recognition accuracy and recognition speed. The accuracy is measured by the *word-error rate* and the *letter-error rate*. The former is obtained by computing first the *Levenshtein distance*, i.e. the minimum number of *insertions*, *deletions*, and *substitutions* of words needed to convert the correct word sequence to the recognition output, and dividing the result by the number of words in the correct sequence. The letter-error rate is computed similarly using letters instead of words. It may be more appropriate measure in dictation applications, for example, because the user may have to correct the recognition result manually.

The recognition speed is measured by the real-time factor, which tells how many seconds does it take to recognise one second of speech. In the experiments, the acoustic probabilities were computed beforehand, so the time measurements include only the decoding process. However, the acoustic probabilities used in the experiments can be computed in real-time.

There are also other measures proposed in the literature. In *spoken document retrieval* tasks [9] or *dialogue systems* [1], it is not so crucial to get all words right, as long as the most important content words are recognised correctly. In such cases, the *term-error rate* is often a better performance measure [17, 22]. It ignores the order of words and gives more weight for the content words. Another modified measure is the *base-form error rate* [22], in which the words are returned to the base forms before computing the word-error rate.

5.2 Confidence intervals

In order to analyse the statistical significance of the results, the Student-t distribution was used to compute 95%-confidence intervals for the estimated means under the assumption that the variance is unknown. The bounds for the $100(1-\alpha)\%$ confidence interval is given by

$$\bar{X} \pm t_{\alpha/2} S/\sqrt{n}, \tag{5.1}$$

where $\bar{X}$ is the estimated mean, S is the estimated variance, and n is the number of samples. In our case, $n = 20$ (the number of speech segments), $\alpha = 0.05$, and the Student-t distribution gives the value $t_{\alpha/2} = 2.093$. [24]

5.3 Experiments on segmented words

The purpose of the first test is to study how the lexicon size and the pruning level affect the quality of the word candidate lists generated by the word expansion module. The speech data was segmented into words, and for each segment, a list of acoustically best words was generated without any language model. Three pruning levels (FAST, MEDIUM, SLOW) shown in Table 5.1 were tested for each of the four lexicons.

Level	**State limit**	**State beam**
FAST	5000	20
MEDIUM	7500	35
SLOW	10000	50

Table 5.1: The settings of three pruning levels used in the test.

The test was performed first so that the starting and the ending times of the words were known beforehand, i.e. the decoder was forced to span recognition result over the whole segment. Table 5.2 shows the results. Then, the same test was repeated but this time only the starting times of the segmented words were given to the decoder, so the decoder had to select best ending times for each word. The results are shown in Table 5.3. For each lexicon and pruning level combination, the following measures are shown:

- First: the percentage of the segments in which the correct word was at the top of the word list, i.e. it had the highest probability.
- Top 50: the percentage of cases in which the correct word was among the top 50 words.
- Position: the average position of the correct word on the list.
- Real-time: the real-time factor of the recognition process, i.e. how many seconds does it take to recognise one second of speech.

As can be seen in the results, all measures get worse when the size of lexicon is increased. The main reason for this is probably that larger lexicons contain more acoustically similar words and, because of the imperfect acoustic models, these words get easily mixed.

Also, as expected, more aggressive pruning makes the recognition faster but the accuracy worse. However, it seems that the average position on the best word list gets actually better when the pruning level is increased, which might sound surprising at first. The reason for this is that words falling out of the list are not counted in the

Lexicon	Pruning	First (%)	Top 50 (%)	Position	Real-time
17911	FAST	82	94	2.3	0.4
17911	MEDIUM	84	99	2.6	0.5
17911	SLOW	86	99	2.6	1.4
20000	FAST	81	94	2.4	0.2
20000	MEDIUM	84	99	2.9	0.6
20000	SLOW	85	99	2.9	1.6
40000	FAST	75	94	2.6	0.3
40000	MEDIUM	77	98	3.1	0.9
40000	SLOW	79	99	3.1	2.7
60000	FAST	73	94	2.7	0.3
60000	MEDIUM	76	98	3.3	1.1
60000	SLOW	77	99	3.4	3.9

Table 5.2: The recognition results of the segmented words with known ending times. For each lexicon and pruning level, the percentage of correct words, the percentage of correct words on the list, average position of the correct words on the list, and recognition time are shown.

Lexicon	Pruning	First (%)	Top 50 (%)	Position	Real-time
17911	FAST	53	95	3.0	0.4
17911	MEDIUM	54	99	3.2	0.5
17911	SLOW	55	100	3.3	1.6
20000	FAST	49	95	3.6	0.4
20000	MEDIUM	49	99	3.9	0.9
20000	SLOW	50	100	4.0	1.8
40000	FAST	40	95	4.5	0.4
40000	MEDIUM	41	99	4.9	1.0
40000	SLOW	40	99	5.3	3.0
60000	FAST	37	94	5.0	0.5
60000	MEDIUM	38	98	5.7	1.3
60000	SLOW	38	99	6.2	3.7

Table 5.3: The recognition results of the segmented words when only the starting times of the words were known. For each lexicon and pruning level, the percentage of correct words, the percentage of correct words on the list, and average position of the correct words on the list, and recognition time are shown.

average, so in the SLOW case there are some barely recognised words which increase the average, and do not get on the list in the FAST case.

Comparing the two tables, we can see that when the ending times of the words are not known beforehand, the top of the list contains more wrong words. This is natural, because many longer Finnish words contain shorter words as a substring. However, it is interesting that even though the ending time is not known, the average position of the correct word remains reasonably high.

5.4 Experiments on continuous speech

5.4.1 Lexicon and language model test

First, the effect of lexicon size and the language model scaling was studied while keeping the pruning parameters fixed. The language model scaling factor defines how the language model log-probabilities are scaled before they are combined with the acoustic log-probabilities. Table 5.4 summarises the parameters used in the test, and Figure 5.1 shows the results of the test.

On the left part of the figure, we see the how the average word-error rates of the 20 test segments are heavily affected by the lexicon size if no language model is used (scale factor 0), but when language model is used, the effect is smaller. On the other hand, the lexicon size does not seem to affect letter-error rates at all when the language model is used.

The right part of the figure shows the average difference of the word-error rates. For each of the 20 segments, the error rates of the 60000-word and 40000-word lexicons are compared to the 17911-word lexicon and the average difference is computed. This shows that, even if the error-rates vary quite much between the segments and the confidence intervals on the leftmost figure get overlapped, the results of the bigger lexicons are consistently worse. The main reason for the increase in the word-error is probably that larger lexicons simply contain more acoustically similar words which get confused in the acoustic matching. However, the language model reduces the size of the lexicon by penalising the improbable words even if they match quite well acoustically.

Pruning	**Values**
State limit	5000
State beam	20
Hypothesis limit	20
Hypothesis beam	5
Global beam	5
Lexicon size	17911, 40000, 60000
LM scaling	0,2,4,...,16

Table 5.4: The parameters used in the lexicon and language model test.

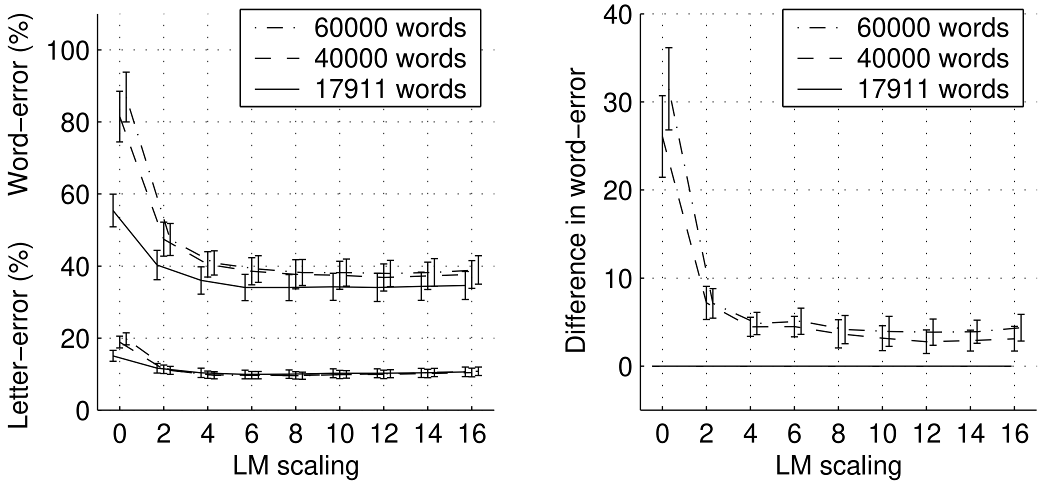


Figure 5.1: The effect of the lexicon size and language model scaling.

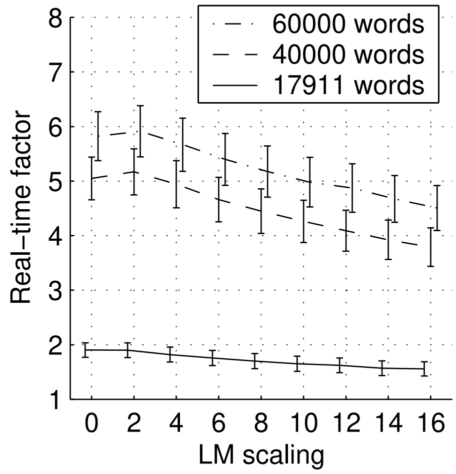


Figure 5.2: The effect of the lexicon size and language model scaling on the real-time factor.

Figure 5.2 shows how the real-time factor changes with the lexicon and language model. Naturally, a bigger lexicon makes the recognition slower, because the recognition network has more states, but it is interesting that increasing the scaling factor of the language model seems to make the recognition faster. One reason might be that giving more weight to language model probabilities pushes improbable hypotheses earlier outside the hypothesis beam thus reducing the search space.

5.4.2 Hypothesis pruning test

In the next test, the hypothesis pruning level was tested while the acoustic pruning parameters, the lexicon, and the language model were fixed. Table 5.5 summarises the used parameters, and Figure 5.3 shows the results.

As can be seen in the figure, the pruning level does not seem to affect the error rates noticeably. It implies that the language model used is not good enough to

		Pruning levels					
Pruning	**Fixed**	**1**	**2**	**3**	**4**	**5**	**6**
State limit	5000						
State beam	20						
Hypothesis limit	–	1	10	20	40	100	200
Hypothesis and global beam	–	1	2	5	10	15	20
Lexicon size	40000						
LM scaling	12						

Table 5.5: The fixed parameters and the six pruning levels used in the hypothesis pruning test.

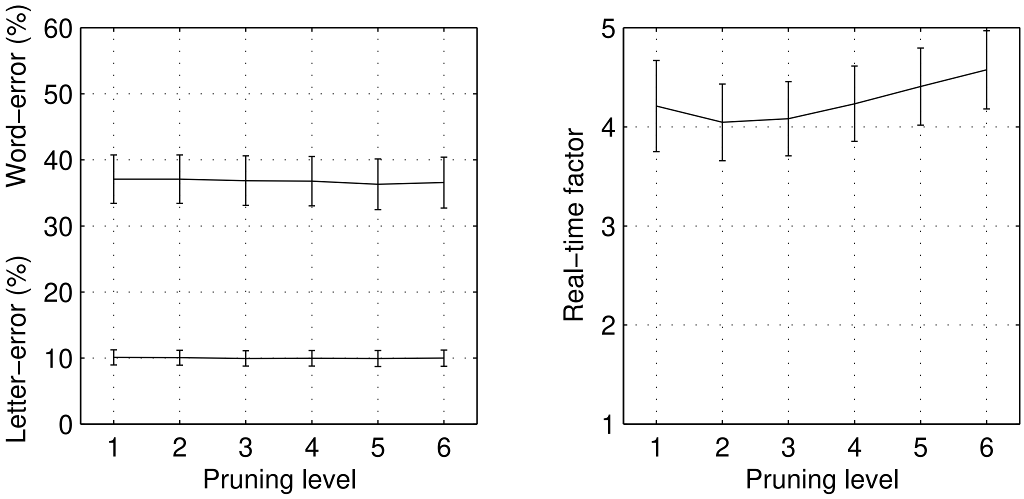


Figure 5.3: The effect of the hypothesis pruning level.

correct mistakes made by the acoustic matching even if it is given a chance. Also, the real-time factor does not seem to change much as can be seen in the rightmost figure.

5.4.3 Acoustic pruning test

In the last test, the acoustic pruning parameters were tested, as shown in Table 5.6. Figure 5.4 shows the results. In the figure we see, how the error rates and the recognition speed change along with the pruning level. The most radical effect can be seen when between levels 1 and 3 when the real-time factor grows near 5. On the levels 4, 5, and 6, small improvements can still be achieved but the computational cost is quite high.

		Pruning levels					
Pruning	**Fixed**	**1**	**2**	**3**	**4**	**5**	**6**
State limit	–	3000	4000	5000	6000	7000	8000
State beam	–	10	15	20	25	30	35
Hypothesis limit	40						
Hypothesis beam	10						
Lexicon size	40000						
LM scaling	12						

Table 5.6: The fixed parameters and the six pruning levels used in the acoustic pruning test.

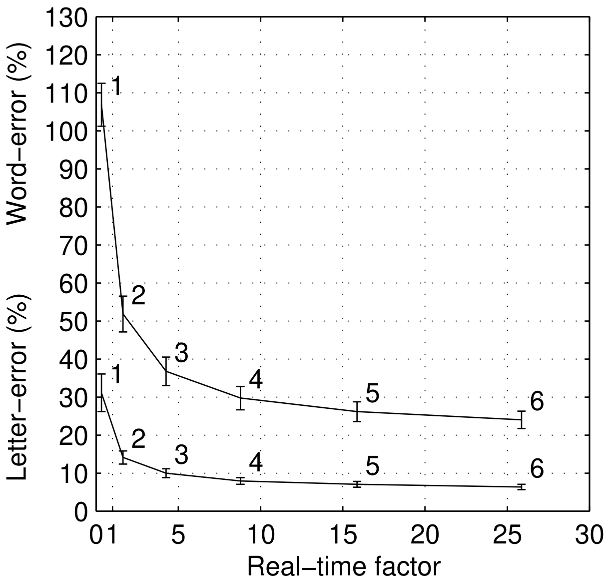


Figure 5.4: The effect of the acoustic pruning level.

Chapter 6

Conclusion

The purpose of this work was to implement a decoder for continuous large vocabulary speech recognition tasks in a hidden Markov model framework. A modular stack-based decoding approach was chosen along the lines of Ducoder [36] in order to allow easy incorporation of complex long-range language models, like high-order N-gram models or topic models.

The performance of the decoder was experimented on Finnish speech data with context-sensitive phoneme-models using Gaussian mixtures and a 3-gram language model. The front-end of the system used mel-frequency cepstrum coefficients.

The effects of the lexicon size and pruning levels were studied on segmented words and continuous speech. As expected, the lexicon size seemed to have a large effect on the recognition results if the language model was not used, but with the language model the lexicon size could be increased without affecting the results considerably. Also, the results seemed to show that using language model helps the recognition considerably, while the choice of the exact language model scaling factor was not so crucial.

It was perhaps a bit surprising that the hypothesis pruning level did not seem to affect the error rates, while the acoustic pruning level had a much larger effect. Even discarding all but the best hypothesis from each stack did not seem to degrade the results noticeably. One reason for this behavior is perhaps that the language model used in the experiments was not so good. It was useful compared to the case without any language model, but it could not utilise the word history even if it was given a chance. Even if the N-gram models have been successful in modeling English, it has been proposed that they are not so suitable for Finnish [33]. Also, it may be that the language of the book used as test data was quite different from the language used to create the 3-gram model,

As the N-gram language models did not seem to give very promising results on Finnish data, and the implemented decoder allows arbitrary language models, it would be interesting to implement also other language models and see how well they would perform—alone or combined with N-gram models. Also, the current N-gram model implementation could be optimised to use memory more efficiently as described in [35].

It would also be interesting to study the word candidate lists in more detail to see how much the log-probabilities of the best words differ compared to the correct word. This information might be used to select the pruning parameters and the language model scaling factor automatically.

Bibliography

[1] J. Allen, D. Byron, M. Dzikovska, G. Ferguson, and L. Galescu. Towards conversational human-computer interaction. *AI Magazine*, 2001.

[2] X. L. Aubert. An overview of decoding techniques for large vocabulary continuous speech recognition. *Computer Speech and Language*, 16(1):88–114, Jan. 2002.

[3] L. E. Baum. An inequality and associated maximization technique in statistical estimation for probailistic functions of Markov processes. *Inequalities*, 3:1–8, 1972.

[4] L. E. Baum and T. Petrie. Statistical inference for probabilistic functions of finite state Markov chains. *Annals of Mathematical Statistics*, 37:1559–1563, 1966.

[5] J. R. Bellegarda. Exploiting latent semantic information in statistical language modeling. *Proceedings of the IEEE*, 88(8):1279–1296, 2000.

[6] W. Byrne, J. Hacič, P. Ircing, F. Jelinek, S. Khudanpur, P. Krbec, and J. Psutka. On large vocabulary continuous speech recognition of highly inflectional language — Czech. In *Proceedings of Eurospeech*, pages 487–489, Aalborg, Denmark, 2001.

[7] S. F. Chen and J. Goodman. An empirical study of smoothing techniques for language modeling. Technical Report TR-10-98, Computer Science Group, Harvard University, Aug. 1998.

[8] P. Clarkson and R. Rosenfeld. Statistical language modeling using the CMU-Cambridge toolkit. In *Proceedings of Eurospeech*, pages 2707–2710, Rhodes, Greece, 1997.

[9] J. S. Garofolo, C. G. P. Auzanne, and E. M. Voorhees. The TREC spoken document retrieval track: A success story. In *Proceedings of the Recherche d'Informations Assiste par Ordinateur: ContentBased Multimedia Information Access Conference*, pages 1–20, 2000.

[10] J. Gauvain, J. Gangolf, and L. Lamel. Speech recognition for an information kiosk. In *International Conference on Spoken Language Processing (ICSLP)*, volume 2, pages 849–852, Philadelphia, PA, 1996.

[11] D. Gildea and T. Hofmann. Topic-based language models using EM. In *European Conference on Speech Communication and Technology*, Budapest, Hungary, 1999.

[12] J. Glass, T. Hazen, and I. Hetherington. Realtime telephone-based speech recognition in the JUPITER domain. In *IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, pages 61–64, Phoenix, 1999.

[13] H. Hermansky. Perceptual linear predictive (PLP) analysis for speech. *The Journal of The Acoustical Society of America*, 87:1738–1752, Apr. 1990.

[14] H. Hermansky. Mel cepstrum, deltas,double-deltas,.. — what else is new? In *Proceedings of Robust Methods for Speech Recognition in Adverse Conditions*, Tampere, Finland, 1999.

[15] H. Hermansky, N. Morgan, A. Bayya, and P. Kohn. RASTA-PLP speech analysis. Technical Report TR-91-069, International Computer Science Institute, Berkeley, CA, Dec. 1991.

[16] M. Jalanko. *Studies of Learning Projective Methods in Automatic Speech Recognition.* PhD thesis, Helsinki University of Technology, 1980.

[17] S. E. Johnson, P. Jourlin, K. Spärck Jones, and P. C. Woodland. Spoken Document Retrieval for TREC-9 at Cambridge University. In *The Ninth Text REtrieval Conference (TREC-9).*, Gaithersburg, Maryland, 2001.

[18] B. H. Juang and L. R. Rabiner. The segmental k-means algorithm for estimating parameters of hidden Markov models. *IEEE Transactions on Acoustics, Speech and Signal Processing*, 38(9):1639–1641, Sept. 1990.

[19] M. Kurimo. *Application of Learning Vector Quantization and Self-Organizing Maps for training continuous density and semi-continuous Markov models.* Licentiate's thesis, Helsinki University of Technology, 1994.

[20] M. Kurimo. *Using Self-Organizing Maps and Learning Vector Quantization for Mixture Density Hidden Markov Models.* PhD thesis, Helsinki University of Technology, 1997.

[21] M. Kurimo and K. Lagus. An efficiently focusing large vocabulary language model accepted for publication. In *Proceedings of the International Conference on Artificial Neural Networks (ICANN'02)*, pages 1068–73, Madrid, Spain, 2002.

[22] K. Lagus and M. Kurimo. Language model adaptation in speech recognition using document maps. In *IEEE Workshop on Neural Networks for Signal Processing*, pages 627–636, Martigny, Switzerland, 2002.

[23] C. D. Manning and H. Schütze. *An Empirical Study of Smoothing Techniques for Language Modeling*, chapter 6. The MIT Press, 1998.

[24] J. S. Milton and J. C. Arnold. *Introduction to probability and statistics: principles and applications for engineering and the computer sciences.* McGraw-Hill, 3rd edition, 1995.

[25] M. Mohri, F. Pereira, and M. Riley. Weighted finite-state transducers in speech recognition. *Computer Speech and Language*, 16(1):69–88, Jan. 2002.

[26] J. J. Odell. *The Use of Context in Large Vocabulary Speech Recognition.* PhD thesis, University of Cambridge, 1995.

[27] S. Ortmanns, H. Ney, A. Eiden, and N. Coenen. Look-ahead techniques for improved beam search. In *Proceedings of the CRIM-FORWISS Workshop*, pages 10–22, 1996.

[28] J. Picone. Signal modeling techniques in speech recognition. *Proceedings of the IEEE*, 81(9):1215–1247, 1993.

[29] L. R. Rabiner. A tutorial on hidden Markov models and selected applications in speech recognition. *Proceedings of the IEEE*, 77(2):257–286, 1989.

[30] M. K. Ravishankar. *Efficient Algorithms for Speech Recognition.* PhD thesis, Carnegie Mellon University, 1996.

[31] S. Renals and M. Hochberg. Decoder technology for connectionist large vocabulary speech recognition. Technical Report CUED/F-INFENG/TR.186, Engineering Department, Cambridge University, 1994.

[32] T. D. Rossing. *The science of sound.* Addsion Wesley, 2nd edition, 1990.

[33] V. Siivola, M. Kurimo, and K. Lagus. Large vocabulary statistical language modeling for continuous speech recognition in finnish. In *Proceedings of Eurospeech*, pages 737–740, Aalborg, Denmark, 2001.

[34] K. Torkkola, J. Kangas, P. Utela, S. Kaski, M. Kokkonen, M. Kurimo, and T. Kohonen. Status report of the Finnish phonetic typewriter project. In T. Kohonen, K. Mäkisara, O. Simula, and J. Kangas, editors, *Artificial Neural Networks*, volume I, pages 771–776, Amsterdam, Netherlands, 1991. North-Holland.

[35] E. W. D. Whittaker and B. Raj. Quantization-based language model compression. In *Proceedings of Eurospeech*, pages 33–38, Aalborg, Denmark, 2001.

[36] D. Willett, C. Neukirchen, and G. Rigoll. Ducoder - the Duisburg University LVSCR stackdecoder. In *IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, Istanbul, Turkey, 2000.

[37] M. Woszczyna. *Fast Speaker Independent Large Vocabulary Continuous Speech Recognition.* PhD thesis, Universität Karlsruhe, 1998.

[38] S. Young. Large vocabulary continuous speech recognition. *IEEE Signal Processing Magazine*, 13(5):45–57, 1996.

[39] S. Young, G. Evermann, D. Kershaw, G. Moore, J. Odell, D. Ollason, V. Valtchev, and P. Woodland. The HTK book (for HTK version 3.1). Technical report, Cambridge University Engineering Department, 2001.

[40] S. J. Young, N. H. Russell, and J. H. S. Thornton. Token passing: a simple conceptual model for connected speech recognition systems. Technical Report CUED/F-INFENG/TR38, Cambridge University, 1989.