TEKNILLINEN KORKEAKOULU
Sähkötekniikan osasto

Panu Somervuo

# Speech recognition using context vectors and multiple feature streams

Diplomityö, joka on jätetty opinnäytteenä tarkastettavaksi diplomi-insinöörin tutkintoa varten Espoossa 16.2.1996.

Työn valvoja: Professori Teuvo Kohonen

Työn ohjaaja: Tekniikan lisensiaatti Mikko Kurimo

**TEKNILLINEN KORKEAKOULU** **DIPLOMITYÖN TIIVISTELMÄ**

| | |
|---|---|
| **Tekijä:** | Panu Somervuo |
| **Työn nimi:** | Speech recognition using context vectors and multiple feature streams (Kontekstivektorien ja monen piirrevirran käyttö puheentunnistuksessa) |
| **Päivämäärä:** 16.2.1996 | **Sivumäärä:** 67 |
| **Osasto:** | Sähkötekniikan osasto |
| **Professuuri:** | Tik-61 Informaatiotieteet |
| **Työn valvoja:** | Professori Teuvo Kohonen |
| **Työn ohjaaja:** | Tekniikan lisensiaatti Mikko Kurimo |

Työssä on tutkittu kontekstivektorien ja monen piirrevirran käyttöä puheentunnistuksessa. Piirrevalinta on tehty oppivaa vektorikvantisaatiota käyttäen ja itse puheentunnistuksessa on käytetty kätkettyjä Markovin malleja.

Kontekstivektori on yhdistelmä useasta peräkkäisestä lyhyen ajan piirrevektorista. Yhdistäminen voi tapahtua keskiarvostamalla, konkatenoimalla tai laskemalla delta-piirteitä. Konkatenointi ja delta-piirteet tuovat informaatiota puheen ajallisesta rakenteesta. Koska ajallisesti peräkkäisten piirrevektorien erotuksen laskeminen liittyy kyseisen piirrevirran autokorrelaatioon, paras tapa käyttää delta-piirteitä ei ole piirrevirran matemaattisen derivaatan mahdollisimman hyvä approksimointi, vaan sellaisten parametrien käyttö, joilla voidaan mitata peräkkäisten piirrevektorien samankaltaisuutta tai eroavuutta mahdollisimman hyvin. Työssä on tehty useita kokeita, joissa on pyritty löytämään sellaisia piirrevektoreita, joiden avulla äänteiden keskiosat voidaan erottaa äänteiden transitiokohdista.

Kontekstivektoreita käyttämällä voidaan koartikulaatio, eli peräkkäisten äänteiden vaikutus toisiinsa, ottaa huomioon jo piirrevektorissa. Tämä on avuksi päällekkäin menevien piirrevektorien luokittelussa. Vaikka kätkettyjen Markovin mallien tarkoituksena on liittää peräkkäiset havaintovektorit yhteen, sen perusversio on ensimmäisen asteen malli, joten kontekstivektorit tuovat luokitteluun lisää informaatiota.

Eri piirreirrotusmenetelmiä voidaan pitää erillisinä informaatiolähteinä, joista kukin muodostaa oman piirrevirtansa. Piirrevirta koostuu ajallisesti peräkkäisistä piirrevektoreista. Eräs mahdollisuus syöttää usea samanaikainen piirrevektori kätkettyyn Markovin malliin on konkatenoida ne yhdeksi havaintovektoriksi, jolloin kätketylle Markovin mallille annetaan vain yksi syötevirta. Vaihtoehtoisesti eri piirrevektorit voidaan pitää erillään, jolloin kätketyllä Markovin mallilla on monta rinnakkaista sisäänmenoa. Työssä tutkittiin kumpaakin vaihtoehtoa. Yksi rinnakkaisten syötevirtojen etu on, että erityyppisten piirteiden koodikirjat voidaan pitää toisistaan erillään ja välttää näin piirteiden konkatenoinnissa syntyvä skaalausongelma.

Avainsanat: puheentunnistus, oppiva vektorikvantisaatio, itseorganisoiva kartta, kätketty Markovin malli.

**HELSINKI UNIVERSITY OF TECHNOLOGY** **ABSTRACT OF THE MASTER'S THESIS**

**Author:** Panu Somervuo

**Name of the thesis:** Speech recognition using context vectors and multiple feature streams

**Date:** 16.2.1996 **Number of pages:** 67

**Faculty:** Faculty of Electrical Engineering
**Professorship:** Tik-61 Computer and Information Science

**Supervisor:** Professor Teuvo Kohonen
**Instructor:** Lic.Tech. Mikko Kurimo

The study investigates the use of context vectors and multiple feature streams in the speech recognition. Learning Vector Quantization is used in feature selection and semi-continuous hidden Markov models are used in the final recognition task.

Context vectors are combinations of several successive short time features. The experimented combination methods are averaging, concatenation and delta calculations. Both the concatenation and delta features bring information about the temporal structure of the speech. It can be simply shown that the calculation of differences of consequtive feature vectors is related to the autocorrelation of the feature stream. Therefore if delta features are used, the best way is not to approximate the mathematical time derivate, but rather to use parameters which measure the similarity or dissimilarity of the consequtive features. Several experiments have been done in order to find discriminative features, which are able to separate phone centers from phone transitions.

The main reason for using context vectors is to take the coarticulation effect into account. This helps in classifying overlapping phone classes. This is beneficial even when HMM is used as a classifier. Although the idea of using HMM is to link consequtive short-time observations together, the basic version of it is the first order model and context vectors bring thus more information available.

Different feature extraction methods can be considered separate information sources, each of which producing an observation stream. One possibility to feed these observation streams into HMM is to concatenate all feature vectors from one time event to a single observation vector. HMM has then a single input stream. Another possibility is to keep different feature vector types in separate streams. In that case HMM has multiple parallel inputs. One of the main interests in this study was to investigate these two alternatives. One advantage of having multiple input streams is that scaling problems can be avoided because vectors with totally different numerical scales can be kept in separate codebooks.

Keywords: speech recognition, learning vector quantization, self-organizing map, hidden Markov model.

# Preface

This study was carried out in the Neural Networks Research Centre at the Helsinki University of Technology.

I wish to thank my supervisor professor Teuvo Kohonen for the opportunity to work in the laboratory and to do my Master's Thesis on this topic. My instructor Lic.Tech. Mikko Kurimo I want to thank for the guidance to the world of hidden Markov models. Special thanks also to other personnel of the Neural Networks Research Centre and the Laboratory of Computer and Information Science for sharing daily conversations and helping in all kind of problems.

Otaniemi, February 16, 1996

Panu Somervuo

# Contents

# List of notations

The main symbols and abbreviations used in this work are described below. If some symbols are used for other purposes, the meaning is explained in the context.

| | |
|---|---|
| $\alpha(t)$ | learning rate |
| $\lambda$ | parameter set of HMM |
| $\boldsymbol{\mu}$ | mean vector |
| $\boldsymbol{\pi}$ | initial state probability vector of HMM |
| $\Sigma$ | covariance matrix |
| $\tau$ | time delay |
| $A$ | HMM state transition probability matrix |
| $a_{ij}$ | HMM state transition probability from state $i$ to state $j$ |
| $a_k$ | LPC coefficient |
| $\arg(\cdot)$ | argument of $(\cdot)$ |
| $B$ | set of $b_i(\mathbf{O}_t)$ |
| $b_i(\mathbf{O}_t)$ | observation probability of state $i$ |
| F0 | fundamental frequency of speech |
| $f_k(\mathbf{O}_t)$ | density function of the $k$th mixture component |
| $\mathbf{m}_i$ | map unit $i$ |
| $\max(\cdot)$ | maximum of $(\cdot)$ |
| $N_c(t)$ | neighborhood of unit $c$ |
| $\mathbf{O}$ | observation vector sequence |
| $\mathbf{O}_t$ | observation vector at time instant $t$ |
| $o_{td}$ | $d$th component of the vector $\mathbf{O}_t$ |
| $P(A\|B)$ | conditional probability |
| $\hat{p}$ | estimate of $p$ |
| $\mathbf{q}$ | HMM state sequence |
| $h$ | shape factor, width of the kernel |
| $t$ | time |
| $\mathbf{x}(t)$, $\mathbf{x}$ | feature vector |
| $z$ | complex variable |

| | |
|---|---|
| AR | autoregressive |
| ARMA | autoregressive moving average |
| BMU | best-matching unit |
| CDHMM | continuous density HMM |
| DEC | Dynamically Expanding Context |
| DHMM | discrete HMM |
| EC | error corrective |
| FFT | fast Fourier transform |
| HMM | hidden Markov model |
| KNN | K nearest neighbors |
| LPC | linear predictive coding |
| LVQ | Learning Vector Quantization |
| ML | maximum likelihood |
| OLVQ | optimized-learning-rate LVQ |
| PCA | principal component analysis |
| pdf | probability density function |
| rms | root mean square |
| SCHMM | semi-continuous HMM |
| SOM | Self-Organizing Map |
| TS-SOM | tree-structured SOM |
| VQ | vector quantization |

# Chapter 1

# Introduction

Speech recognition can be defined as a mapping from continuous acoustic signal to a discrete set of symbols. The conversion from an acoustic waveform to a written equivalent of the message information is of the main interest in this study.

Being a fascinated research topic also by itself, speech recognition has many potential applications. The general goal is to understand normal human speech and then to be able to perform some task based on this understanding. This is a very natural goal in that it requires machines to adapt to humans rather than vice versa.

The interaction between a speaker and a listener is described in figure 1.1. First an idea which has arosen in the speaker's mind is put on words. These words are then converted to acoustic signals in the speaker's articulatory system. Acoustic signals consist of pressure changes, which propagate in the air. When these pressure waves reach the listener's ear, they activate first the auditory nerves. After that the activated parts of the brain cause the listener to associate the perception to some image or idea in his/her mind. The information content has thus been extracted from the air pressure waves. In the artificial speech recognition system, the listener is replaced by a machine.

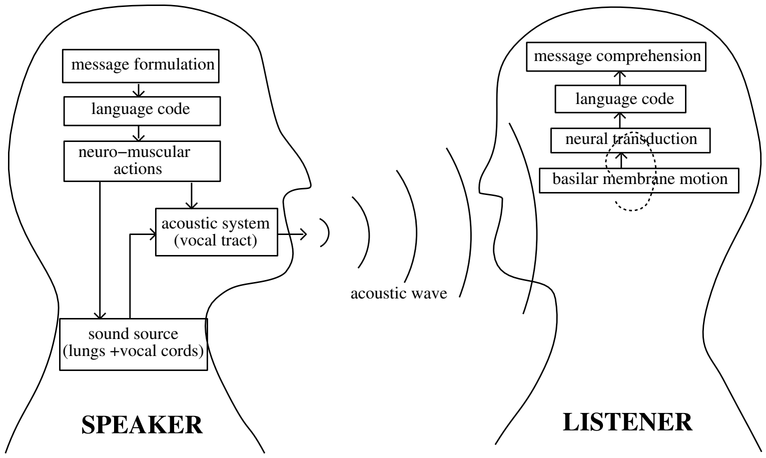


Figure 1.1: *Communication by speech.*

Both speech synthesis and speech recognition have been tried to do by machines, but the flexibility of a human is hard to achieve. In spite of intensive studying, at least until now humans have been much more capable in understanding synthesized speech than machines have been able to recognize human speech. This is mostly due to the wide experience of humans. We can easily tolerate erroneous speech and replace missing parts of it by using our knowledge about the language, the speaker, the context of the speech and the meaning of the words.

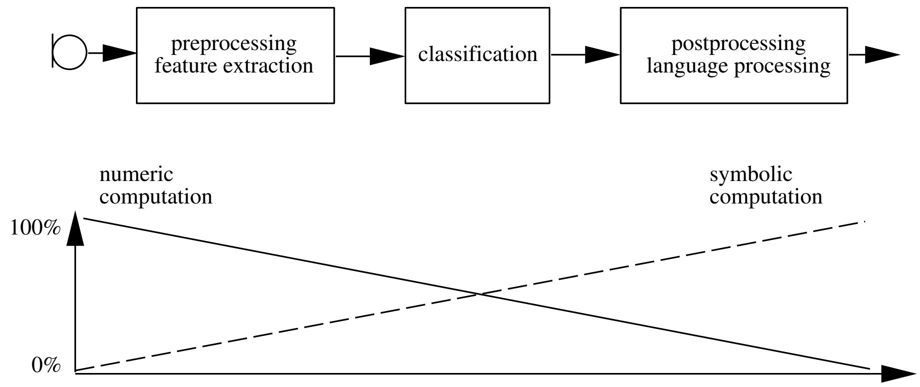


Figure 1.2: *Block diagram of the speech recognition system. The borders of the blocks need not be fixed and probably a great amount of feedback between the blocks is required in order to make a flexible and well-performing system.*

Figure 1.2 represents a block diagram of the speech recognition system. The two first blocks (labeled as preprocessing and classification) perform the signal to symbol transformation. The ensuing block (postprocessing) converts the phonetic writing then into the orthography. The lower plot in the figure represents the amount of numeric and symbolic computation in the different stages of the recognition process. It is evident that first stages of this kind of pattern recognition system involve more signal to symbol conversions, but in the later stages of the recognition, the need of symbol processing becomes more dominant.

In the speech recognition system, some limitations are usually done concerning the nature of the speech to be recognized. These limitations may include the number of the speakers, the size of the vocabulary and the amount of the noise in the speech. Because of the vastness of this research topic, all aspects of speech recognition are not covered in this study. The main emphasis is on the feature extraction and signal to symbol transformation. A good overall description of the development of the speech recognition system can be found in (Lee, 1988).

## 1.1 Some aspects to the speech recognition

There are several reasons for the difficulties in the speech recognition. Natural speech has variations in many levels. In addition to that different speakers have different voices, there is also considerable variation in the voice of one speaker. In the normal conversation, one speaker may stress some parts of the words. The loudness and the pitch of the voice change and also the speaking rate varies. Even if the speaker tries to use as steady voice as possible, no two uttered sounds are generally equal. Speech is typically nonstationary signal and the transients may carry more information than the stationary parts.

Some similarities between the handwritten language and the spoken language are presented next in order to grasp the ideas of the speech recognition. Both forms consist of certain building blocks. Written language consists of written characters and spoken language consists of consequtive speech segments.

When reading handwritten characters, various drawings in the paper can be recognized and associated to the certain abstract classes. The physical character which can be seen is called a *graph*. The abstract class, where a graph belongs to, is called a *grapheme*. One grapheme may have many possible realizations, which all vary, but still represent the same class. Similar variants of one grapheme are called *allographs*.

In the spoken language the smallest meaningful unit is a *phoneme*. It doesn't have any meaning alone, but it makes possible to discriminate different words. The following Finnish word pairs give an example of this. The meaning of the word changes, when one phoneme is changed. However, it has to be noted that a phoneme does not mean the same as a letter, although in Finnish language the correspondence is very close.

| | | |
|---|---|---|
| $\boxed{\text{T}}\text{ALO}$ | - | $\boxed{\text{P}}\text{ALO}$ |
| $\text{SA}\boxed{\text{N}}\text{A}$ | - | $\text{SA}\boxed{\text{M}}\text{A}$ |
| $\text{KAN}\boxed{\text{A}}$ | - | $\text{KAN}\boxed{\text{I}}$ |

The physical sound produced when a phoneme is uttered is called a *phone*. One phone is thus a realization of a phoneme. There exist many ways to pronounce one phoneme and a group of phones which contain similar variants of one single phoneme is called an *allophone*.

The speech recognition could now be defined as an attempt to transform phones back to the phonemes. This would give a phonetic (or phonemic) representation of the speech. Other units of speech may also be chosen as building blocks of the speech recognition like diphones (phoneme pairs), triphones (three consequtive phonemes), syllables, or whole words. The higher the level of the recognition unit is, the more accurate can the recognition result be expected to be. But at the same time the database of the recognition system has to be larger, because each upper level has to contain all legal combinations of the units from the lower level.

It is clear that the size of the vocabulary has to be fixed, if the speech unit to be recognized is a whole word. But if smaller units are chosen to be the building blocks of the recognition, the vocabulary of the system can be kept unlimited.

Here phonemes have been chosen to be the building blocks of the speech recognition. It could be thought that as a consequence no specific language needs then to be fixed. Of course there exist different phonemes in different languages, but adding them would just bring new phoneme classes to the existing system and the basic idea and method of signal to symbol transformation may remain unaltered. However, the most challenging problem may be to develop well-performing conversion rules from the phonetic writing to the orthography of the words. Different languages require naturally different conversion rules.

An important issue is what happens, when several building blocks of the language are concatenated in order to form words. In the case of written language and if calligraphy is used, the borders of consequtive characters may blend together. In the speech, the counterpart of this blending is called coarticulation.

Coarticulation effect is a consequence of our physical articulatory system. Because of the inertia in the articulators, previous phones have impact on the ensuing ones. The causality may occur also reversed in time. Different phonemes have varied requirements for the articulators and when articulator gestures for one sound does not conflict with those of an ensuing phoneme, the articulator may move toward or adopt a state of an ensuing phoneme. The articulation of a phoneme depends thus on its phonemic context.

## 1.2 Basics for this study

This study can be thought to be continuation to the research started in the early 80's in the Laboratory of Computer and Information Science (Kohonen, 1988; Torkkola et al., 1991). The speech recognition system studied here is a speaker-dependent, isolated words system. The vocabulary of the system is unlimited. This is possible because the speech unit to be recognized is a phone. The accuracy of the speech recognition is determined by the similarity between the recognized word and its correct spelling.

The new speech database was collected by 15 male speakers. Each of them uttered four times 350 Finnish words. Three speech sets were used for training of the system and one speech set was used for testing it. A speech recognition system based on hidden Markov models was trained to each of the 15 speakers and according to those recognition results the speeches of two speakers were chosen as development material to be used in this study. One of the chosen speakers obtained the lowest error rate and another speaker represents the average speaker. The goal of the feature and parameter selection was not to obtain the best recognition results for these two speakers but rather to investigate the effect of adjustable parameters and try to find parameters which could be used for other speakers as well.

### 1.2.1 Language of the speech corpus

Finnish is a phonetic language, which means that there is a rather simple correspondence between the phonemic appearance and the correct spelling of the words. Therefore it suits well the experiments concentrating on the signal to symbol transformation and almost omitting the use of symbol manipulation (the postprocessing part of the speech recognition system).

Finnish alphabet has 29 letters, but the words used in experiments consisted of 20 alphabets (A, D, E, H, I, J, K, L, M, N, O, P, R, S, T, U, V, Y, Ä, Ö). Nine rare alphabets of foreign origin (B, C, F, G, Q, W, X, Z, Å) were left out. B, F and G are pronounciated distinguishingly form other letters, but C is pronounced like /S/[1] or /K/ depending on the word. Q is pronounced like /K/, W sometimes like /V/, X like /K,S/, Z like /T,S/ and Å like /O/. So including these letters to the word list would complicate the conversion from phoneme string to the correct spelling of the word.

Other exceptions to the simple correspondence between phonemic representation and the correct spelling of the word is the combination of letters N and G or the combination of N and K. If N and G are successive letters, they are pronounced like /$\eta$/ and if N and K are successive letters, the pronunciation is like /$\eta$,K/.

Difficulties may also occur in the case of double phonemes. The duration of a phone contains partly the information, whether there are one or two letters in the orthography. But the division between a single and a double phoneme is not totally trivial. The duration depends on the context of a phoneme.

It seems quite obvious that in order to achieve an errorless orthography, some knowledge about possible words must be available in the system even in the case of such a phonetic language like Finnish. DEC (Kohonen, 1986a; Kohonen, 1995) is a way to improve the orthography of the speech recognition, but it was not used in this study.

[1] A letter between // denotes a phoneme.

# Chapter 2

# Tools

In the following sections the main tools used in this study are introduced. The speech recognition system is based on hidden Markov models (HMMs). Concept of HMM is introduced in section 2.3. Here HMMs use Self-Organizing Maps (SOM) as probability density approximations. SOM algorithm is described in section 2.1. Learning Vector Quantization was used in finding the discriminative features. LVQ is described in section 2.2.

## 2.1 Self-Organizing Map

### 2.1.1 Vector quantization

Vector quantization is a method to represent vector spaces by a finite number of vectors. The set of chosen reference or prototype vectors is called a codebook. Figure 2.1 shows an example of a two-dimensional codebook.

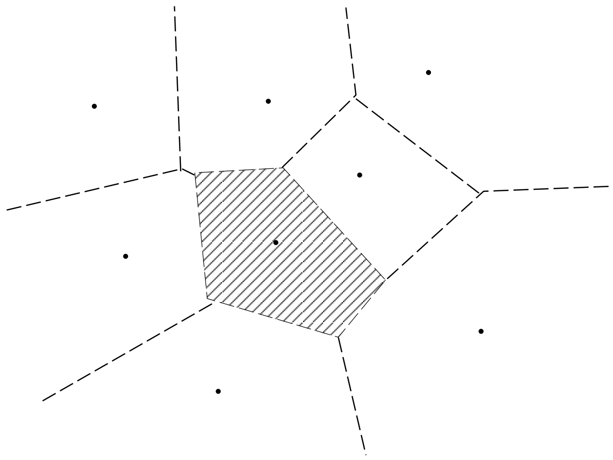

Figure 2.1: *Example of a two-dimensional codebook. Points are the reference vectors and dashed lines indicate the decision borders according to the nearest neighbor classification. The decision borders divide the input space into Voronoi regions.*

### 2.1.2 Basic SOM algorithm

SOM algorithm (Kohonen, 1981; Kohonen, 1995) is a method to form an ordered codebook. The word 'ordered' means that similar samples in the input space are mapped to SOM units near each other. This topology preserving property distinguishes SOM from other VQ-methods.

SOM can be used to map feature vectors from high dimensional source space into lower dimensional destination space. Usually the destination space has the dimension of two. Two-dimensional maps have seemed to be powerful enough to represent high dimensional spaces and also the maps can then be visualized easily in a two-dimensional display.

The basic iterative learning algorithm of SOM is described as follows:

Let $\mathbf{x}$ be an input vector. When it is fed to the SOM, the best-matching unit $\mathbf{m}_c$ (the nearest codebook vector to $\mathbf{x}$) and its neighbors are updated closer to $\mathbf{x}$.

$$\mathbf{m}_i(t+1) = \begin{cases} \mathbf{m}_i(t) + \alpha(t)[\mathbf{x}(t) - \mathbf{m}_i(t)] & , i \in N_c(t) \\ \mathbf{m}_i(t) & , i \notin N_c(t) \end{cases} \tag{2.1}$$

$N_c$ is the neighborhood of the winner and $\alpha(t)$ is the learning rate.

The teaching is continued several rounds, picking the input vector $\mathbf{x}(t)$ at each round $t$ randomly or cyclically from the training samples. The neighborhood $N_c(t)$ around the best-matching unit is usually wide in the beginning of the training and shrinks monotonically with time. The use of this neighborhood gives the topology preserving property to the codebook. Learning rate $\alpha(t)$ decreases also during the training.

### 2.1.3 SOM as a probability density function approximation

The basic property of SOM algorithm is that the density of the reference vectors approaches the pdf of the input vectors during the training.

SOM units and their Voronoi regions can be considered as Parzen kernels. Parzen estimation (Parzen, 1962) is a standard nonparametric density estimation method. The estimator for one kernel is formulated as

$$\hat{p}(\mathbf{x}) = \frac{1}{N} \sum_{j=1}^{N} \gamma(\|\mathbf{x}_j - \mathbf{m}\|) \ , \tag{2.2}$$

where $\gamma$ is the Parzen kernel function and $\mathbf{m}$ is its center point. $\{\mathbf{x}_j\}_{j=1}^{N}$ is the set of input samples. The most common kernel types are rectangular, triangular, exponential and Gaussian. To obtain valid pdf estimates the kernel must be positive and the integral over all kernels of the codebook must be normalized to one.

There are several advantages of using SOM as a pdf approximation. Because of the neighborhood, all codebook vectors can be taken already from the beginning of the pdf estimation into use and the whole codebook is effectively utilized. The neighborhood provides also a smooth density distribution of the codebook vectors and helps thus the training of the continuous (semi-continuous) HMM mixture densities.

### 2.1.4 Tree-Structured SOM

Search of the best-matching unit is computationally the heaviest operation in SOM. But by teaching the map hierarchically, it is possible to use tree search for this. Tree-Structured SOM was not used in the experiments of this study, but it is described here, because it can bring considerable speedup to the classification. Tree-Structured SOM (TS-SOM) consists of several layers as depicted in figure 2.2.

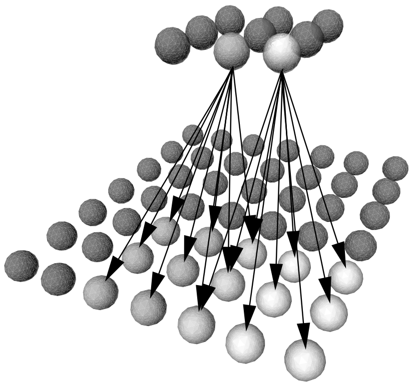

Figure 2.2: *Two layers of the hierarchical SOM. Each SOM unit has child units in the lower layer. In this map, each upper layer unit has* $3 \times 3$ *child units.*

The speedup is achieved due to the limited search area in each layer of the map. Let the size of the topmost map be $size_0$ units, the number of layers be $L$ and the search region in each map layer (except the topmost map) be $S$ units. By using the tree search, $size_0 + (L-1) \cdot S$ distance calculations have to be made in BMU search. But by using the normal search, the number of distance calculations is the number of units in the bottom layer. The savings can be considerable, if large maps are used and the size of the search region $S$ can be kept small.

Teaching of the TS-SOM is done layer by layer. The idea is to proceed from the topmost map to the bottom layer by teaching each layer totally before going to the next layer. The already taught layer is thus considered to be 'frozen'. Finding the BMU is done by using the tree search. For that purpose a search region $S$ has to be defined. For updating, another region denoted as $U$ is defined. The size of the search region determines the probability of finding the real BMU of the map. Because of the limited search area, this is by no means guaranteed. Especially when the input space is high dimensional, this may be a problem. To improve the finding of the real BMU, the updating region $U$ should be smaller than the search region $S$.

In (Koikkalainen and Oja, 1990; Koikkalainen, 1994), TS-SOM was introduced as a recursive algorithm. However, in this representation the recursion has been removed. The active layer, i.e. the layer which is currently being taught, is denoted as $al$. The best-matching unit of the layer $l$ is denoted as $BMU_l$ and its search region is denoted as $S(BMU_{l-1})$. This is because it consists of the child units of $BMU_{l-1}$.

Teaching algorithm of TS-SOM

1. teach the topmost map (layer 0) in the normal way.
2. for each map layer $al = 1 \ldots L-1$,
   - initialize units of layer $al$:
     - copy each parent unit to the middle of its child unit region
     - interpolate other child unit vectors according to the nearest neighbors of the parent unit
   - for $t = 0 \ldots$ number of teaching rounds in $al$,
     - get the next data entry
     - find $BMU_{al}$ by using tree search:
       * find BMU in topmost map ($BMU_0$)
       * for $l = 1 \ldots al$,
         · find $BMU_l \in S(BMU_{l-1})$
     - adapt $BMU_{al}$ and its neighbors $\in U(BMU_{al})$

The main difference to the basic SOM is that the neighbourhood of a unit remains fixed. But the learning rate $\alpha(t)$ is still a function of time. Several modifications to the algorithm are of course possible and there are no rules that the neighborhood may not also be a function of time.

In the figure 2.2, the search region is 3×3 child units. One side of the region is common to two parent units. If the size of the layer $l$ is $side_l \times side_l$ units, the size of the next layer is then $(2 \cdot side_l + 1) \times (2 \cdot side_l + 1)$ units. However, as it was mentioned, the choice of the neighborhood and search region can be done in many other ways too.

## 2.2 Learning Vector Quantization

While SOM has the property to adapt to the density of the input vectors at the same time preserving the topology of the input space, it is not necessarily the best method to discriminate samples of different classes. For that purpose another algorithm, called Learning Vector Quantization, has been developed (Kohonen, 1986b; Kohonen, 1995). While the basic SOM algorithm is based on the unsupervised learning, LVQ is based on the supervised learning. Actually LVQ is not just one algorithm, but there exists a set of its variants. LVQ was used in this study for finding discriminative feature vectors, as described in section 4.2.

In the LVQ method, vector quantization is not used to approximate the density functions of the feature classes, but to directly define the class borders according to the nearest-neighbor rule. Here the main algorithms are described briefly. Input samples are picked from the training set one by one.

In LVQ1, the learning rule is

$$\mathbf{m}_c(t+1) = \mathbf{m}_c(t) + \alpha(t)[\mathbf{x}(t) - \mathbf{m}_c(t)] \;, \tag{2.3}$$

if $\mathbf{x}$ and $\mathbf{m}_c$ belong to the same class, and

$$\mathbf{m}_c(t+1) = \mathbf{m}_c(t) - \alpha(t)[\mathbf{x}(t) - \mathbf{m}_c(t)] \ , \tag{2.4}$$

if $\mathbf{x}$ and $\mathbf{m}_c$ belong to different classes. $\mathbf{m}_c$ is the nearest codebook vector to the given input vector. No other codebook vectors are updated.

LVQ2.1 defines a 'window', which means a zone around the midplane of the two nearest neighbors to $\mathbf{x}$. These units are denoted as $\mathbf{m}_i$ and $\mathbf{m}_j$. If $\mathbf{m}_i$ belongs to the same class as $\mathbf{x}$, but $\mathbf{m}_j$ belongs to the different class and $\mathbf{x}$ falls into the 'window', the learning rule is

$$\begin{aligned} \mathbf{m}_i(t+1) &= \mathbf{m}_i(t) + \alpha(t)[\mathbf{x}(t) - \mathbf{m}_i(t)] \\ \mathbf{m}_j(t+1) &= \mathbf{m}_j(t) - \alpha(t)[\mathbf{x}(t) - \mathbf{m}_j(t)] \ . \end{aligned} \tag{2.5}$$

LVQ3 has an addition to LVQ2.1. If $\mathbf{x}$, $\mathbf{m}_i$ and $\mathbf{m}_j$ all belong to the same class, the updating rule is

$$\begin{aligned} \mathbf{m}_i(t+1) &= \mathbf{m}_i(t) + \epsilon\alpha(t)[\mathbf{x}(t) - \mathbf{m}_i(t)] \\ \mathbf{m}_j(t+1) &= \mathbf{m}_j(t) + \epsilon\alpha(t)[\mathbf{x}(t) - \mathbf{m}_j(t)] \ . \end{aligned} \tag{2.6}$$

Finally, the optimized-learning-rate LVQ1 (OLVQ1) is a modification of the LVQ1. It differs from LVQ1 so that each codebook vector $\mathbf{m}_i$ has an individual learning rate $\alpha_i(t)$.

### 2.2.1 Tree-Structured LVQ

Following the ideas of the TS-SOM, some attempts towards the tree-structured LVQ could also be made. It could be something like this: TS-SOM is taught layer by layer and each layer is finetuned by LVQ (possibly LVQ2) before proceeding to the next layer. Alternatively only one or two bottom layers can be finetuned.

LVQ and hierarchical search have been used already before in the concept of Hypermap (Kohonen, 1991; Kohonen, 1995), but the search procedure there is a bit different than in TS-SOM. The idea of Hypermap is to use the context of the feature vectors hierarchically.

## 2.3 Hidden Markov models

Hidden Markov model is a stochastic signal model. The use of HMM in speech recognition can be described as an attempt to catch the temporal structure of the speech and link consequtive short-time observations together. The strength of the HMM method lies in its mathematical framework (theory) and implementational structure (practice). A good overview of hidden Markov models in speech recognition can be found in (Rabiner, 1989; Juang and Rabiner, 1991), but the basics of the HMMs are represented also here.

A Markov process or chain is a system, which is in one of the $N$ states at any discrete time instant $t$. The state variable is denoted as $q_t$. Generally, a full description of the system would require knowledge about the whole history of the state sequence, but in a first-order Markov process the current state of the system depends only on the previous state. When the state transition probabilities are time independent, the process can be described by a state transition probability matrix $A = [a_{ij}]$, where

$$a_{ij} = P(q_t = j | q_{t-1} = i) \quad , \quad i, j = 1, \ldots, N \ . \tag{2.7}$$

The probabilities satisfy the following axiomatic constraints:

$$\sum_{j=1}^{N} a_{ij} = 1 \ , \tag{2.8}$$

$$a_{ij} \geq 0 \ . \tag{2.9}$$

The initial state of the system is denoted as $q_0$ and it is defined by an initial probability vector $\boldsymbol{\pi} = [\pi_1, \pi_2, \ldots, \pi_N]^T$. Then for any state sequence $\mathbf{q} = (q_0, q_1, \ldots, q_T)$, the probability of being generated by the Markov process is

$$P(\mathbf{q}|A, \boldsymbol{\pi}) = \pi_{q_0} a_{q_0 q_1} \cdots a_{q_{T-1} q_T} \ . \tag{2.10}$$

A Markov process is called 'hidden', if the state sequence $\mathbf{q}$ cannot be directly observed. Instead, there are only indirect observations of the states available. No one-to-one correspondence between an observation and a state necessarily exists, but each state has a probability of having produced the observation. Because of this stochastic nature of the outputs, the observations can be used only for computing the probabilities of different state sequences.

In the speech recognition task, the observations are the extracted feature vectors of the speech and each HMM state is associated to some speech unit. The goal is then to find the hidden state sequence from the observations. In the case of phonetic speech recognition, the decoded state sequence gives then the phoneme sequence of the speech. Figure 2.3 has an example of a phoneme model.

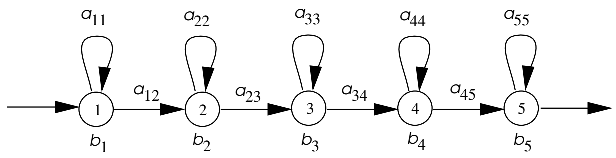


Figure 2.3: *A five-state phoneme model. This is called a left-to-right model, because the state index must increase or remain the same along the time in the model. Each state can be thought to model one part of the phoneme.*

When each phoneme has its own model, arbitrary phoneme chains can be formed, provided that there is a positive transition probability from the last state of each phoneme model to the first state of each phoneme model. This is depicted in figure 2.4.

The hidden Markov model is defined next more formally. An observation at time instant $t$ is denoted as $\mathbf{O}_t$. The probability of $\mathbf{O}_t$ being generated by the state $i$ is denoted as

$$b_i(\mathbf{O}_t) = P(\mathbf{O}_t | q_t = i) \ . \tag{2.11}$$

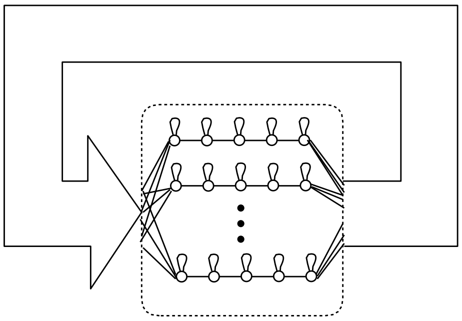

Figure 2.4: *The box contains all phoneme models. This system can be used for decoding a phone sequence of arbitrary length.*

The set of probabilities $b_i(\mathbf{O}_t)$ of all states is denoted as $B = \{b_i(\mathbf{O}_t)\}_{i=1}^{N}$. The observations produced by the states are assumed to be independent of each other and thus the probability of the observation sequence $\mathbf{O} = (\mathbf{O}_1, \mathbf{O}_2, \ldots, \mathbf{O}_T)$ being produced by the state sequence $\mathbf{q}$ can be calculated by the product

$$P(\mathbf{O}|\mathbf{q}, B) = b_{q_1}(\mathbf{O}_1) b_{q_2}(\mathbf{O}_2) \cdots b_{q_T}(\mathbf{O}_T) \ . \tag{2.12}$$

Now the joint probability of $\mathbf{O}$ and $\mathbf{q}$ being produced by the system is simply the product of (2.10) and (2.12)

$$P(\mathbf{O}, \mathbf{q}|\boldsymbol{\pi}, A, B) = \pi_{q_0} \prod_{t=1}^{T} a_{q_{t-1}q_t} b_{q_t}(\mathbf{O}_t) \ . \tag{2.13}$$

The triple $\lambda = (\boldsymbol{\pi}, A, B)$ defines a hidden Markov model.

### 2.3.1 Three subproblems for HMM

The development of HMM can be divided into three subproblems (Rabiner, 1989):

- The evaluation problem: Given a model and a sequence of observations, what is the probability that the model generated the observations?
- The decoding problem: Given a model and a sequence of observations, what is the most likely state sequence in the model that produced the observations?
- The learning problem (the estimation problem): Given a model and a set of observations, what should the model parameters be so that the model has a high probability of generating the observations?

### 2.3.2 The evaluation problem

In the evaluation problem, the task is to compute

$$P(\mathbf{O}|\lambda) = \sum_{\mathbf{q}} \pi_{q_0} \prod_{t=1}^{T} a_{q_{t-1}q_t} b_{q_t}(\mathbf{O}_t) \ . \tag{2.14}$$

Direct evaluation of formula (2.14) would require summations over $N^T$ sequences, where $T$ is the length of the sequence and $N$ is the number of states. But by using the forward-backward algorithm proposed by Baum (Baum, 1972), the amount of computation can be reduced significantly.

The computation proceeds as follows. First the so called forward variable $\alpha_t(i)$ is introduced. It is defined as

$$\alpha_t(i) = P(\mathbf{O}_1, \mathbf{O}_2, \ldots, \mathbf{O}_t, q_t = i|\lambda) \ . \tag{2.15}$$

Because of the first-order Markov process assumption, $\alpha_t(i)$ can now be solved as follows:

1. Initialization:

$$\alpha_1(i) = \pi_i b_i(\mathbf{O}_1) \ , \qquad 1 \leq i \leq N \tag{2.16}$$

2. Induction:

$$\alpha_{t+1}(j) = \left[\sum_{i=1}^{N} \alpha_t(i) a_{ij}\right] b_j(\mathbf{O}_{t+1}) \ , \qquad \begin{array}{l} 1 \leq t \leq T-1 \\ 1 \leq j \leq N \end{array} \tag{2.17}$$

3. Termination:

$$P(\mathbf{O}|\lambda) = \sum_{i=1}^{N} \alpha_T(i) \tag{2.18}$$

From the second step of the forward-procedure, the dynamic programming nature of the solution is clearly seen. Figure 2.5 shows the computation graphically. It can be seen that the amount of calculations is of the order of $N^2T$.

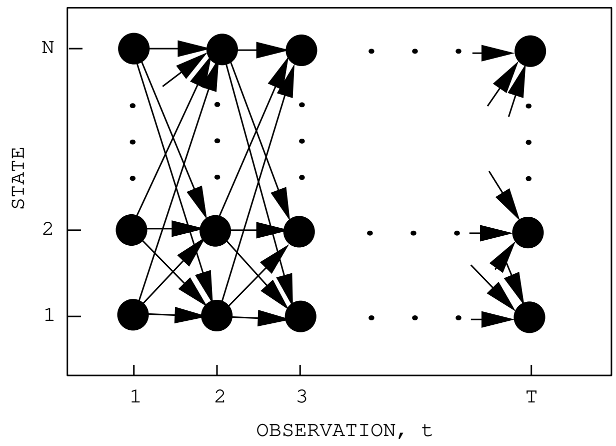


Figure 2.5: *Computation of $\alpha_t(i)$ begins from the observation at time $t = 1$ and continues step by step to the end of the observation sequence at time $t = T$.*

The backward-procedure is very similar to the forward-procedure. The only difference is that the computation starts from the time instant $T$ (last observation) and proceeds to the beginning of the observation sequence. The forward-variable is thus replaced by a backward-variable. The backward-procedure is not needed in the evaluation problem, but later in the HMM training (if so called Baum-Welch algorithm is used).

### 2.3.3 The decoding of the observation sequence

For finding the best state sequence associated to the observations, some kind of optimality criterion must be defined. If the criterion is to choose the most likely state sequence, the decoding problem can be solved in a very similar way as the evaluation problem. In the Viterbi algorithm (Forney, 1973), the computation proceeds in a sequential manner so that the probabilities of the previous subsequences have to be computed only once. When the interest is to find one sequence with the highest probability, only one path per state is needed in further calculations. Because of the dynamic programming, each time instant $t$ has to be processed completely before going to the time instant $t+1$. After that it suffices that each state holds only the most likely path from the initial state to that state and the probability of this path. This is again due to the property of the first-order Markov process. The following expression clarifies the case. Probability measure $\delta_t(i)$ is defined as follows:

$$\delta_t(i) \triangleq \max_{q_1,q_2,\ldots,q_{t-1}} P(q_1, q_2, \ldots, q_t = i, \mathbf{O}_1, \mathbf{O}_2, \ldots, \mathbf{O}_t|\lambda) \ . \tag{2.19}$$

Now the following recursion holds (because of the first-order Markov process assumption):

$$\delta_{t+1}(j) = \left[\max_i \delta_t(i) a_{ij}\right] b_j(\mathbf{O}_{t+1}) \ . \tag{2.20}$$

The complete algorithm is presented below. Variable $\psi_t(j)$ is needed for retrieving the state sequence.

1. Initialization:

$$\delta_1(i) = \pi_i b_i(\mathbf{O}_1) \ , \qquad 1 \le i \le N \tag{2.21}$$

$$\psi_1(i) = 0 \tag{2.22}$$

2. Recursion:

$$\delta_t(j) = \max_{1\le i\le N}[\delta_{t-1}(i)a_{ij}]b_j(\mathbf{O}_t) \ , \qquad 2 \le t \le T, \quad 1 \le j \le N \tag{2.23}$$

$$\psi_t(j) = \arg\max_{1\le i\le N}[\delta_{t-1}(i)a_{ij}] \ , \qquad 2 \le t \le T, \quad 1 \le j \le N \tag{2.24}$$

3. Termination:

$$P^* = \max_{1\le i\le N}[\delta_T(i)] \tag{2.25}$$

$$q_T^* = \arg\max_{1\le i\le N}[\delta_T(i)] \tag{2.26}$$

4. Path (state sequence) backtracking:

$$q_t^* = \psi_{t+1}(q_{t+1}^*) \ , \qquad t = T-1, T-2, \ldots, 1 \tag{2.27}$$

### 2.3.4 HMM training

In solving the estimation problem, the objective of maximum likelihood (ML) is often followed. This means to maximize $P(\mathbf{O}|\lambda)$ over all parameters of $\lambda$ for a given observation sequence $\mathbf{O}$. However, the maximum likelihood principle is not the only possibility. A different approach is to use a corrective training, when the criterion of training is to directly minimize the error rate on the training data (Bahl et al., 1988).

In the following, the segmental k-means algorithm is explained shortly, because it has been used in the experiments. The algorithm can be found in (Rabiner et al., 1986; Rabiner, 1989) and a more formal description in (Juang and Rabiner, 1990). Segmental k-means algorithm is an extended version of the Viterbi estimation (Jelinek, 1976) and it is based on a modified ML-principle. While the objective of the ML-estimation is to maximize

$$P(\mathbf{O}|\lambda) = \sum_{\mathbf{q}} \pi_{q_0} \prod_{t=1}^{T} a_{q_{t-1}q_t} b_{q_t}(\mathbf{O}_t) \ , \tag{2.28}$$

the optimization objective of the segmental k-means algorithm is

$$\max_{\mathbf{q}} P(\mathbf{O}, \mathbf{q}|\lambda) = \max_{\mathbf{q}} \pi_{q_0} \prod_{t=1}^{T} a_{q_{t-1}q_t} b_{q_t}(\mathbf{O}_t) \ . \tag{2.29}$$

This is called a state-optimized likelihood. The computational benefits compared to the maximization of (2.28) are evident.

The segmental k-means algorithm starts from an initialized model $\lambda$. It involves then iteration in two fundamental steps: 1) segmentation and 2) optimization. The segmentation step is equivalent to the decoding of an observation sequence and can be performed via Viterbi algorithm. This gives $\max_{\mathbf{q}} P(\mathbf{O}, \mathbf{q}|\lambda)$. Each observation vector is thus labeled according to the model $\lambda$. Given then the state sequence $\mathbf{q}$ and the observation sequence $\mathbf{O}$, the optimization step finds a new set of model parameters $\lambda'$ so as to maximize state-optimized likelihood (2.29). This can be written as

$$\lambda' = \arg\max_{\lambda} \left\{ \max_{\mathbf{q}} P(\mathbf{O}, \mathbf{q}|\lambda) \right\} . \tag{2.30}$$

By taking the logarithm of (2.30) gives

$$\lambda' = \arg\max_{\lambda} \left\{ \max_{\mathbf{q}} \left[ \log P(\mathbf{O}|\mathbf{q}, \lambda) + \log P(\mathbf{q}|\lambda) \right] \right\} . \tag{2.31}$$

Because $\max_{\mathbf{q}} \log P(\mathbf{q}|\lambda)$ is a function of only the state transition probability matrix $A$, and $\max_{\mathbf{q}} \log P(\mathbf{O}|\mathbf{q}, \lambda)$ is a function of only the observation distributions $B$, the optimization of (2.31) can be performed separately in two parts.

The updated estimate of $a_{ij}$ is

$$\hat{a}_{ij} = \frac{\text{number of transitions from state } i \text{ to state } j}{\text{number of transitions from state } i \text{ to any state}} . \tag{2.32}$$

The set of $\{\hat{b}_i(\mathbf{O}_t)\}_{i=1}^{N}$ is updated to give higher probability of being produced the observations $\{\mathbf{O}_t\}_{t=1}^{T}$.

One example of updating the estimate of $B$ is given below in order to give some kind of image of the procedure. In this example the codebook is used directly as a probability density approximation (the case of the discrete density HMM). The updated estimate of $b_{ik}$ ($k$ is the index of the code vector) is then

$$\hat{b}_{ik} = \frac{\text{number of the observation vectors classified into code vector } k \text{ in state } i}{\text{number of all observation vectors in state } i} . \tag{2.33}$$

This provides an updated model $\lambda'$. It replaces the old model $\lambda$ and the algorithm goes back to the segmentation part. This iteration is continued until a stopping criterion is met. The stopping criterion may be a threshold of the difference between two consequtive models or simply a maximum number of iterations.

### 2.3.5 Different density approximations of HMMs

In the following, discrete density HMM (DHMM), continuous density HMM (CDHMM) and semi-continuous density HMM (SCHMM) are introduced. As it can be noticed in the names, their differences are how the density functions of the observation vectors are modeled.

### Mixture density

Before going to the different HMM models, the term mixture density is introduced. The mixture density is expressed as

$$b(\mathbf{O}) = \sum_{m=1}^{M} c_m f_m(\mathbf{O}) \ . \tag{2.34}$$

$f_m(\mathbf{O})$ is the pdf kernel and $c_m$ is the weight of the $m$th mixture component. $M$ is the number of the mixture components and $\sum_{m=1}^{M} c_m = 1$. By varying the number of mixture components, densities of virtually any shape can be approximated.

### Discrete HMM

The discrete observation density HMM uses directly the VQ codebook. Each codebook vector is associated to the certain probability value. The probability density function is thus modeled by a set of small cells, which are the regions of the Voronoi tessellation.

The estimation of the probabilities $B$ can be performed in two phases: first the determination of the codebook and then the estimation of the probability values.

Forming the codebook is accomplished by some kind of VQ method. SOM provides an excellent tool for approximating the pdf. However, the minimization of the quantization error and the representation of the input space are not necessarily the best objectives for an optimal codebook. After all, the ultimate goal is to get as few recognition errors as possible. Discrimination ability of the codebook can be obtained by using LVQ (Kohonen, 1986b).

One advantage of the DHMM is that no form of the pdf is assumed. Another advantage is, that a powerful algorithm for discriminative training already exists (LVQ). Drawback of the DHMM is that VQ partitions the continuous acoustic space into separate (Voronoi) regions destroying the original signal structure. One solution to smooth this effect is to use a larger codebook, but in practice, the finiteness of the training data delimits the codebook size.

### Continuous density HMM

To obtain a continuous pdf estimate, the form of the pdf is first determined and then the parameters of the model are estimated. The most common parametric pdf used in CDHMMs is the mixture Gaussian density

$$b_i(\mathbf{O}_t) = \sum_{m=1}^{M} c_{im} b_{im}(\mathbf{O}_t) \quad , \quad i = 1, \ldots, N \ , \tag{2.35}$$

where $b_{im}(\mathbf{O}_t)$ is a $D$-dimensional multivariate Gaussian density with covariance matrix $\mathbf{\Sigma}_{im}$ and mean vector $\boldsymbol{\mu}_{im}$:

$$b_{im}(\mathbf{O}_t) = \frac{1}{(2\pi)^{D/2}|\mathbf{\Sigma}_{im}|^{1/2}} e^{-\frac{1}{2}(\mathbf{O}_t - \boldsymbol{\mu}_{im})^T \mathbf{\Sigma}_{im}^{-1} (\mathbf{O}_t - \boldsymbol{\mu}_{im})} \ . \tag{2.36}$$

When the observation source is simple, the mixture Gaussians provide an easy way to estimate the pdf. But if the observation source generates a complicated high-dimensional pdf and the number of mixtures must thus be large, the computations become difficult due to the excessive number of the parameters. Large number of model parameters require also large amount of training data. When the length of a feature vector increases, the size of the covariance matrix grows quadratically. To reduce the computation, covariance matrix may be replaced by a diagonal matrix. If the parameters of a feature vector are uncorrelated, no deterioration to the model is done.

The concept of parameter tying is also of importance. The idea is to reduce the number of free parameters. Tying similar parameters together gives more training data per each estimated parameter.

### Semi-continuous HMM

SCHMM (Huang and Jack, 1989) is a special form of the continuous mixture HMM. The output probability density function is written as

$$b_i(\mathbf{O}_t) = \sum_{k=1}^{M} c_{ik} f_k(\mathbf{O}_t) \quad , \quad i = 1, \ldots, N \; , \tag{2.37}$$

where $k$ is the index of the codebook vector and $f_k(\mathbf{O}_t)$ is a pdf kernel associated to it. SCHMM combines thus the discrete codebook and the continuous pdf. In the case of Gaussian pdf kernel, the $k$th nearest codevector to the observation vector $\mathbf{O}_t$ takes the place of $\boldsymbol{\mu}_{ik}$. All states of one HMM share the same codebook and kernels, as can be seen in the equation (2.37) ($f_k(\mathbf{O}_t)$ does not depend on the state $i$ and it is thus the same for all states). In the SCHMM used in this study, each phoneme had a five-state HMM as depicted in figure 2.3. The SCHMM of this study is based on the work done by Mikko Kurimo (Kurimo, 1994). Because all states in one phoneme model use the same codebook, there are now more training data available for each code vector. This is thus one form of the parameter tying.

In the case of continuous mixture density, $c_{ik}$ denotes the weight of the $k$th kernel $f_k(\mathbf{O}_t)$. But this can be considered also vice versa: VQ codebook is determined first and the probability value $c_{ik}$ is given for each code vector. The advantage of the discrete codebook to model a pdf nonparametrically is thus used. However, the VQ operations cause still quantization errors. Now, in order to smooth the quantization effect, a function of the quantization error is used as a weighting factor. This is denoted as $f_k(\mathbf{O}_t)$.

If the main idea is to eliminate the vector quantization effects of DHMM, the question is, what interpolation methods would be appropriate. A simple and intuitive way to make a pdf continuous would be to linearly interpolate the probability value between $M$ discrete codebook values according to the quantization error of the observation vector. However, in (Huang and Jack, 1989), Gaussian kernels were used and the performance of SCHMM was reported to exceed the performance of both DHMM and CDHMM.

### 2.3.6 HMM problems

There are some problems involved to the use of HMM in the speech recognition. One limitation of the model is the assumption of the first-order Markov process. It can be expected that this kind of model is inflexible to exploit certain acoustic-phonetic cues like duration and spectral transients.

To compensate the problem of the first order model, feature vector should then contain sufficient information about its context. On the other hand, originally it was the HMM, which was supposed to use the temporal information of the speech. If that is included already in the feature vector, the question raises, whether there should be then some changes in the HMM.

Finally, some problems in the HMM training are typical to any real world parameter estimation tasks. The limited size of the training data must be taken into account. If all class interactions are not represented in the training set, some model parameters may be left undetermined.

# Chapter 3

# Feature vectors

In the following sections several possibilities for feature vectors used in speech recognition are described. The material is divided into two sections. First several short time feature vectors are introduced. They serve as building blocks in the later feature processing. The combinations of several successive short time feature vectors are denoted as long time feature vectors. They can be also called context vectors. An overview of the feature vectors used in speech recognition can be found in (Picone, 1993). Topics concerning the features and human auditory system can be found in (Karjalainen, 1987).

## 3.1 Short time feature vectors

Short time feature means a feature vector, which has been computed from one short time data frame. In this chapter, different short time feature vectors are introduced.

### 3.1.1 Raw data

Analog speech signal is first sampled using 16 bits at 16 kHz sampling rate. An example of a speech signal is shown in figure 3.1. High frequencies are enhanced by using a simple first-order filter, whose transfer function is $H(z) = 1 - cz^{-1}$. Parameter $c$ determines the location of the zero in $z$-plane. The value 0.95 has been used. In time domain the filtering is $y(n) = x(n) - cx(n-1)$, where $y(n)$ is the output of the filter and $x(n)$ is its input. The resulting data stream is then truncated into frames at fixed time intervals. The frames are weighted by using the Hamming window and consequtive data frames overlap giving better time resolution. Window length is 256 samples (16 ms) and the interval between two windows is 128 samples (8 ms from the beginning of the previous window).

### 3.1.2 Spectrum

The speech recognition is usually in one way or another based on the spectrum content of the speech. FFT provides an efficient method to compute the spectrum (Rabiner and Radar, 1972). Figure 3.2 shows an example of the spectrogram.

Formants are the resonance frequencies of the vocal tract. Moving the articulators alters the shape of the vocal tract, which in turn changes the frequency response of the speech production system. By examining formant movements, different phones in the spectro-

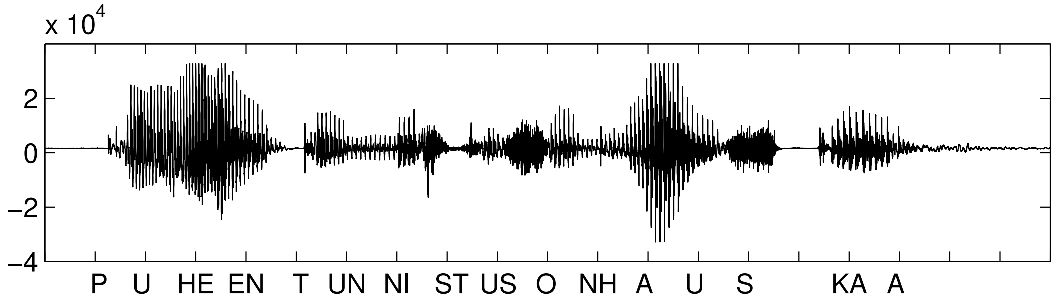


Figure 3.1: *2 seconds of speech. Tickmarks are at intervals of 100 ms. This speech is an example of a continuous speech and there are no breaks between the words. It can be clearly seen that silence precedes each of the plosives /P/, /T/ and /K/.*

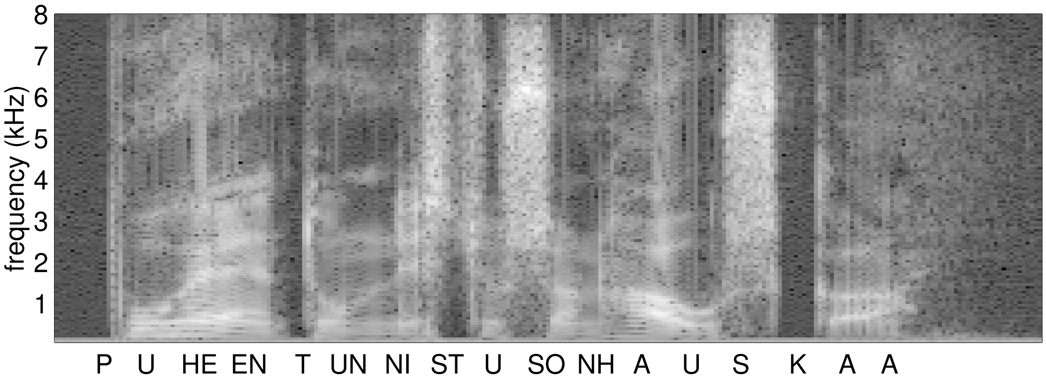


Figure 3.2: *Spectrogram. Consequtive power spectrum vectors are plotted along the time. Dark color indicates weak energy and bright color indicates peaks in the spectrum. The peaks occur due to the resonances of the vocal tract. These resonances are called formants. The movements of the formants are clearly visible in this spectrogram as white paths along the time. It can be also seen that the energy of /S/ is concentrated on the high frequencies.*

gram can be identified. However, the 128-point power spectrum vector (first half of the symmetric 256-point power spectrum) is not used directly as a 128-point feature vector. Instead, it is divided into the fixed number of channels. Each channel can then be thought to be an output of one bandpass filter of the filter bank. Filter bands may be located nonuniformly on the frequency scale, and their passbands may overlap.

Several possibilities to choose the filter bank exist. Some experiments were done by using simulated annealing[1] to find the optimal passband widths, locations and weights of the filters. The method was similar than in (Liu, 1995), but here it was used in phone center classification. However, the results were not better than using the mel-scale filter bank. This doesn't mean that the mel-scale filter bank would be the optimal choice for speech recognition, but even if a better one were found, it would probably not alone improve the recognition rate significantly.

Mel-scale filter bank can be thought to approximate the critical bands of the human's inner ear. Filter spacing is linear up to 1000 Hz and after that follows the logarithmic spacing. However, in the filter bank the filter locations are fixed, which is not true for the critical bands in the human's ear. This lack is partly compensated by setting passbands of the adjacent filters to overlap, giving thus better resolution to spectral changes.

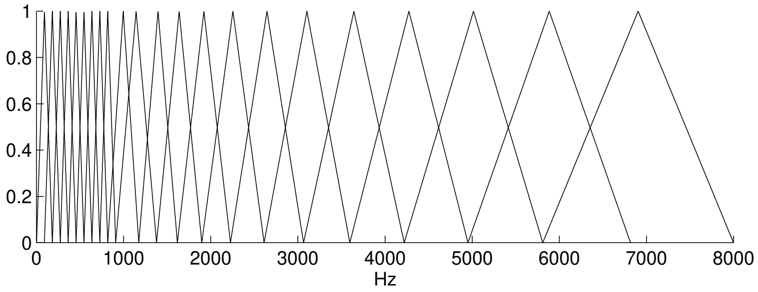


Figure 3.3: *Mel-scale filter bank.*

### 3.1.3 Cepstrum

Cepstrum belongs to the area of homomorphic signal processing (Oppenheim and Schafer, 1975). It was first introduced in 1963 by Bogert, Healy and Tukey. They found that the logarithm of the power spectrum of a signal containing an echo had an additive periodic component due to this echo, and thus the Fourier transform of the log-power spectrum should exhibit a peak at the echo delay (Bogert et al., 1963).

By considering the speech production as a linear time varying system, it can be denoted as $S = VE$, where $V$ is the transfer function of the vocal tract and $E$ is the excitation[2]. By taking the logarithm of $S$, we get $\log(S) = \log(V) + \log(E)$. Now if the spectral contents of $V$ and $E$ differ sufficiently from each other, it is possible to separate their contributions by linear filtering. Since the formant structure of $V$ varies slowly in frequency compared to the harmonics or noise in $E$, the linear separation after an inverse Fourier transform can be done (O'Shaughnessy, 1987). Figure 3.4 shows an example of a cepstrum. It has been computed from the whole 64 ms time window.

[1] Simulated annealing is a stochastic optimization algorithm (Kirkpatric et al., 1983).

[2] The lip radiation would also belong to the model, but it has been left out of the model. It can assumed to be the same for all phonemes.

The complex cepstrum involves the use of the complex logarithm, but the cepstrum, as it has traditionally been used, involves only the logarithm of the magnitude of the Fourier transform (Rabiner and Schafer, 1978). By using the inverse discrete Fourier transform of the logarithmic magnitude spectrum, the cepstrum can be computed as

$$c(n) = \frac{1}{N}\sum_{k=0}^{N-1} \log|S(k)| e^{j(2\pi/N)kn} \quad , \quad 0 \leq n \leq N-1 \ . \tag{3.1}$$

By noting that the log magnitude spectrum is a real symmetric function, the equation above can be simplified to a discrete cosine transform. Cepstral coefficients can be calculated also from the LPC coefficients (Rabiner and Schafer, 1978).

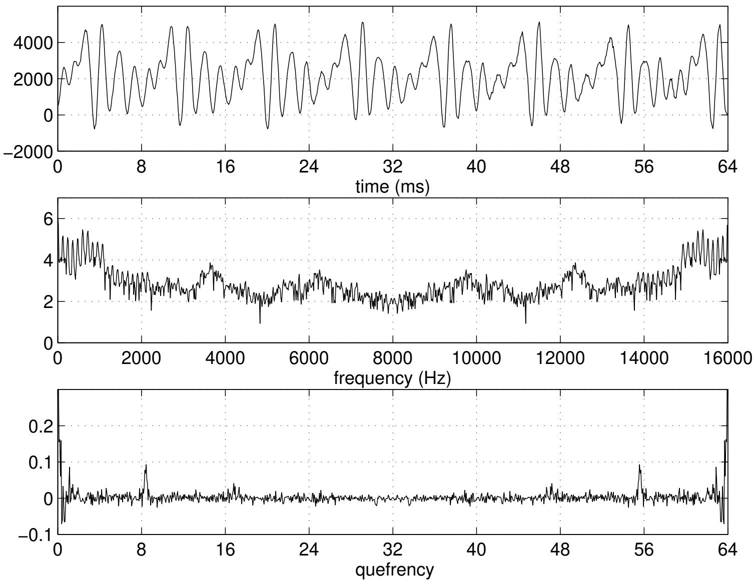


Figure 3.4: *Raw data, log magnitude spectrum and real cepstrum. Clear peak in the cepstrum can be found due to $F0$. It can be read from the plot that $F0$ is around 125 Hz (1/8 ms). In order to get the peak of $F0$, the length of the time window must be greater than one period of speech. In this case it was 64 ms, but 32 ms would be also sufficient. c(0) has been removed from the cepstrum plot.*

$c(0)$ is the average value of the logarithmic magnitude spectrum. It represents thus the power of the speech. All the rest cepstral coefficients are free from the power level of the speech. Because cepstrum is the spectrum of the spectrum, the first lowest coefficients characterize the envelope of the speech spectrum. Thus very few of them are enough to represent the smooth shape of the spectrum.

### 3.1.4 Mel-cepstrum

In (Davis and Mermelstein, 1980), a slightly different method for calculating M cepstral coefficients was proposed:

$$melc(i) = \sum_{k=1}^{K} \log|X_k| \cos[i(k-\frac{1}{2})\frac{\pi}{K}] \quad , \quad i = 1, 2, \ldots, M \ , \tag{3.2}$$

$X_k$ is the output of the $k$th bandpass filter from mel-scale filter bank and $K$ is the number of filters. The two main differences between formulas (3.1) and (3.2) are that in (3.2) the frequency scale is not linear, but follows the mel-scale and the spectrum is smoothed because of the filter bank.

Mel-cepstrum has been proved to be very useful in the speech recognition. In (Jankowski Jr. et al., 1995), mel-cepstrum outperformed LPC based cepstrum. Because of the cosine transform in the formula (3.2), the mel-cepstrum can be thought to be related to the principal component analysis (PCA)[3]. In fact, investigating the variances of the mel-cepstrum components, it can be seen that the variance of the component $melc(i)$ decreases as $i$ increases.

It should be noted that the fundamental frequency F0 cannot be detected by using mel-cepstrum, because the periodic component in the spectrum caused by the glottal excitation gets lost to the widths of the bandpass filters of the filter bank.

### 3.1.5 Linear Predictive Coding

The most characteristical features of the speech consist of the peaks of the spectrum. The peaks are due to the resonances of the vocal tract. These resonances are called formants. When the speech signal is considered to be stationary during a frame of N samples, the vocal tract can be modeled by constant coefficients (in that frame). Let us denote the vocal tract model as a filter $H$. If $H$ may have both zeros and poles, we have an ARMA model, but if the vocal tract is considered to be a closed transfer line, we can model it by using only poles and thus have an AR model. This is naturally a simplification since the actual vocal tract has zeros due to the glottis as well as the nose. However, the spectral peaks as well as general shaping of the spectrum can be modelled by using an all-pole model. The transfer function of an AR model is of the form

$$H(z) = \frac{Y(z)}{X(z)} = \frac{G}{1 - \sum_{k=1}^{M} a_k z^{-k}} \ , \tag{3.3}$$

where $G$ is the gain parameter, $M$ is the order of the model and $a_k$ is the $k$th coefficient. In time domain, this is

$$y(n) = \sum_{k=1}^{M} a_k y(n-k) + Gx(n) \ . \tag{3.4}$$

There are three basic methods for estimating the coefficients $a_k$. These are the autocorrelation method, the covariance method and the lattice method. The first two methods differ from their assumptions of the modeled system and also their estimates of coefficients $a_k$ may differ. The autocorrelation method gives always stable $H(z)$, which means that its roots lie inside the unit circle, but for the covariance method this is not necessarily true. Although it is a simple procedure to reflect a pole from outside the unit circle to inside preserving the same frequency response, in speech recognition, the autocorrelation method is almost exclusively used. Durbin's recursive solution for the autocorrelation method and Cholesky decomposition for the covariance method as well as solution to the lattice formu-

[3] The name of the transform is principal component, eigenvector, Hotelling, or Karhunen-Loève transform (Gonzalez and Woods, 1992).

lation can be found in (Rabiner and Schafer, 1978). There are also comparisons between all these three methods in that book.

Before computations, the model order has to be fixed. There should be enough poles to model the formants and also some additional poles to approximate the possible zeros as well as the general spectral shaping. If it is assumed that formants occur at every 1 kHz and the sampling rate is 16 kHz, we need $2{\cdot}8 + 4 = 20$ poles for the model. 8 kHz is half of the sampling rate and also the upper limit to the observed spectrum, coefficient 2 is because poles exist as complex conjugate pairs and the rest 4 poles are additional.

LPC analysis is widely used in various fields of speech processing, especially in speech analysis. The method can be used in various different ways, e.g., using the coefficients directly, or alternatively to compute some derivatives of them like reflection coefficients. One possibility is also to calculate the spectrum of the signal from these coefficients. An example of this is in figure 3.6. The spectrum is smooth compared to the Fourier-based spectrum estimates. But because the spectrum vector is later divided into channels, this smoothness doesn't necessarily bring any advantage in the speech recognition task.

### LPC in formant tracking

By calculating the LPC coefficients $a_k$ we get the denominator of the transfer function in polynomial form. By factoring this denominator polynomial into first-order factors, we get the poles of the transfer function. From them we can obtain the pole locations at the frequency axis. It is simply a projection from the $z$-plane to the unit circle. The denominator of 3.3 is expressed as

$$1 - \sum_{k=1}^{M} a_k z^{-k} = \prod_{k=1}^{M} (1 - p_k z^{-k}) = z^{-M} \prod_{k=1}^{M} (z - p_k) \ , \tag{3.5}$$

where $p_k$ is the complex pole $p_k = r_k e^{j\phi_k}$.

By Euler's rule we get

$$r_k e^{j\phi_k} = r_k \cos(\phi_k) + j r_k \sin(\phi_k) \ . \tag{3.6}$$

Now, because

$$\frac{Im(p_k)}{Re(p_k)} = \frac{r_k \sin(\phi_k)}{r_k \cos(\phi_k)} = \tan(\phi_k) \ , \tag{3.7}$$

we can solve the pole location

$$\phi_k = \arctan\left(\frac{Im(p_k)}{Re(p_k)}\right) \ . \tag{3.8}$$

Figure 3.7 shows an example of the pole locations in the $z$-plane as well as projected to the frequency axis.

The information about pole locations can then be used for tracking the formant movements. This can be done by calculating the differences of pole locations at fixed time delay apart. This could be beneficial especially in the speech segmenting, where the interest is to find the borders of the phones, but not yet to classify the phones into the phonemes. It could be thought that the pole locations would move more in phone transitions than in phone centers. However, one shortcoming of this method is the lack of knowledge which coefficients correspond to which formants. Even if the poles would be projected into unit circle and sorted according to the angle frequency, problems may occur if the poles do not represent formants but model only the general spectral shape. No guarantee is given that the calculated poles are really formants. LPC is a parametric method, and it may be sensitive to noise. The additional poles used for general spectral shaping may cause troubles, because the pole movements can be something quite different than the formant movements. And even if the true formants were obtained, problems may occur, if two formants cross each other.

Some experiments were done, where the feature vector was the difference of two LPC coefficient vectors. Such features were calculated in phone centers and phone transitions and these two classes were tried to be separated by using LVQ. This approach wasn't very successful, but it raised many questions and thoughts about the possibilities of LPC.

#### Other LPC methods

Finally, there exist several modifications to the basic LPC. Different variants of LPC was however not the main emphasis in this study. The reason for preliminary experiments was just the interest to see if formant movements could be caught somehow by using LPC. Despite the slightly discouragin results, the use of some kind of LPC coefficient derivative seems still a tempting idea for formant tracking, but the topic needs further studies and more careful considerations. One idea would be to use many simultaneous models of different order and then compare the poles of each model and try to find the formants that way.

### 3.1.6 Group delay spectrum

When a transfer function has a pole near the unit circle, the phase response changes abruptly. Figure 3.5 shows how magnitude response and phase response change when a pole is moved towards the unit circle.

Since formants correspond to complex-conjugate pairs of poles with relatively narrow bandwidths, i.e. the poles lie near the unit circle, each spectral peak associated with a phase shift is affiliated with a formant (O'Shaughnessy, 1987, page 225). This can be used in deciding whether a spectral peak really corresponds to a formant. But the information about the phase shift can be used also directly.

Let the phase response be denoted as $\Theta(\omega)$. Group delay is defined as $-\frac{\partial\Theta(\omega)}{\partial\omega}$. Group delay is thus a natural choice for determining the amount of a phase shift. As it can be seen in the figure 3.6, the shape of the group delay spectrum looks similar to the magnitude of the LPC spectrum, but the peaks are enhanced. Figure 3.7 shows the group delay spectrum with the pole locations of the LPC transfer function. The magnitude of one peak is mainly determined by the distance of the nearest pole to the unit circle.

Itakura and Umezaki have proposed the idea of the smoothed LPC group delay spectrum (Itakura and Umezaki, 1987; Itakura et al., 1990). Their idea was to smooth too sharp peaks of the group delay. But their interest was on the final speech recognition. If we are interested only in speech segmenting, sharp peaks could be rather an advantage in

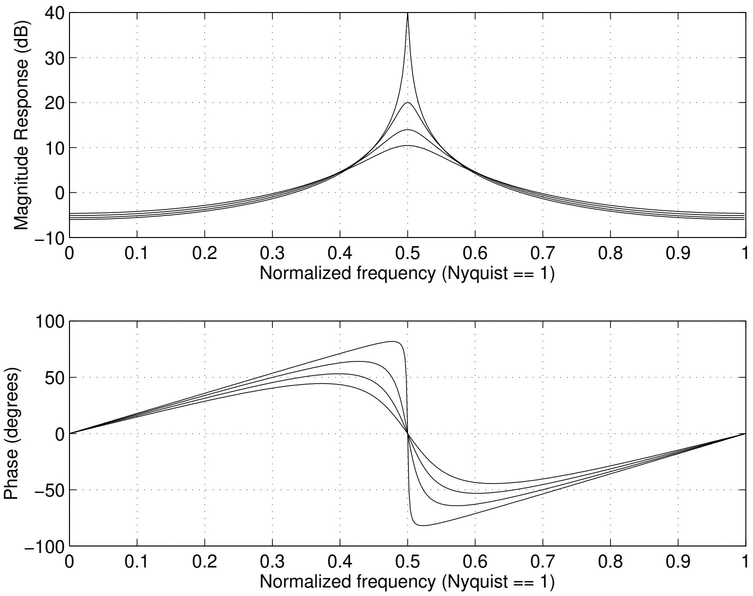


Figure 3.5: *Magnitude and phase responses of the transfer function* $H(z) = \frac{1}{1+az^{-2}}$ *with the following values of a: 0.7, 0.8, 0.9 and 0.99. When the pole is moved towards the unit circle, the peak in the magnitude response becomes sharper and the phase shift becomes more abrupt.*

finding the formant movements. For that task the differences of two group delay spectra from fixed time delay apart could be used. Some experiments to separate phone centers from phone transitions were done but without any remarkable success. In the method experimented, the group delay spectrum was fed to filter bank to get a feature vector. The method studied was thus a simple formant enhancement procedure.

Similar statements can be said about group delay than about LPC spectrum. For the human's eye it looks better than conventional Fourier spectrum. But to benefit from the spectral sharpening, the bands of the filter bank should be sufficiently narrow so that formant movements could be observed in the output of the filter bank. If one single band of a filter is too wide, both the initial position and the final position of the formant remain inside the same filter band or affect only weakly other filter bands.

### 3.1.7 Power of the speech signal

The power of the speech from one data window can be calculated by summing all the outputs of a filter bank. The power can also be calculated directly from the samples by taking the square root of the average of the squared sample values. This is then the root mean square of the data frame. The weighting window has been left out of the equation.

$$\mathrm{rms} = \sqrt{\frac{1}{N}\sum_{k=0}^{N-1} s(k)^2} \tag{3.9}$$

One single rms value does not necessarily bring any significant information about the speech, but several of them from a longer time span and their differences can bring information about the dynamics of the speech. This can be useful especially in the phone segmentation. rms can be used also to detect the start and the endpoint of the utterance.

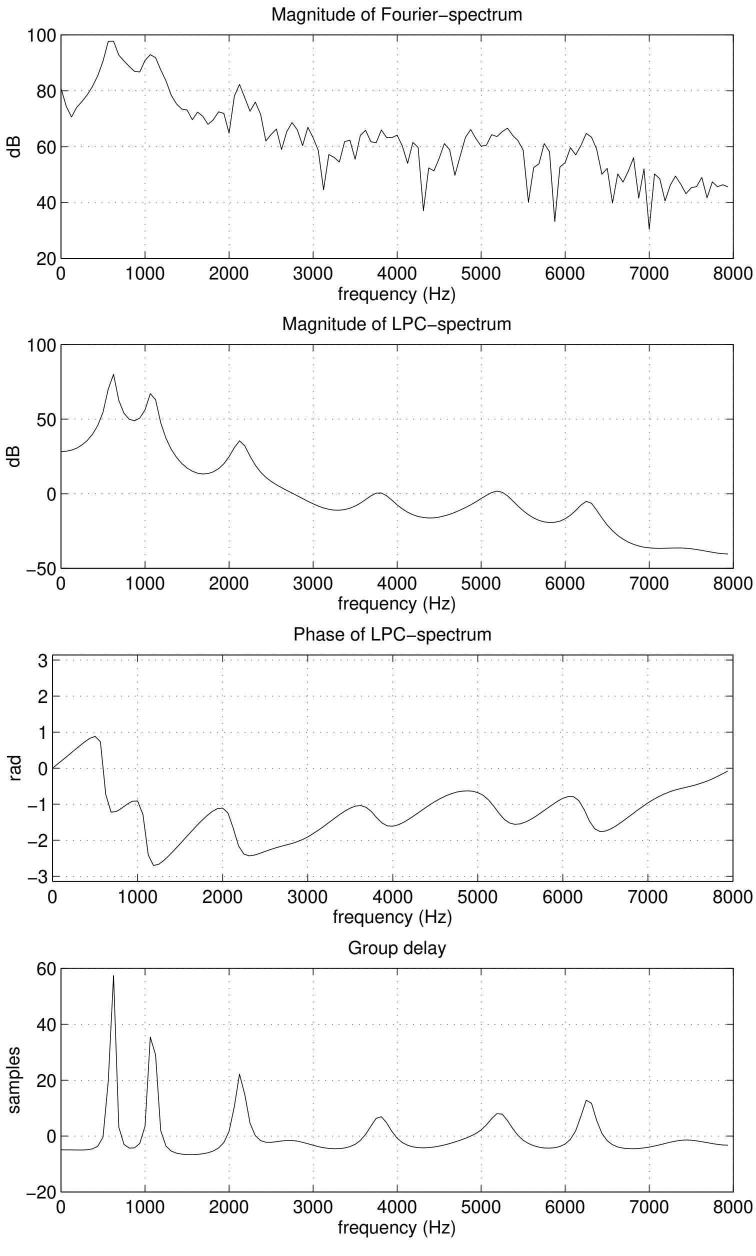


Figure 3.6: *Fourier magnitude spectrum, LPC magnitude spectrum, LPC phase spectrum and group delay spectrum. These have been computed using one 16 ms window from the last /A/ in figure 3.1.*

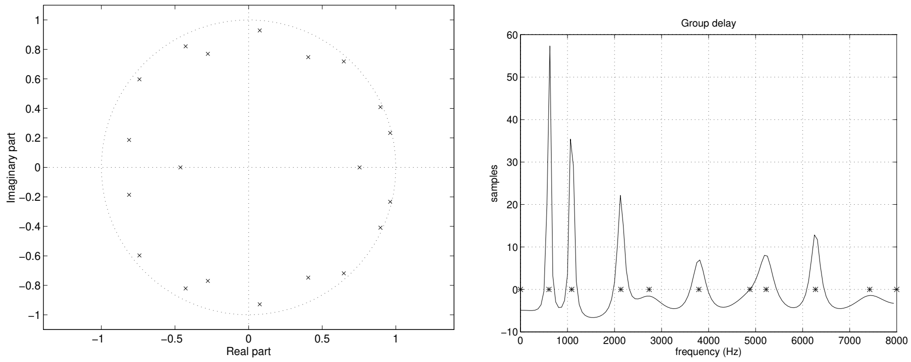


Figure 3.7: *In the right group delay spectrum with pole locations. Pole locations in the z-plane are shown in the left. It can be seen how they are projected to the unit circle which resembles the frequency axis. The poles were computed as described in section 3.1.5.*

## 3.2 Choosing the best short time feature vector

20-dimensional mel-cepstrum was chosen to be the building block of further feature extraction process. Also rms value was considered to be useful. Both of these have already been successfully used in previous studies in the laboratory. Although it cannot be stated that the chosen cepstrum vector would be the optimal one, good results have been obtained by using it.

Comparisons between different preprocessing methods have been done in (Jankowski Jr. et al., 1995). Several signal processing front ends were compared against mel-cepstrum. It was interesting to note, that auditory model based preprocessing methods were not better than mel-cepstrum when the speech to be recognized was clear or moderately noisy. Auditory model based front ends were slightly better only when there was much noise at the present. But even then, the differences of the performances were small. Furthermore, auditory model based front ends require much more computation than mel-cepstrum. In the same article, also comparisons between LPC cepstrum and mel-cepstrum were done. Mel-cepstrum outperformed LPC based cepstrum clearly in the presence of noise. The choice of the mel-cepstrum as a short time feature vector can thus be justified.

## 3.3 Long time feature vectors

The term long time feature vector means here a feature vector which has information from a longer time span than just one 16 ms data frame. Also the term context vector can be used. Long time feature vectors are formed by combining several short time feature vectors. This can be done by averaging, concatenation and calculating differences of them. Averaging reduces the influence of noise. Averaging several Fourier spectra computed from consequtive data windows gives more reliable spectrum estimate. Prerequirement for this is that the signal remains stationary enough during the consequtive time windows. Both the concatenation and difference computation bring information about the temporal structure of the speech. All these three techniques can be combined in multiple ways for calculating one feature vector.

### 3.3.1 General approach for short time feature vector combination

In (Picone, 1993) a compact representation for a system able to do all kind of linear filtering was introduced. The linear filtering methods include all linear average calculations and difference calculations. The representation starts by defining a signal measurement matrix

$$\mathbf{X} = \begin{bmatrix} x(0,0) & x(0,1) & \dots & x(0,N_x-1) \\ x(1,0) & x(1,1) & \dots & x(1,N_x-1) \\ \vdots & \vdots & \ddots & \vdots \\ x(N_f-1,0) & x(N_f-1,1) & \dots & x(N_f-1,N_x-1) \end{bmatrix} \tag{3.10}$$

where $x(n,m)$ denotes the $m$th signal measurement at frame $n$. $N_x$ denotes the total number of signal measurements for each frame. A signal measurement means here an element of one short time feature vector, so it doesn't have to be a simple raw data sample. One row contains all features calculated from one time window. In (Picone, 1993) $N_f$ denoted the total number of frames in the signal, but here the number of rows ($N_f$) in the matrix depends on the number of short time features needed for one long time feature vector calculation. This is of course merely an implementational aspect. The matrix can be stored to the buffer, and every time, when a new feature vector is obtained, it replaces the oldest feature vector. This can be easily implemented by using modulo arithmetic in memory address indexing, in the similar way than ring buffering is implemented in signal processors.

Since the introduction of a signal measurement matrix in (Picone, 1993) was more a conceptual one, it is not continued further here. It suffices to note that it is possible to represent all the feature extraction procedures in a compact manner, if needed.

### 3.3.2 Concatenation

Concatenation means that two or more feature vectors are put into one feature vector after one each other. The features to be concatenated may represent similar or different feature types. They can also be calculated from a single or several time windows. For example, an rms value can be concatenated to the cepstrum vector calculated from the same time window or two cepstrum vectors calculated from two different time windows can be concatenated. In concatenation, the dimension of the final feature vector equals to the sum of the dimensions of the feature vectors to be concatenated. High-dimensional feature vectors cause more computational load in distance calculations and they may require also more reference vectors in a codebook in order to get good classification results.

### 3.3.3 Delta features

Although the term delta feature may have originated to mean the time derivative of the feature stream (Rabiner et al., 1988), here it means any kind of difference of the feature vectors. Delta features or simply deltas are thus feature vectors, which tell about the temporal changes of the speech. The simplest approximation to the first-order time derivative in signal processing is a *finite difference approximation*, FDA.

$$\mathbf{d}(t) = \frac{\mathbf{x}(t) - \mathbf{x}(t-\tau)}{\tau} \tag{3.11}$$

$\mathbf{x}(t)$ is the feature vector calculated from speech signal. When $\tau$ approaches zero, equation (3.11) becomes a definition of derivative. However, the time interval $\tau$ doesn't have to be the smallest possible value. In the speech recognition task it can be more profitable to compute the deltas from a longer time interval.

The clusters in figure 3.8 represent different classes. Classes are overlapping and one single feature vector doesn't necessarily tell what class one feature vector represents.

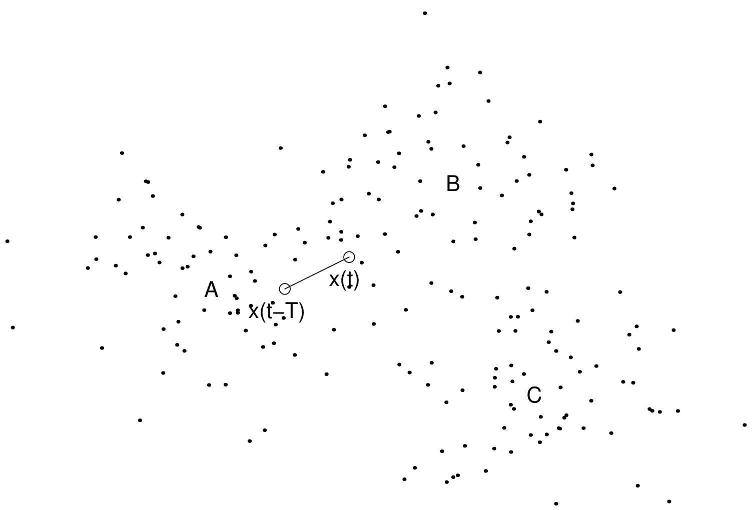


Figure 3.8: *Feature classes and two feature vectors calculated $\tau$ time units apart from each other.*

Let us investigate two feature vectors at time instants $t - \tau$ and $t$. Feature vector $\mathbf{x}(t)$ is in the region where two classes $A$ and $B$ overlap. It cannot thus be classified surely to either of the classes. However, the difference vector $\mathbf{x}(t) - \mathbf{x}(t-\tau)$ is directed towards class $B$, and that indicates, that the next class would be likely class $B$. If HMM is used as a classifier, this difference of the features can help in choosing the right state of the Markov model. However, it could be argued that the same effect can be achieved by concatenating these two vectors. The consequent concatenated feature vector would contain also in that case the information about the direction of the feature stream.

If $\tau$ is very small, the difference of two feature vectors doesn't bring clear picture about the trend of the feature stream, and consequtive deltas are pointed to random directions because of noise and small alterations in the speaker's voice. This is depicted in figure 3.9. The best value for $\tau$ depends naturally on the speech to be recognized.

Delta features can be computed in several ways. The following delta is a simple difference symmetrical around the time instant $t$.

$$\mathbf{d}_s(t) = \frac{\mathbf{x}(t + N_d) - \mathbf{x}(t - N_d)}{2N_d} , \tag{3.12}$$

where $2N_d$ is the chosen time difference. Although equation (3.12) is noncausal, because it requires samples following the time instant $t$ to be known, in practice this causes only delay of $N_d$ samples in the computation of the delta.

Deltas can also be computed by using regression analysis (Furui, 1986)

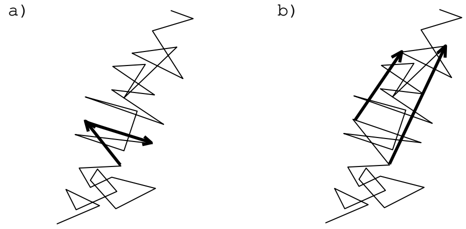


Figure 3.9: *Feature vectors in feature space plotted along time. Arrows are difference features plotted at different time spans $\tau$ apart. In a), where $\tau$ is small, the consequtive deltas point to arbitrary directions, but in b), where $\tau$ is larger, the consequtive deltas point more to the same direction. This describes thus better the trend of the feature vectors in a larger time scale.*

$$\mathbf{d}_r(t) = \frac{\sum_{n=-N_d}^{N_d} n\mathbf{x}(t+n)}{\sum_{n=-N_d}^{N_d} n^2} \ . \tag{3.13}$$

This represents a linear phase filter approximation to an ideal differentiator (Picone, 1993). In (Furui, 1986) $N_d$ was set to three and the total interval for delta computation was thus seven frames (56 ms). However, the best recognition results using only cepstral regression coefficients were obtained by using nine frames (72 ms).

However, simple difference has been reported to perform slightly better than the regression analysis (Lee, 1988). The best results were obtained using a 40 ms time interval.

It should be noted that the use of denominators in the formulas (3.12) and (3.13) is unnecessary, if the same parameters are used all the time.

### 3.3.4 Deltas and correlation

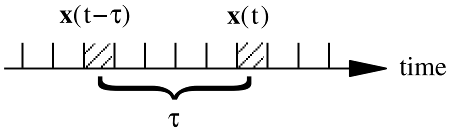


Figure 3.10: *Feature stream. Two feature vectors $\tau$ time units apart each other.*

The use of deltas is not meant to approximate the time derivate of the feature vectors but rather to measure the similarity (or dissimilarity) of two feature vectors along the time. Figure 3.10 shows a feature stream. Let us investigate a simple delta defined as

$$\mathbf{d}(t) = \mathbf{x}(t) - \mathbf{x}(t-\tau) \ . \tag{3.14}$$

Average[4] power of delta feature $\mathbf{d}(t)$ is

$$\begin{aligned} P_d &= \langle \mathbf{d}^T(t)\mathbf{d}(t) \rangle \\ &= \langle [\mathbf{x}(t) - \mathbf{x}(t-\tau)]^T [\mathbf{x}(t) - \mathbf{x}(t-\tau)] \rangle \\ &= \langle \mathbf{x}^T(t)\mathbf{x}(t) \rangle - 2\langle \mathbf{x}^T(t)\mathbf{x}(t-\tau) \rangle + \langle \mathbf{x}^T(t-\tau)\mathbf{x}(t-\tau) \rangle \\ &= 2P_x - 2\langle \mathbf{x}^T(t)\mathbf{x}(t-\tau) \rangle \end{aligned} \tag{3.15}$$

Because $P_x$ can be considered a constant, maximizing $P_d$ means the same as minimizing the term $\langle \mathbf{x}^T(t)\mathbf{x}(t-\tau) \rangle$. This term is the autocorrelation of the feature vector stream. Autocorrelation measures the relatedness or dependence of the signals and by choosing $\tau$ so that the autocorrelation is minimized means thus the same as maximizing the measure of dissimilarity. This shows that delta features are related to the correlation of the feature stream.

Intuitively it is clear that the difference of two vectors, what best represents the change of the feature stream has also the largest power.

[4]Averaging operator $\langle \cdot \rangle$ is defined as $\langle z(t) \rangle = \lim_{T\to\infty} \frac{1}{T} \int_{-T/2}^{T/2} z(t)dt$.

# Chapter 4

# Discriminative features

The first goal to select the appropriate features was to find a good feature vector, which could separate phone centers from phone transitions. The starting point for this was the assumption that phone classification is not a hard problem as long as the phone locations in a word are known. Then the classification can be based on either classification of phone centers or classification of phone transitions. The latter means the same as classifying phone pairs. But even if the classifier doesn't have a separate segmentation stage, like in the case of the HMM, phone center/phone transition -classification provides information about the discriminative capability of a feature vector. Therefore, several experiments were done in order to find a feature vector, which could discriminate phone center and phone transition classes. Delta features were supposed to be especially useful for this task.

## 4.1 Experimented mel-cepstrum combinations

Combinations of 20-dimensional mel-cepstrum vectors are introduced here. Experiments using averaging and concatenation have been done also earlier (Mäntysalo, 1992; Mäntysalo et al., 1994), but this study brings the delta features as a new element.

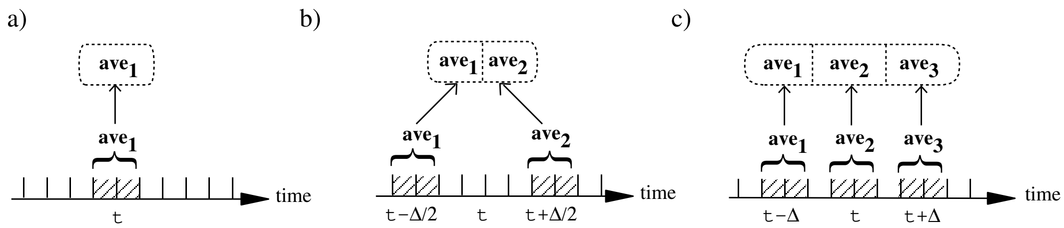


Figure 4.1: *Averaging and concatenation of feature vectors. a) single feature vector, b) two concatenated feature vectors, c) three concatenated feature vectors.*

Concatenation is depicted in figure 4.1. All feature vectors to be concatenated may be averaged. This gives thus two parameters, the number of feature vectors to be averaged and the time span between the feature vectors to be concatenated. Although there are two time spans to be determined in figure 4.1, one common value for both was used in the experiments.

In the simplest form, deltas are computed as depicted in figure 4.2a. In the static part of a phone, like the phone center, the difference of two features is expected to be small.

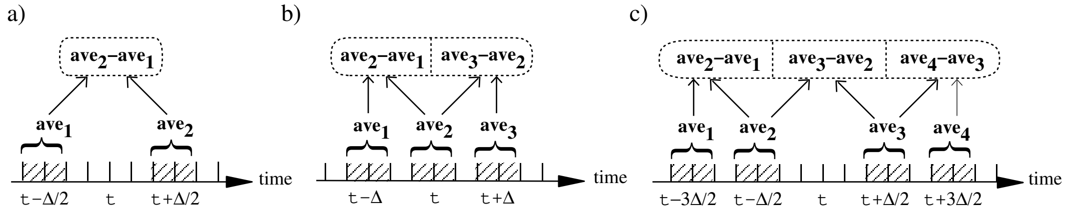


Figure 4.2: *Delta computations. a) single delta, b) two concatenated deltas, c) three concatenated deltas.*

But if the centroid of the delta computation window is in the transition part of a phone, two feature vectors belong to different phones and the delta feature should have bigger magnitude.

In the figure 4.2b, two deltas are concatenated. Let us suppose that the time window covers the duration of one phone. If the centroid of the time window is in the middle of the phone, the beginning and end the time window are then in transition parts. Again, different values of deltas are expected for phone centers and phone transitions.

If three deltas are concatenated (figure 4.2c) and the time window covers slightly more than one phone duration, the middle delta should have small values in the middle of a phone. If the beginning and the end of a time window reach the previous and ensuing phones, the corresponding deltas should have bigger values. Respectively, if the middle delta is in the transition part of a phone, the magnitudes of deltas are vice versa.

The delta concatenation procedure could be continued further by adding more and more deltas. But although this would bring more information to the feature vector, its dimension would grow at the same time. The limit must be set somewhere in order to handle the codebook calculations. Three previously introduced feature types were intuitively meaningful.

Because deltas can be thought to tell information only about the changes of speech, another feature type was also experimented, where a static feature vector was added to the previously introduced delta features.

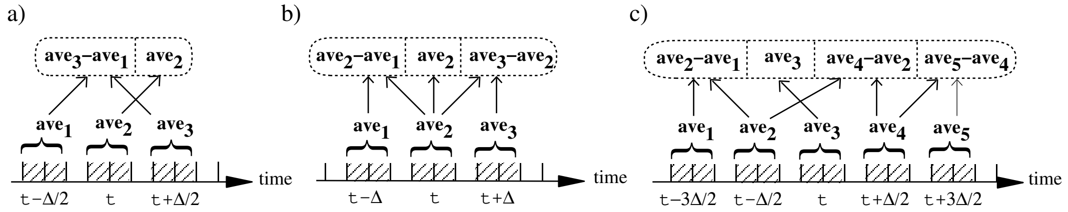


Figure 4.3: *Delta features with a static component. a) single delta, b) two concatenated deltas, c) three concatenated deltas.*

Feature vector proposed in the figure 4.3b is intuitively appealing. It can be considered as a phone model. There is a static part which fixes the feature to the certain phoneme class. Deltas in the beginning and in the end of the phone take the effect of the previous phone and the ensuing phone into account. Coarticulation effect has thus been taken into account.

In addition to the previous one, the feature in the figure 4.3c takes also the change in the middle part of the phone into account.

To continue further, all building blocks of the delta computations and static parts could be also concatenated feature vectors. However, in the experiments, only averaging was used.

Table 4.1 shows the number of short time data frames needed in the long time feature calculation for each feature type. The number of averaged features is denoted as *ave* and the time span of two consequtive feature vectors in the concatenation and delta computation is denoted as *delta*. The unit of *ave* and *delta* is one short time data frame.

| feature type | number of data frames |
|---|---|
| 1 static | $ave$ |
| 2 static | $delta + ave$ |
| 3 static | $2 \cdot delta + ave$ |
| 1 delta | $delta + ave$ |
| 2 deltas | $2 \cdot delta + ave$ |
| 3 deltas: | |
| nonoverlapping middle delta | $3 \cdot delta + ave$ |
| overlapping middle delta | $2 \cdot delta + ave$ |

Table 4.1: *Number of data frames needed for different feature computations.*

## 4.2 Experiments using LVQ

Feature vectors were calculated only from the phone centers and the phone transitions are labeled into two classes, c and t, respectively. The presegmented speech was used so that the phoneme locations in the words were known. Two tests were performed, one test for separating phone centers from phone transitions (c/t-classification) and another test for classifying phone centers (c-classification). All the experiments were done with the public domain software package `LVQ_PAK` (Kohonen et al., 1996). The name of the corresponding program of the package is in the parentheses.

1. Codebook vectors are initialized with samples from teaching data so that all the classes contain an identical number of codebook vectors. All the candidates are tested against all the other samples and if the classification with the KNN-method[1] correct, then the candidate will be accepted as an initial codebook vector. (program `eveninit`)

2. The medians of the shortest distances among the codebook vector classes are computed. If the median of the class is greater than the average value, new codebook vectors are added to that class and if the median of the class is smaller than the average value, some vectors are deleted. (program `balance`)

3. The codebook vectors are trained iteratively with the OLVQ1-algorithm using 10000 iterations. Learning-rate initial values $\alpha_i(0) = 0.3$ for all classes $i$. (program `olvq1`)

4. The codebook vectors are trained iteratively with the LVQ1-algorithm by using learning-rate initial value $\alpha(0) = 0.2$ and decreasing it linearly during the learning

[1] In the KNN classification, K nearest neighbors are searched and the classification is done according to the majority voting.

process. Total number of iterations is 100 times the number of the codebook vectors. (program `lvq1`)

This kind of learning scheme was found to give the best results in phone center classification in (Mäntysalo, 1992; Mäntysalo et al., 1994). It was also used in c/t-classification there.

The results of the experiments are shown in the figures 4.4-4.17. Results of the c/t-classifications are in the leftmost diagram and results of the phone center classifications are in the rightmost diagram. The results of speaker 1 are in the upper diagrams and the results of speaker 2 are in the lower diagrams. Two parameters were investigated, the amount of averaging (*ave*) and the interval between two feature vectors in concatenation and delta computation (*delta*). Size of the codebook was 1000 vectors in all the experiments.

Figure 4.4 shows the effect of averaging successive mel-cepstra. In figure 4.5 two mel-cepstra are concatenated. The parameter *delta* is the interval between two features to be concatenated. The six plots in the same diagram correspond to the averagings of feature vectors to be concatenated. The topmost plot is the result of averaging six successive mel-cepstra and the lowermost plot is the result without averaging. The plots may cross each other, which may make the reading of the exact classification rates difficult. But it has to be emphased that the plots are meant to to show the trend of the recognition accuracy as the function of the delta parameter. So the shape of the c/t-classification plot is here more interesting than the absolute recognition accuracies. Putting all the six plots into the same diagram enables also to see the effect of the averaging. It is interesting to see how much the recognition accuracies differ using different amount of averaging.

Figure 4.6 shows the results of three concatenated mel-cepstra with averaging. Figures 4.7-4.13 show the results of using only delta features. Two different forms of the feature vector containing three concatenated simple deltas were experimented. In the overlapping version, the middle delta window overlaps with the first and last delta windows. In the case of regression delta, only overlapping version was experimented. Figures 4.14-4.17 show the results of using simple deltas catenated to one static feature.

## 4.3 Consideration about the c/t-classification

Different features can be compared according to the length of the time window, which they have been computed from or according to the dimension of the feature vector. It is clear that more information is available, if the time window of the feature vector computation is enlarged. In all of the experiments the training data and testing data was computed from different speech sets, but the vocabulary in the sets was the same. The danger of this is that if the time span for e.g. delta computation exceeds several phone durations, the delta features may adapt to the specific vocabulary and order of phonemes instead of single phonemes or phoneme pairs. Therefore one criterion in feature selection was not to use too many consequtive data frames for one feature. The duration of phones vary considerably, but 100 ms can be considered a crude average value. This is valid for short plosives /K/, /P/ and /T/ also, because there precedes always silence before the sound burst, and that silence is included to the duration of those phonemes. The drawback of this is in the phone center classification, because the preceding silence of a plosive sound is easily mixed with the normal silence, which was also one 'phoneme' class in the experiments. But this slight deterioration in the results is not very crucial, because the idea of these experiments

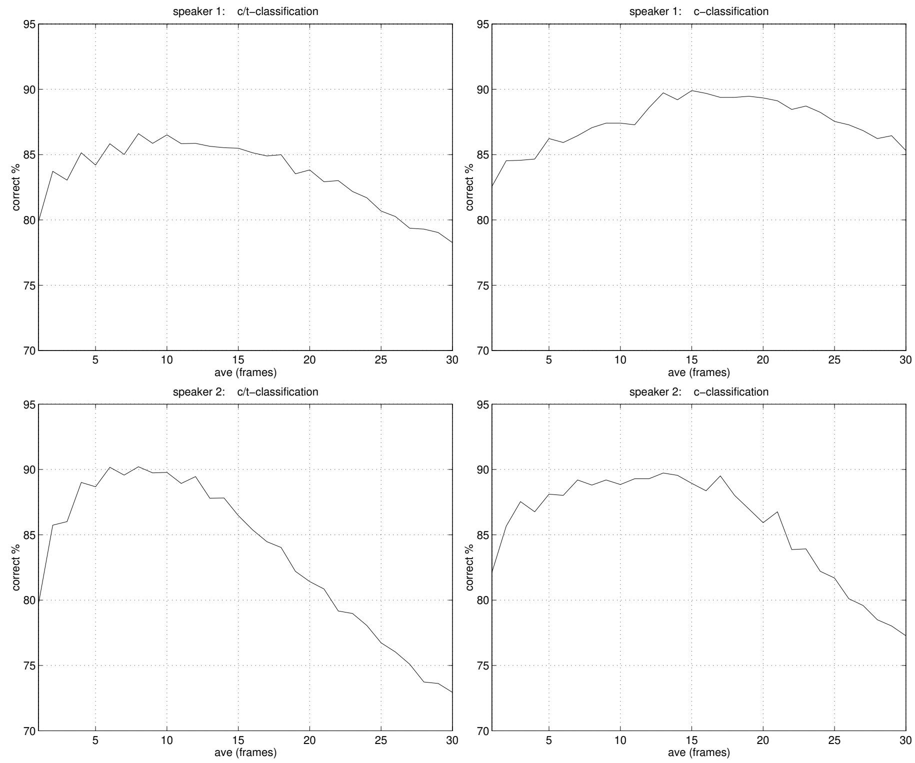


Figure 4.4: *Results using one static mel-cepstrum with averaging. Recognition accuracies are plotted as a function of the number of averaged mel-cepstra. The leftmost diagram shows the results of the phone center/phone transition -classification and the rightmost diagram shows the results of the phone center classification. The results of speaker 1 are in the topmost diagrams and the results of speaker 2 are in the lower diagrams.*

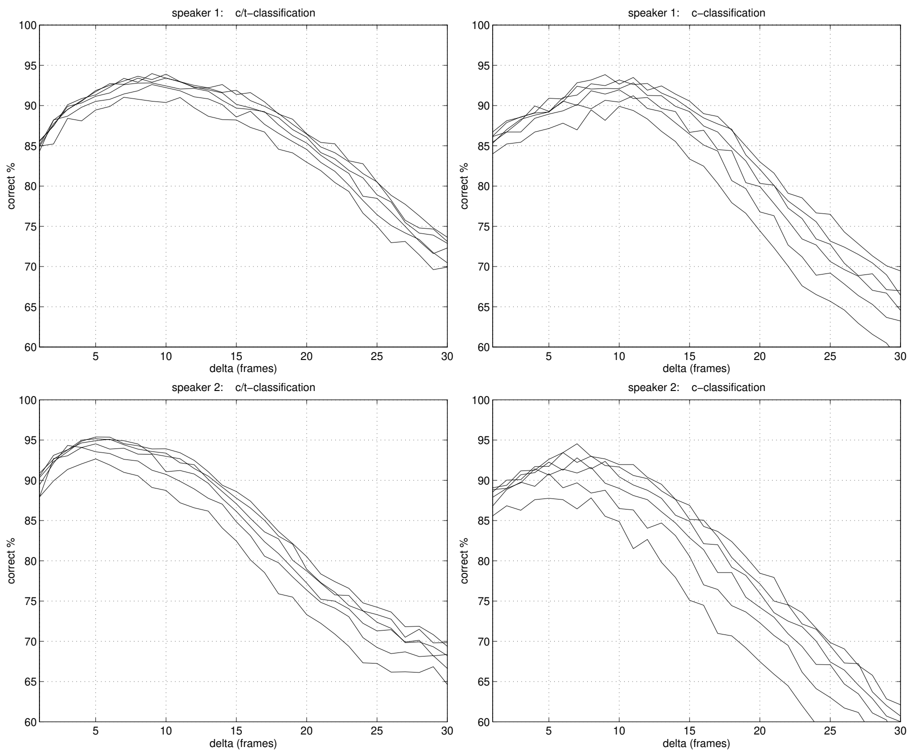


Figure 4.5: *Results using two concatenated static mel-cepstra. Six plots are the results using six different averagings (see the text for the explanation).*

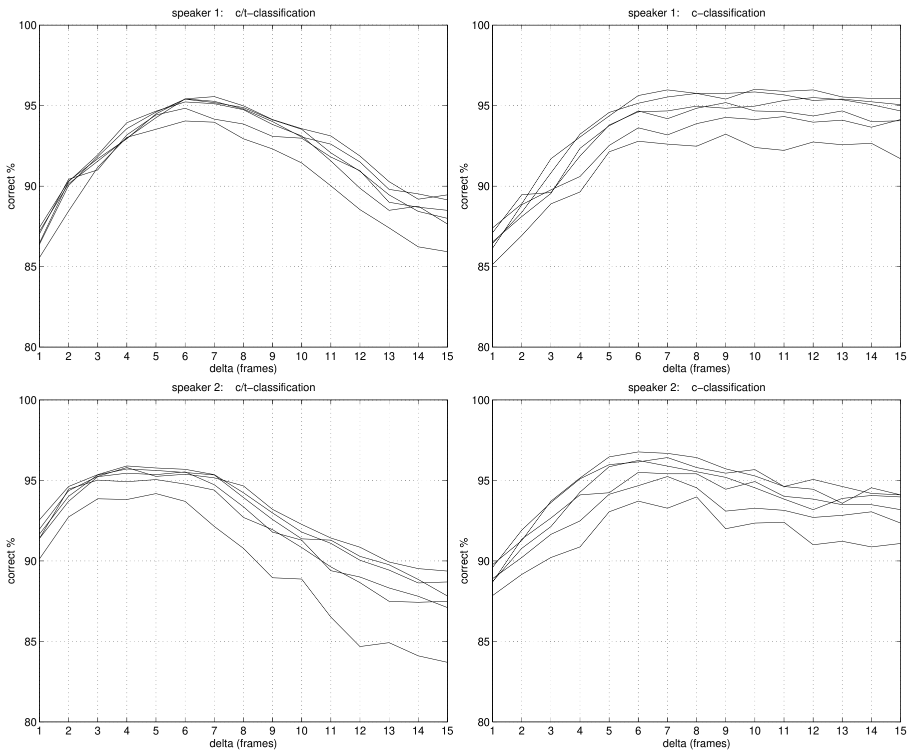


Figure 4.6: *Results using three concatenated static mel-cepstra. Parameter delta is the interval between two feature vectors to be concatenated. Six plots are the results using six different averagings (see the text for the explanation).*

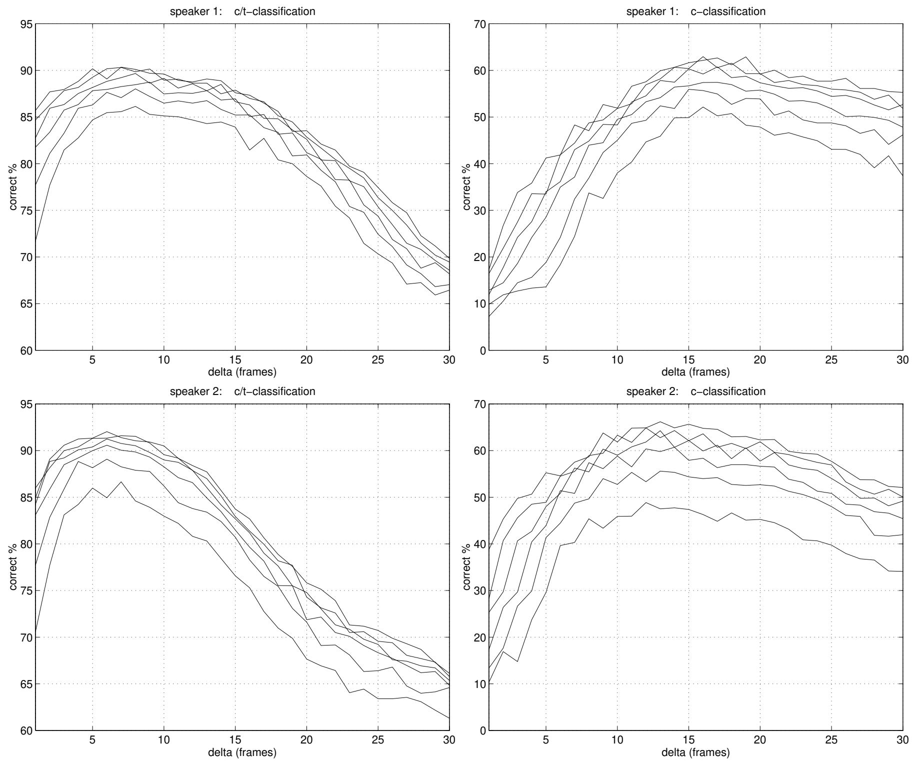


Figure 4.7: *Results using one delta mel-cepstrum. Six plots are the results using six different averagings (see the text for the explanation).*

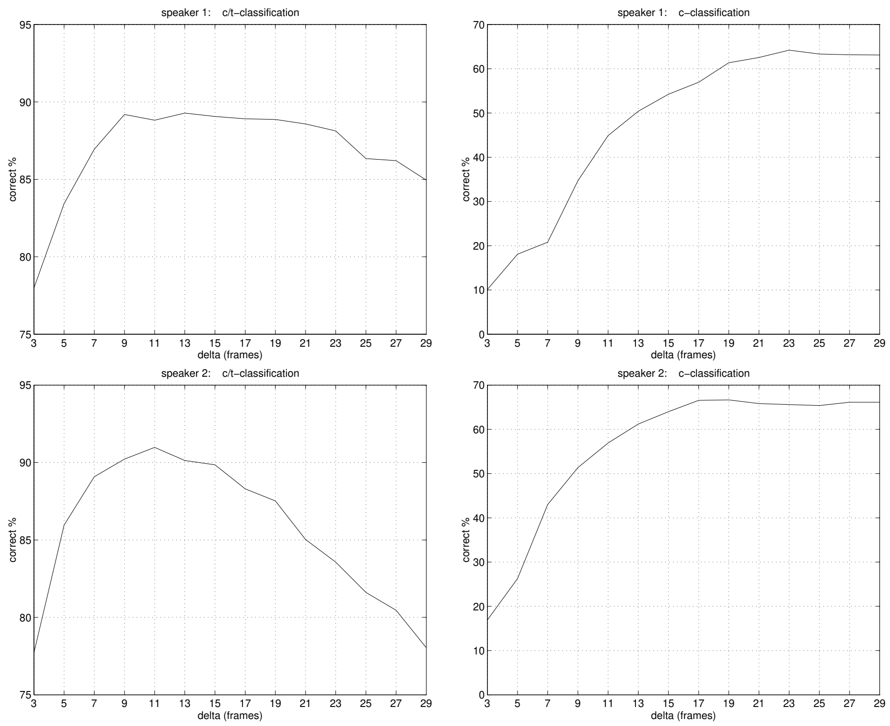


Figure 4.8: *Results using one regression delta mel-cepstrum.*

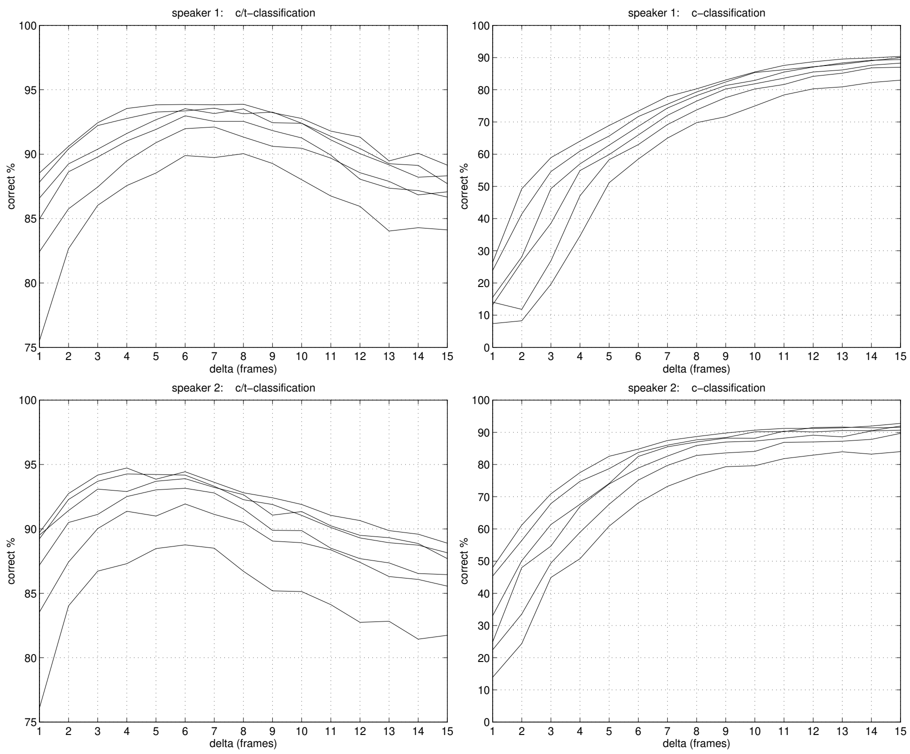


Figure 4.9: *Results using two concatenated delta mel-cepstra. Parameter delta is the interval of one delta computation. Six plots are the results using six different averagings (see the text for the explanation).*

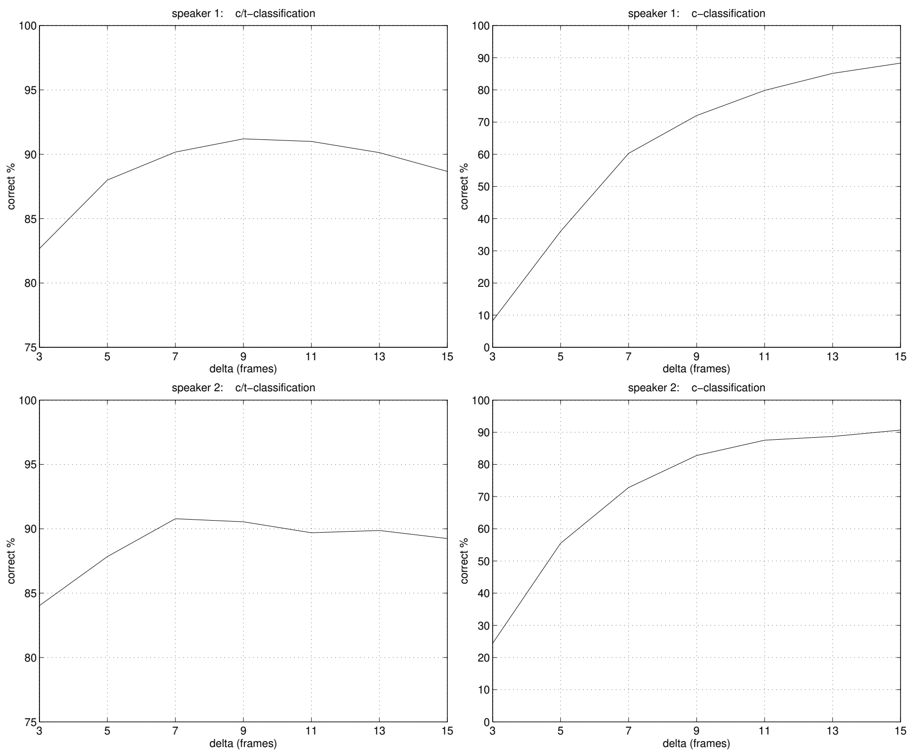


Figure 4.10: *Results using two concatenated regression delta mel-cepstra. Parameter delta is the interval of one delta computation.*

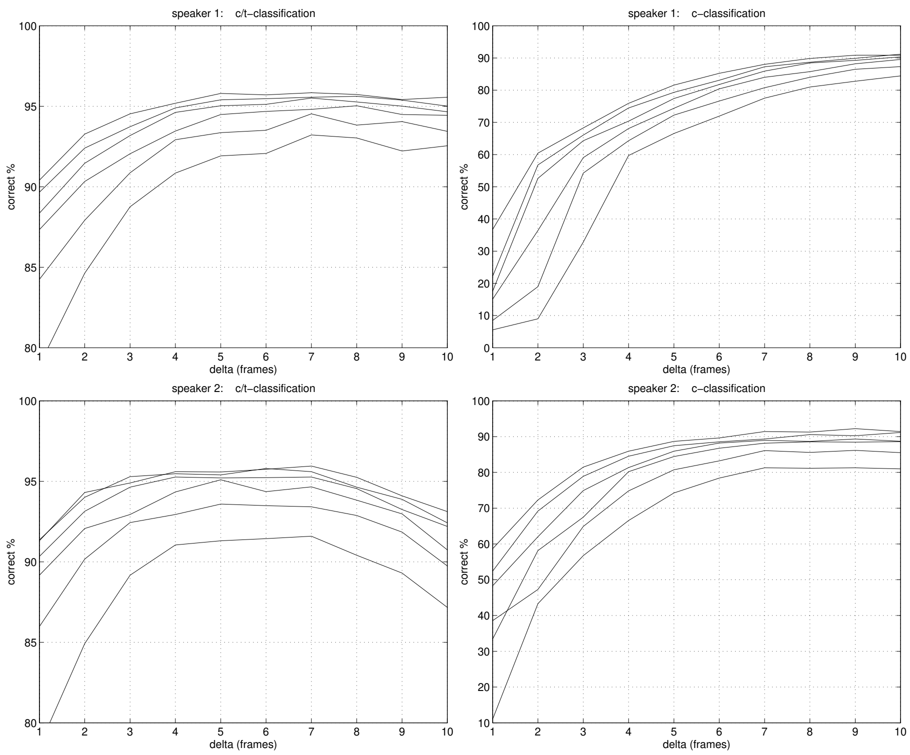


Figure 4.11: *Results using three concatenated nonoverlapping delta mel-cepstra. Parameter delta is the interval of one delta computation. Six plots are the results using six different averagings (see the text for the explanation).*

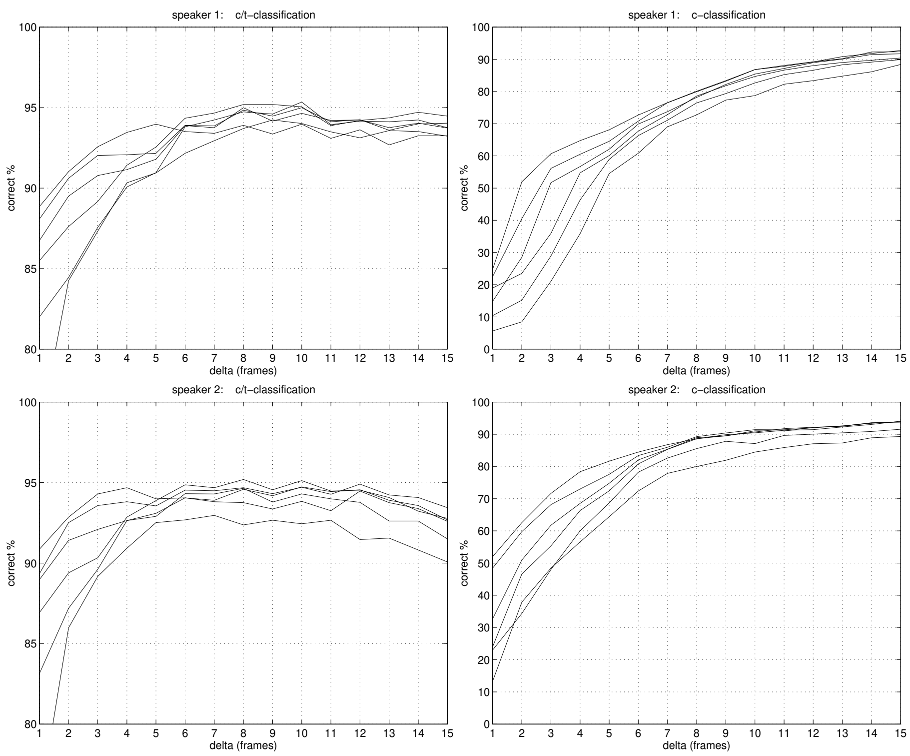


Figure 4.12: *Results using three concatenated overlapping delta mel-cepstra. Parameter delta is the interval of one delta computation. Six plots are the results using six different averagings (see the text for the explanation).*

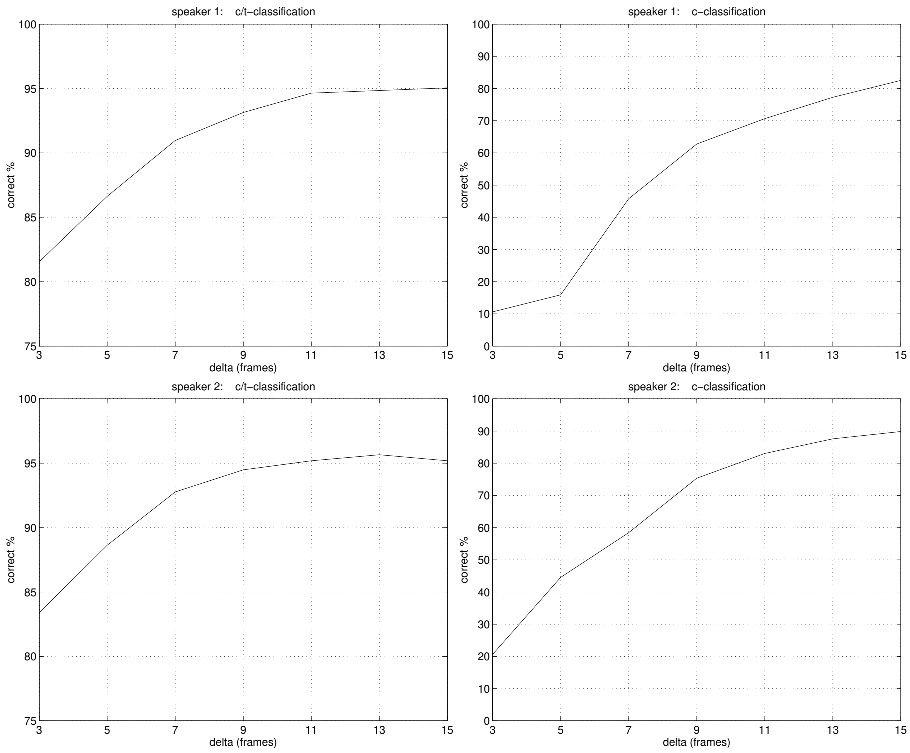


Figure 4.13: *Results using three concatenated overlapping regression delta mel-cepstra. Parameter delta is the interval of one delta computation.*

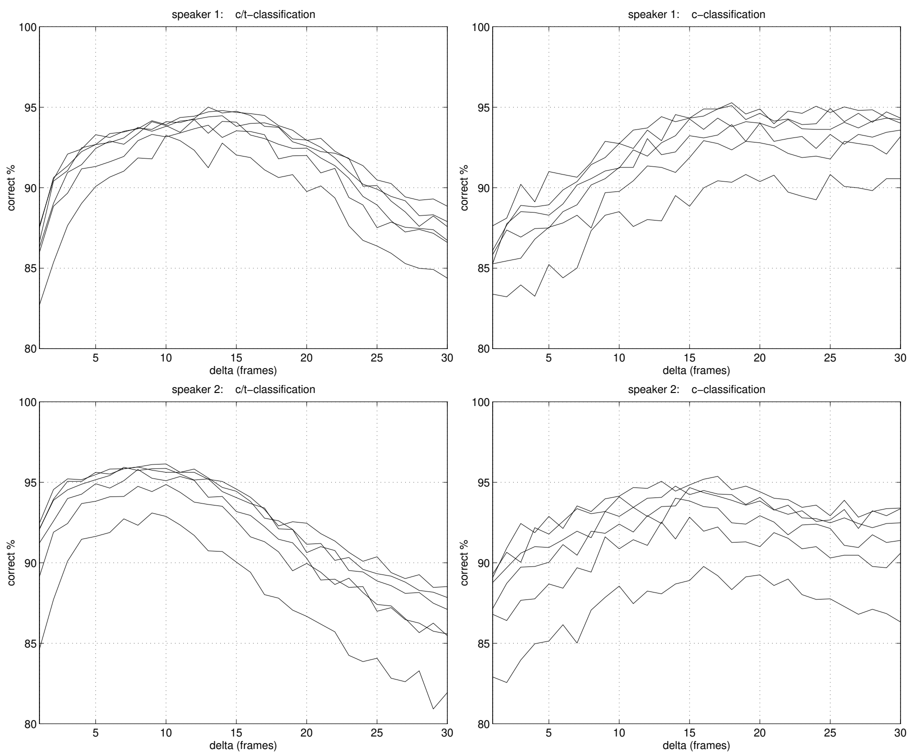


Figure 4.14: *Results using one static mel-cepstrum concatenated to one delta mel-cepstrum. Six plots are the results using six different averagings (see the text for the explanation).*

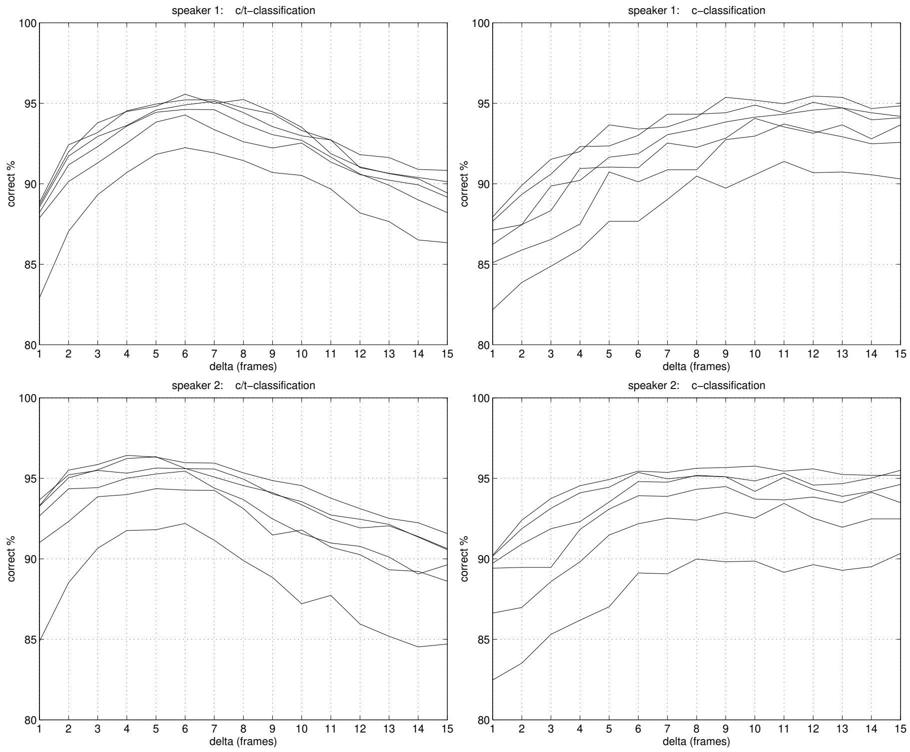


Figure 4.15: *Results using one static mel-cepstrum concatenated to two delta mel-cepstra. Parameter delta is the interval of one delta computation. Six plots are the results using six different averagings (see the text for the explanation).*

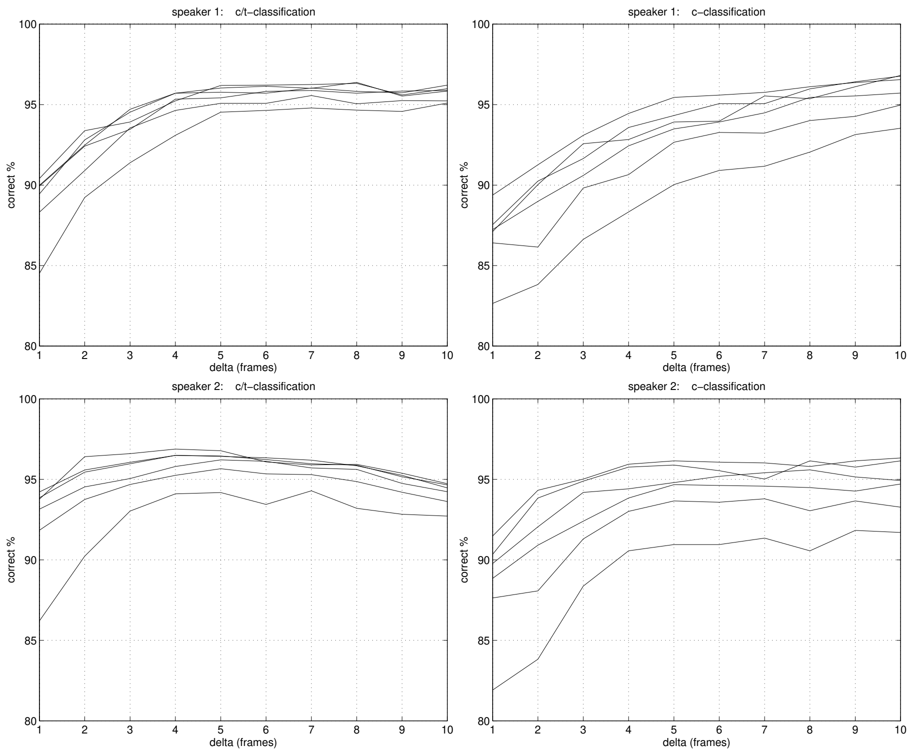


Figure 4.16: *Results using one static mel-cepstra concatenated to three nonoverlapping delta mel-cepstra. Parameter delta is the interval of one delta computation. Six plots are the results using six different averagings (see the text for the explanation).*

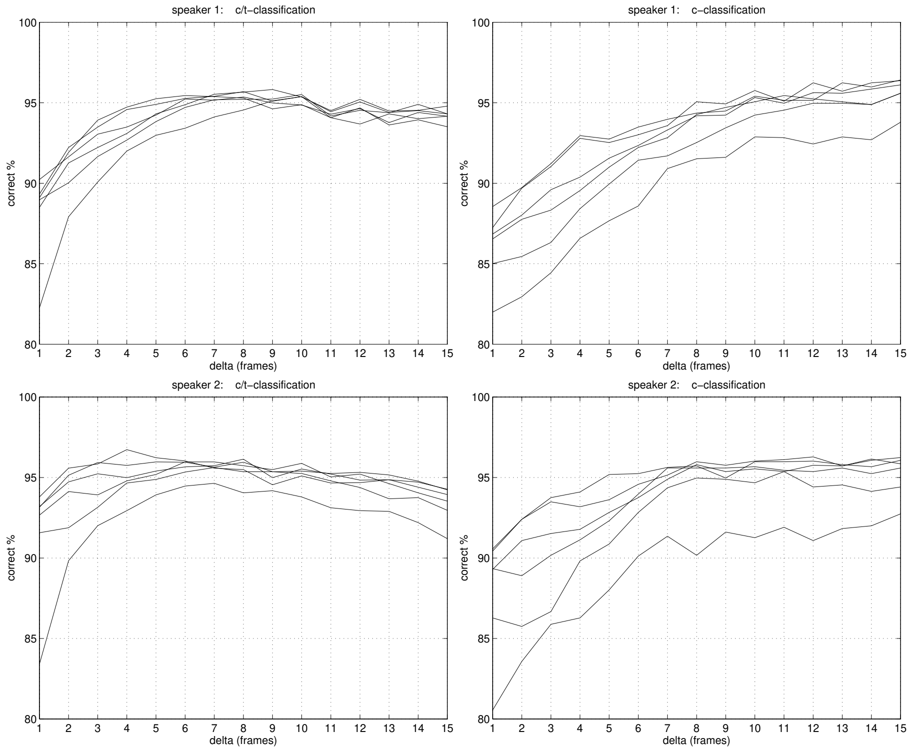


Figure 4.17: *Results using one static mel-cepstra concatenated to three overlapping delta mel-cepstra. Parameter delta is the interval of one delta computation. Six plots are the results using six different averagings (see the text for the explanation).*

was rather to compare different mel-cepstrum combinations than to get the best possible recognition rates.

In the c/t-classification, static features and delta features gave quite similar results, especially when averaged deltas were used. But in the phone center classification, static features were clearly better. The phone center classifications using delta features improved considerably, when one static feature vector was added to the pure delta feature.

As it can be seen in the recognition rates, the shapes of the accuracy plots are slightly different for different speakers. This is likely due to the different speaking rates of the speakers. Phone durations of the speaker 1 are longer than phone durations of the speaker 2. Delta feature is naturally by no means a time invariant feature. But in the normal dictation, the speaking rate can be expected to stay in tolerable limits. According to the results, the time span of six frames (48 ms) was found to be an appropriate choice for both speakers for both static feature concatenation and simple delta computation. For regression deltas the time intervals giving the best results were greater. Time intervals between seven and eleven frames (56 ms - 88 ms) gave the best results. But it has to be noted that if averaging is used, the time window of the feature computation increases. The recognition results of both concatenated static features and delta features improved in general along the amount of the averaging. Longer time span than six frames gave better results also when one static feature was concatenated to one simple delta (4.14).

If the results of phone center classifications are examined, it seems to be that longer time spans give better results. Nevertheless, 48 ms was chosen to be the delta value in all long time feature calculations. If one phone duration is considered to be 100 ms on an average, then two concatenated 48 ms deltas fit perfectly to that time. Better results could be obtained by choosing an individual delta for each feature and each speaker, but the limited training data sets limit for this. If the parameters are too well matched to the particular training and testing set combination, good results can be naturally obtained, but the generalizing ability is not necessarily so good.

It was expected that the delta features would grasp the formant movements. But because of the cosine transform following the spectrum computation in mel-cepstrum computation, it was thought that the formant movements would be obseved more clearly using only mel-spectrum. Therefore, also delta mel-spectrum was used in c/t-experiment. However, the results were slightly inferior compared to those of delta mel-cepstrum.

In section 3.3.4 the connection between delta computation and the correlation of the consequtive features was described. But because speech contains changes of different time scales, one fixed time span in delta computation is not necessarily the best choice. The feature is assumed to indicate better the changes of different time scales, if the delta computation is performed in different time scales. Linear regression (equation 3.13) is an example of this. All delta computations can be represented as

$$\mathbf{d}(t) = \sum_{n=N_1}^{N_2} c_n \mathbf{x}(t+n) \ . \tag{4.1}$$

In the case of linear regression, $N_1 = -N_2$ and the coefficient $c_n = n$. Figure 4.18 shows the coefficients $c_n$ as a weighting window of the feature stream. Naturally this kind of representation is not valid only for deltas, but it includes all kind of linear combinations of the vectors in a feature stream.

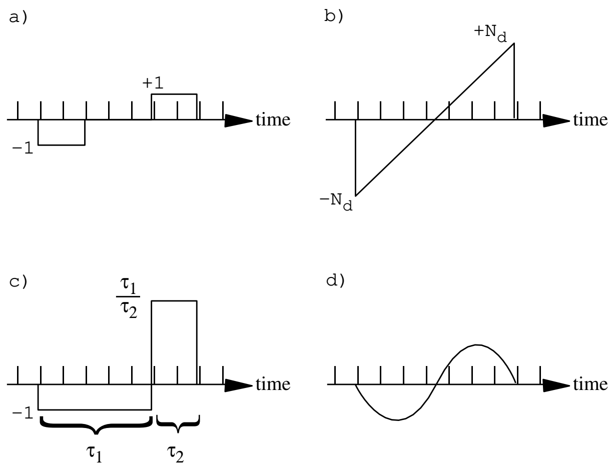


Figure 4.18: *Weighting of features in various delta computations. a) simple delta and b) linear regression.*

One feature vector having two concatenated deltas with different time spans was also experimented, but it didn't improve the obtained c/t -separation results.

# Chapter 5

# Experiments using HMM with one feature stream

This chapter contains the results of experiments using one input stream in the HMM. The SCHMM used in the experiments was based on (Kurimo, 1994). Each phoneme has a five-state SCHMM. After HMM classification, two measures are calculated. These are the error rate and the correct recognition rate. Levenshtein distance is used for the error rate. It determines the smallest number of additions, deletions and replacements of individual letters to transform one word into another. The correct recognition rate is the ratio of the number of correct phonemes in recognized word and the number of phonemes of its correct spelling. The additional phonemes in the recognized word do not deteriorate the recognition rate. This is useful, because it obtains information about the performance, if vocabulary or grammary based postprocessing part were included to the system. The errors in the results are very often due to the confusion between single and double phoneme. The recognized word may contain two successive same letters, although there should be only one in the correct spelling of the word.

## 5.1 Shape factor of the SOM unit kernel

The biggest difference between SCHMM and CDHMM is that in CDHMM both the mean vectors and the variances are separately estimated for all states. But in the SCHMM used in the experiments here, only mean vectors are estimated. This is done by using SOM. A fixed parameter, called a shape factor, is used instead of the variance.

The Gaussian probablity density function is given as

$$f(\mathbf{O}_t) = \frac{1}{(2\pi)^{D/2}|\mathbf{\Sigma}|^{1/2}} e^{-\frac{1}{2}(\mathbf{O}_t - \boldsymbol{\mu})^T \mathbf{\Sigma}^{-1} (\mathbf{O}_t - \boldsymbol{\mu})} \ . \tag{5.1}$$

If the covariance matrix is replaced by the diagonal matrix, the formula reduces to

$$f(\mathbf{O}_t) = \frac{1}{(2\pi)^{D/2} \prod_{d=2}^{D} \sigma_d} e^{-\sum_{d=1}^{D} \frac{(o_{td} - \mu_d)^2}{2\sigma_d^2}} \ . \tag{5.2}$$

The denominator in the term preceding the exponent part is a constant. The kernel of the SOM unit $k$ used in the SCHMM experiments was

$$f_k(\mathbf{O}_t) = e^{-\frac{\sum_{d=1}^{D}(o_{td}-\mu_{kd})^2}{h}} . \tag{5.3}$$

The kernel is thus the exponent part of the equation (5.2) with all variances $\sigma_d^2$ equal to $h/2$. Each code vector had the common $h$. A more sophisticated method would be to use an individual shape factor for every kernel (Hämäläinen, 1995).

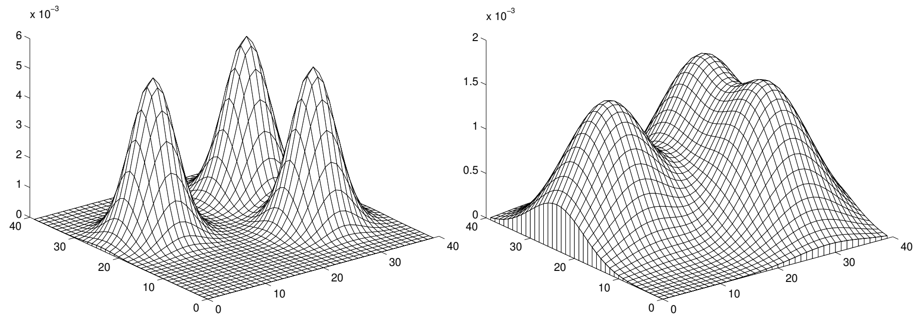

Figure 5.1: *Two density functions of the form* $\sum_{k=1}^{3} e^{-|\mathbf{x}-\boldsymbol{\mu}_k|^2/h}$. *In the left* $h = 20$*, and in the right* $h = 75$*.*

The effect of the shape factor is considered next. If the value of the shape factor is small, the tail of the probability function decreases rapidly. But if its value is large, the tail decreases slowly. So in order to get the same probability value, the observation vector must be closer to the codebook vector if $h$ is small than if $h$ is large. Figure 5.1 shows a two-dimensional density function with three kernels. This suggests that if it is wanted that a certain feature has impact only when it is very near to the codebook vector, the shape factor should be small. This could be used e.g. when the amount of the training vectors of the certain feature type is insufficient compared to other feature types. However, in the experiments done this was not the case, because all feature types were computed from the same raw data.

The effect of $h$ was investigated by using 5 epochs of Viterbi-training. Each round consisted of all the training data. The idea was to scan the $h$-axis with sparse grid in order to get some idea of the effect of the parameter. Figures 5.2 and 5.3 show the results of using static and delta features, respectively. Figure 5.4 shows the results when one static feature is concatenated to the delta features.

## 5.2 Consideration about the experiments

It is surprising that the recognition accuracies using the HMMs are better than the recognition accuracies in the phone center/phone transition -separation and in the phone center classification (section 4.2). The final phoneme recognition was expected to be more difficult task than the two previous tests. In the phone center classification, pure delta features gave fairly poor results, but now the results were comparable to those of static feautures.

In the c/t-classification (section 4.2), the results for two speakers were fairly similar, sometimes even better for the speaker 2, but now the results of the speaker 1 are clearly better. This is mostly evident in the case of delta features. The average square of the quantization error (the average of the squared distances between the feature vectors and their BMUs) was calculated for both speakers and in the case of delta features, the average

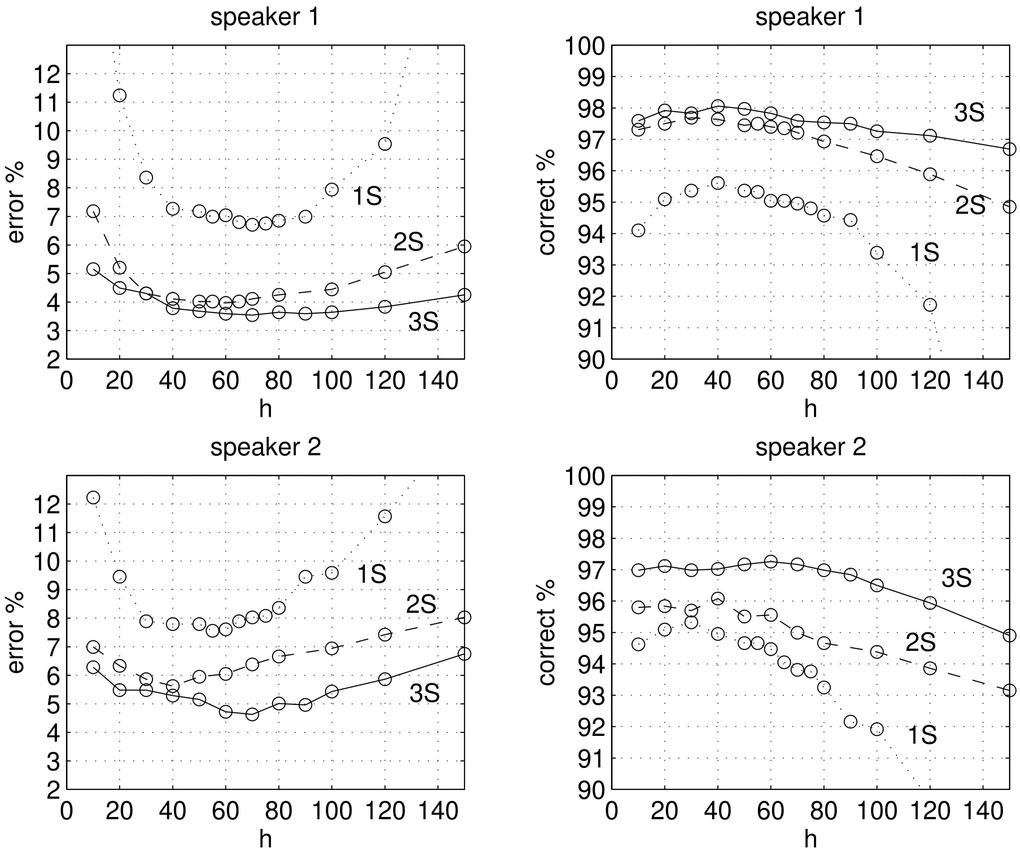


Figure 5.2: *Recognition results using one (1S), two (2S), and three (3S) concatenated static mel-cepstra.*

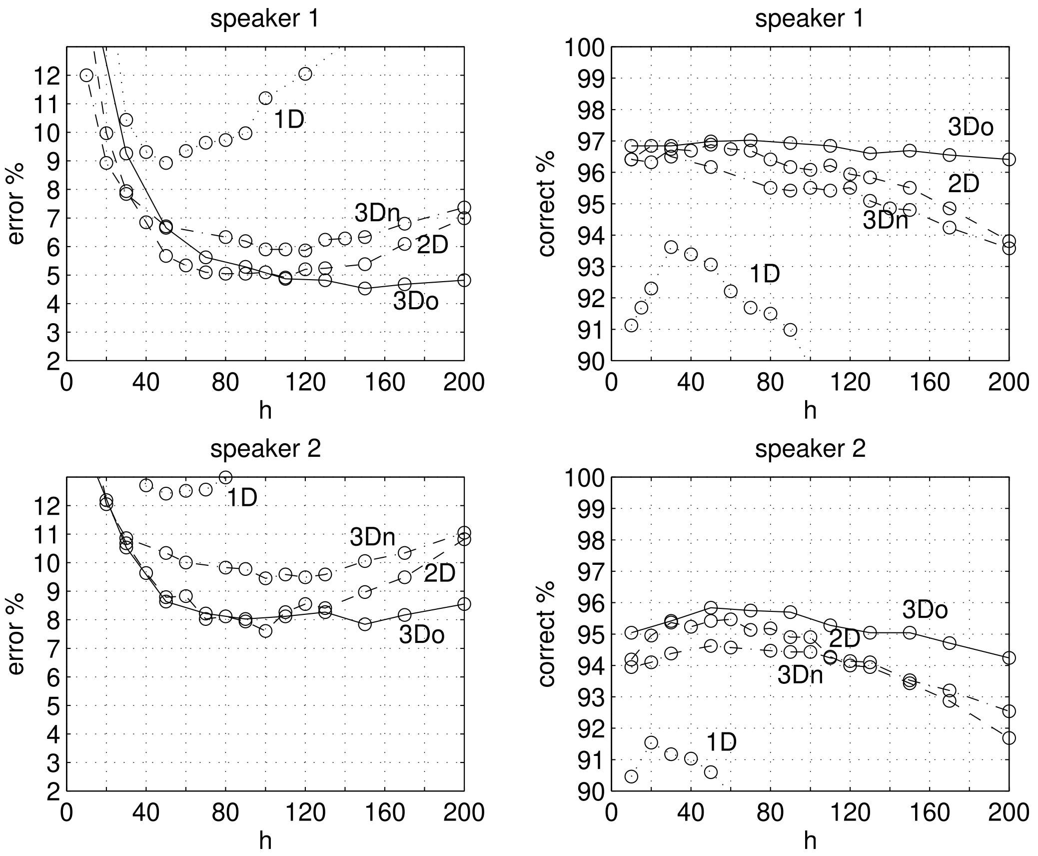


Figure 5.3: *Recognition results using one (1D), two (2D), and three (3D) concatenated delta mel-cepstra. n denotes nonoverlapping middle delta and o denotes overlapping middle delta. Deltas are simple differences.*

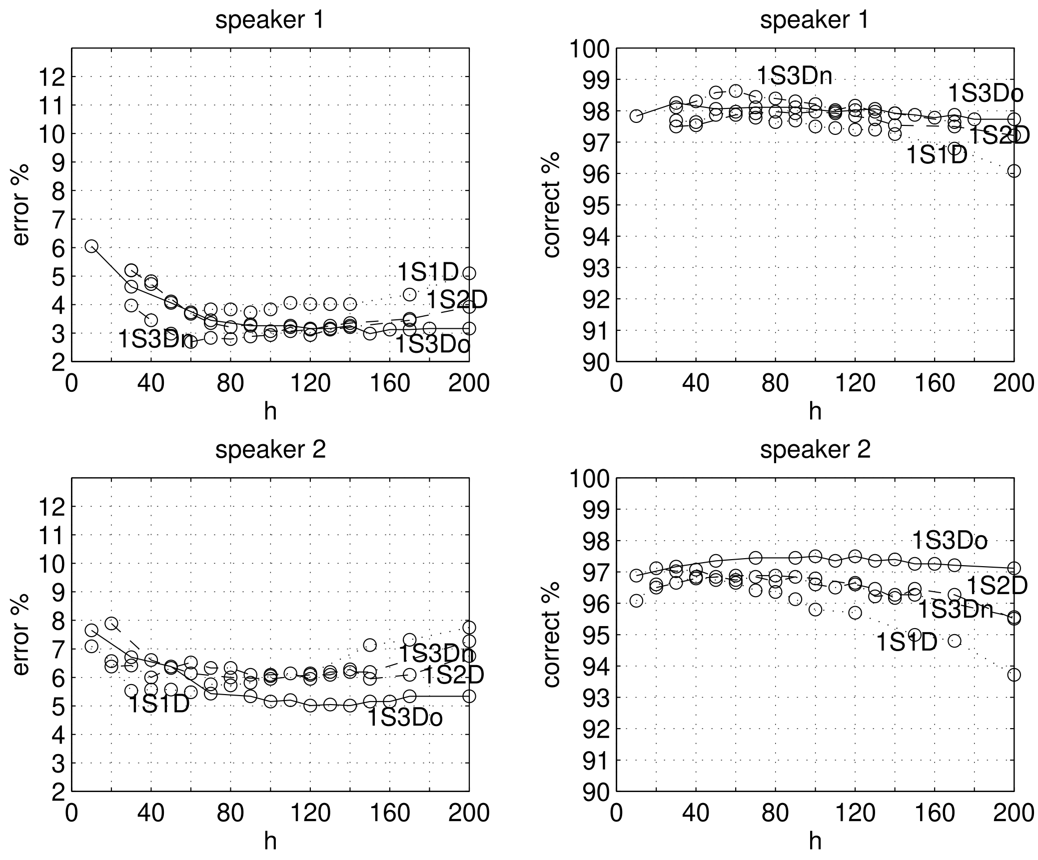


Figure 5.4: *Recognition results using one static mel-cepstrum (1S) concatenated to one (1D), two (2D), and three (3D) delta mel-cepstra. n denotes nonoverlapping middle delta and o denotes overlapping middle delta. Deltas are simple differences.*

distance of the speaker 1 was smaller. This suggests that the speech of the speaker 1 is more steady than the speech of the speaker 2 and gives thus an explanation for the differences in the recognition accuracies.

Quite surprisingly, two concatenated deltas gave better results than one static feature for the speaker 1 and for the speaker 2 the results were the same. In the case of static features, the results improved if more vectors were concatenated. But in the case of pure deltas, two concatenated deltas gave better results than three deltas if the middle delta window was nonoverlapping. Better results were obtained using overlapping middle delta.

In the case of concatenated static part and delta part, the results of the speaker 1 improved if more deltas were added. The results of the speaker 2 were not so clear. In general, the differences in the results for one speaker were much smaller having concatenated static and delta features than having only static features or delta features. But this is because in the pure static and pure delta feature diagrams one plot is the result using only one 20-dimensional feature vector. In the case of the combined static and delta feature diagram, all features are concatenated and the differences are not very significant. The improvement in the results is thus most evident if an additional feature vector is concatenated to the single static or single delta feature. This additional feature may be static or delta, in both cases the results improve. This shows the importance of the context.

The shapes of the one static feature plot (1S in figure 5.2) and the one delta feature plot (1D in figure 5.2) are the steepest compared to the other plots. So if only one single static or delta feature is used, the choice of the shape factor $h$ is more critical than if any of the context features are used.

It can be seen in the figures 5.2 - 5.4 that the plots become more flat as the dimension of the feature vectors increases. The average quantization error increases also, as can be seen in table 5.2. The shape factor $h$ giving the lowest error rates seem to be related to the quantization error. Most clearly this can be seen in the case of pure deltas in figure 5.3. Best values of $h$ to the features 1D, 2D, and 3Do are 50, 100, and 150, respectively. The ratio of these values 1:2:3 is the same for the average square of the quantization error of these features, as can be seen in the table 5.2.

| feature | speaker 1 | speaker 2 |
|---|---|---|
| 1S | 275.8 | 238.6 |
| 2S | 693.4 | 629.2 |
| 3S | 1133.8 | 1063.3 |
| 1D | 430.7 | 463.9 |
| 2D | 989.6 | 1084.8 |
| 3Do | 1563.7 | 1704.4 |
| 1S1D | 844.6 | 830.4 |
| 1S2D | 1375.4 | 1422.0 |
| 1S3Do | 1961.7 | 2057.6 |

Table 5.1: *The average square of the quantization error for the training set of two speakers. Codebook (SOM) size was 70 units for each phoneme. See the explanation for the features in figures 5.2 - 5.4.*

# Chapter 6

# Multiple feature streams

Speech recognition using multiple feature streams is described in figure 6.1. Each feature extraction method produces a feature vector as its output. Consequtive outputs from one feature extraction method form one feature stream. In the terms of HMM, feature vectors are observation vectors and feature streams are observation streams. Outputs of several feature streams at certain time instant form a feature vector set. This feature set is fed to the classifier.

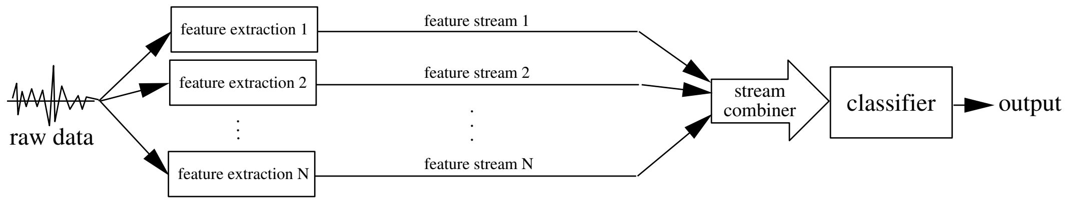


Figure 6.1: *Multiple feature streams.*

There are now two possibilities, how to feed the observation streams to HMM. One possibility is to concatenate all observation vectors from one time event to a single observation vector. HMM has then a single input stream. Another possibility is to keep different feature vectors in separate streams. In that case HMM has multiple parallel inputs. Multiple feature streams have been used e.g. in (Lee, 1988).

One of the main interests in this study was to investigate these two alternatives. Is it profitable to keep the two or more feature vectors separately or to concatenate them into a single feature vector? Because codebooks are used, this comparison belongs also to the study of multistage vector quantization.

## 6.1 Advantages of using multiple feature streams

When different feature vectors are kept separately, scaling problems can be avoided. If vectors with totally different numerical scales are concatenated (e.g. power and delta mel-cepstrum coefficients), information may be lost. If Euclidean distance is calculated between two concatenated vectors, the result will likely be dominated by the terms with large amplitudes and variances, even if the true information may lie in the parameters with small amplitudes. The influence of different variances among the elements in one feature vector can be eliminated by scaling the feature vector according to its variance.

But this doesn't finally eliminate the problem of scaling. Some feature vector elements may be more important than others and they would need greater weighting. The choose of optimal weighting is generally difficult task, but if a long feature vector can be divided into smaller parts, which are meaningful entitities, the search space of the weight combinations can be reduced and the weight optimization can be done more efficiently. How the division of the feature vector into smaller parts is done is crucial. If the feature vector consists of parts with different type or quality, the division can be done heuristically. Each feature element type is put to separate vectors. For example, if static and delta vectors are used only, it is natural to keep them separated. The task is harder, if a set of feature vectors is given with no a priori information about it, e.g. how the vectors have been produced and thus how they should be divided into parts. One possibility could be then to investigate the correlations of the vector elements and to try to divide one feature vector into subparts, which are as uncorrelated as possible.

Keeping the feature vectors separately means also smaller quantization error when the codebooks are used. This is due to the smaller number of elements in the feature vector (dimension of the vector). Multiple codebooks have also the advantage to represent larger amount of reference vectors. If each of the $S$ feature streams has a codebook of $N$ prototype vectors, the virtual size of the combined codebook is $N^S$ vectors.

## 6.2 Multiple observation streams in HMM

An observation of the stream $s$ at time instant $t$ is denoted as $\mathbf{O}_t^s$. Let $S$ be the number of the parallel streams. The output probability $b_i(\mathbf{O}_t)$ is then the joint probability of the $S$ probabilities. But if independecy of the streams is assumed, the output probability is simply the product

$$b_i(\mathbf{O}_t) = \prod_{s=1}^{S} b_{is}(\mathbf{O}_t^s) \tag{6.1}$$

If the probability is 'normalized' (like in (Gupta et al., 1987), where two streams were used) so that

$$b_i(\mathbf{O}_t) = \sqrt[S]{\prod_{s=1}^{S} b_{is}(\mathbf{O}_t^s)} \tag{6.2}$$

the logarithm of the probability will be

$$\begin{aligned} \log\{b_i(\mathbf{O}_t)\} &= \log\left\{\prod_{s=1}^{S} (b_{is}(\mathbf{O}_t^s))^{1/S}\right\} \\ &= \sum_{s=1}^{S} \frac{1}{S} \log\{b_{is}(\mathbf{O}_t^s)\} \end{aligned} \tag{6.3}$$

This can be generalized so that the weights of the streams are not equal.

$$\log\{b_i(\mathbf{O}_t)\} = \log\left\{\prod_{s=1}^{S} (b_{is}(\mathbf{O}_t^s))^{w_s}\right\}$$

$$= \sum_{s=1}^{S} w_s \log \{b_{is}(\mathbf{O}_t^s)\} \qquad (6.4)$$

To be consistent with the preceding formulas, the sum of the weights must be equal to one.

$$\sum_{s=1}^{S} w_s = 1 \qquad (6.5)$$

Now each stream has an individual weighting. The weight can then be determined according to the information content of the stream so that a more important stream has more weight than a less important stream.

The weighting can be continued from stream-dependent weights also further. In (Rogina and Waibel, 1994), state-dependent stream weights were introduced.

## 6.3 Experiments with multiple streams

The effect of the different stream weightings was experimented next. First the same stream weights were used in the HMM training and testing. Two stream combinations were experimented: a) one stream having single static mel-cepstra (1S) and another stream having single delta mel-cepstra (1D) and b) one stream having single static mel-cepstra (1S) and another stream having two concatenated delta mel-cepstra (2D). The results of the experiments are shown in figure 6.2. Also the combination of three concatenated delta mel-cepstra in one stream and one static mel-cepstrum in another stream was experimented, but the results were not better than using two concatenated deltas.

In the next experiment it was investigated how much the results change, when HMM training is done using some weights and in testing the weights are changed. This shows if it is profitable or not to tune the weights separately from HMM training. First the HMM training was done with equal stream weights (0.5, 0.5) and the testing was done with the different stream weight combinations. Then the HMM was trained several times with different stream weights but tested with equal stream weights (0.5, 0.5). Results are in figure 6.3. It can be seen, that the weight tuning after HMM training indeed has effect and the results are determined almost entirely according to the weights used in the testing phase. However, near the 1S-stream value of 0.5, the differences of results are smallest. This is at the same time near the minimum of the initial error plot (dotted line). It is interesting to note, that as the initial results get worse, the new stream weights give better results.

## 6.4 Attempts for finding optimal stream weights

Let us investigate two streams with static features in one stream and delta features in other stream. In the beginning and end of a phone delta features would be thought to be more important. But in the middle part of a phone, static features could be more useful.

Four different heuristic weightings in table 6.4 were experimented. They follow the form of figure 6.4. In the experiment the static feature was one mel-cepstrum and the delta feature was one delta mel-cepstrum. But as it can be seen in the figure 6.5, the results

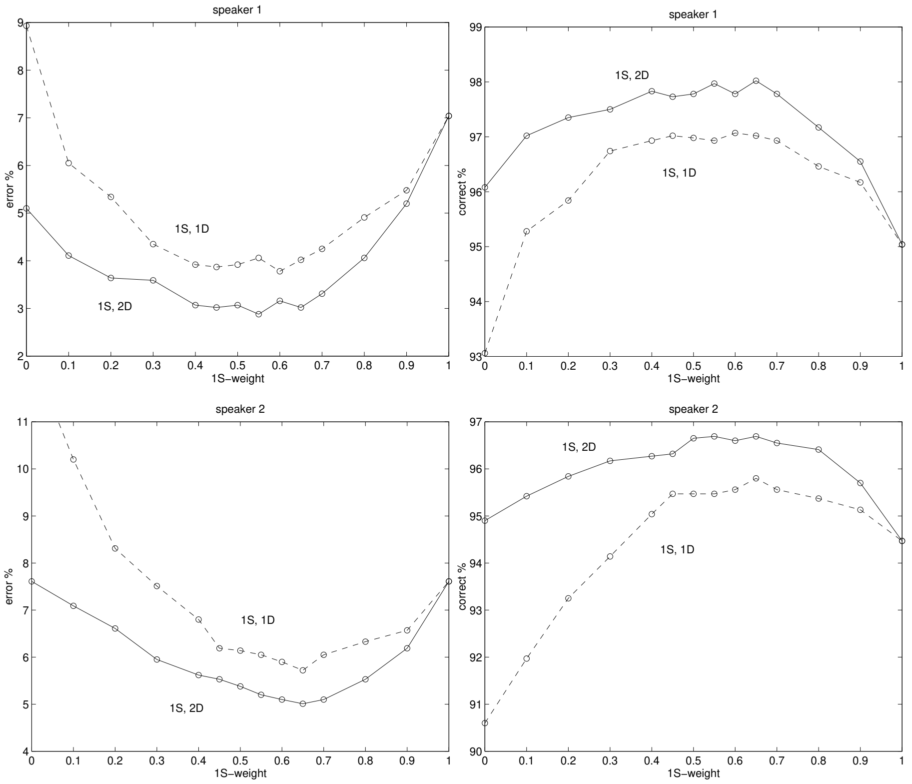


Figure 6.2: *Effect of the stream weighting in two cases. Dashed line shows the results of one static mel-cepstrum (1S) and one delta mel-cepstrum (1D). Solid line shows the results of one static mel-cepstrum (1S) and two concatenated delta mel-cepstra (D2). It can be seen that in both cases the best results are obtained by weighting static and delta streams almost equally.*

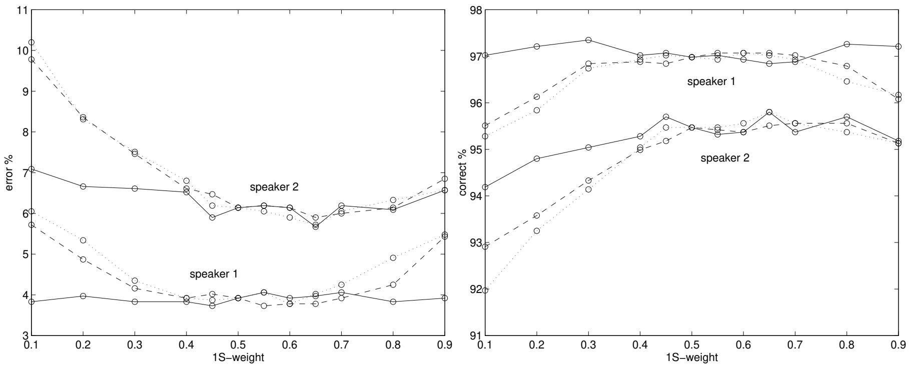


Figure 6.3: *Recognition results using one static mel-cepstrum stream (1S) and one delta mel-cepstrum stream (1D). Dashed line shows the result using equal stream weights (0.5, 0.5) in HMM training and different weights in testing. Solid line shows the result using different weights in the training and equal weights (.5,.5) in the testing. As a reference, dotted line shows the result of using same weights in the HMM training and testing.*

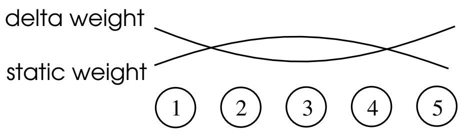


Figure 6.4: *A five-state phoneme model. Weighting favors delta stream in the beginning and end states and static stream in the middle state of the model.*

did not improve compared to the case where equal stream weights were used. However, in this experiment every phoneme model had the same weighting. It could be argued that different phonemes may require different weightings. This suggests to use state-dependent stream weighting (Rogina and Waibel, 1994).

| Weighting | | phoneme model state | | | | |
|---|---|---|---|---|---|---|
| | stream | 1 | 2 | 3 | 4 | 5 |
| w1 | static | .45 | .50 | .55 | .50 | .45 |
| | delta | .55 | .50 | .45 | .50 | .55 |
| w2 | static | .40 | .50 | .60 | .50 | .40 |
| | delta | .60 | .50 | .40 | .50 | .60 |
| w3 | static | .30 | .50 | .70 | .50 | .30 |
| | delta | .70 | .50 | .30 | .50 | .70 |
| w4 | static | .10 | .50 | .90 | .50 | .10 |
| | delta | .90 | .50 | .10 | .50 | .90 |

Table 6.1: *Experimented weightings in a two-stream HMM.*

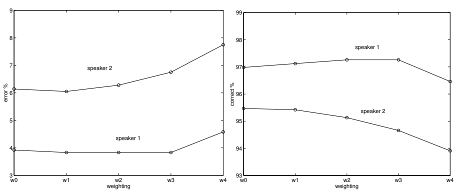

Figure 6.5: *The results of using heuristic weighting for one mel-cepstrum stream and one delta mel-cepstrum stream. Leftmost value is the result of equal stream weighting.*

There are now two basic principles, how to adapt the stream weights. The adaptation may rely on the ML-principle or alternatively it can be error-corrective tuning.

### 6.4.1 Weight tuning according to maximum likelihood

In the weight adaptation based on ML-principle, the contribution of the feature streams giving the highest probability to the final state probability is increased. The teaching procedure may start by setting equal stream weights to each state. After that the weights of the streams in one state are tuned according to their contribution to the state probability in the correct Viterbi path.

Weight updating may be done after each training word or alternatively it may take place after the whole training set has been introduced.

### 6.4.2 Error corrective weight tuning

In the HMM training, the correct word is known. This is the basis of the training. HMM path can be forced to contain only those states, whose output symbols are in the training word. But in error corrective (EC) tuning, also another HMM path is used. That is the path which has the highest probability of being produced by the training observation sequence without any constraints to have only the correct states. This situation corresponds to the use of HMM in the classification task. Let the first path be denoted as forced path and the latter one as unforced path.

In the EC training, the idea is to adapt stream weights so that the unforced path approaches the sequence of forced path. One form of this kind of training was introduced in (Rogina and Waibel, 1994).

Error corrective tuning was experimented as a batch solution. Figure 6.6 has an example of one case, where /E/ and /L/ have frequently been misclassified into /Y/. The plot shows a cumulative amount of correction for the stream weights of each state. Each time, when the unforced path has a state, which does not belong to the forced path, the probability of the forced path state is increased and the probability of the unforced path state is decreased. This is done by calculating the differences of the stream probabilities between the two states. The difference of the stream probabilities is added to the cumulative stream weight of the forced path state and subtracted from the cumulative stream weight of the unforced path. Those streams having the largest probability differences tune their weights most. But because of the batch version, the stream weights are not updated until the whole training set is introduced. The stream weights are then tuned according to the similar plot like in figure 6.6 using a suitable learning rate parameter. The scaling of the stream weights has to be done so that all stream weights are between 0 and 1 and the sum of the stream weights in each state is equal to 1.

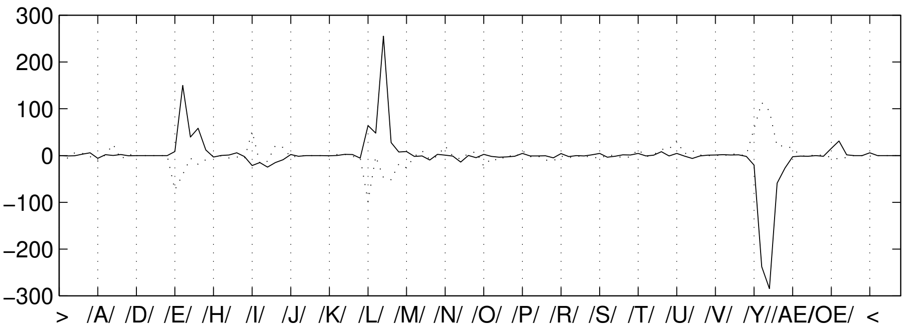


Figure 6.6: *Error corrective weight tuning. Each phoneme model has 5 states. Solid line indicates the cumulative weight correction for the mel-cepstrum stream and dotted line indicates the cumulative weight correction for the delta mel-cepstrum stream in each HMM state.*

However, despite the intriguing idea of the training, the recognition results did not improve. At least one case, when the results cannot improve is, when the stream probabilities of the wrong state in the unforced path are all higher than those of the forced path state. Then the observation probabilities $B$ of the HMM should be tuned instead of only stream weights. This suggests further studies on the topic, because the idea of setting automatically the stream weights and finding the discriminative features for each state is very tempting. Once the system knows the correct state sequence, it can then sort different feature streams according to their information content by giving different weights to them.

# Chapter 7

# The final test

A final test was performed using the speech of six speakers. Those two speakers, whose speech was used in the previous experiments of this study, were not involved in this test. Two parallel streams were used, one containing static 20-dimensional mel-cepstra and another containing concatenations of two mel-cepstra differences. The time interval of one mel-cepstrum difference was six frames (48 ms). Stream weights were equal in all states (0.50). The feature selection was done according to the results obtained in this study and the speeches of the six speakers had no influence on that.

Table 7 shows the improvement in the accuracy using the new system. The baseline system used five-state SCHMMs as phoneme models and one 21-dimensional mel-cepstrum as a feature vector (the 21st vector component was rms value). The improvement in the accuracy is thus due to the use of two concatenated mel-cepstrum differences. The obtained error reduction is 35 %.

| | baseline system | | new system (2 feature streams) | |
|---|---|---|---|---|
| speaker | error% | correct% | error% | correct% |
| 1 | 6.94 | 95.94 | 4.20 | 97.92 |
| 2 | 8.93 | 94.14 | 5.67 | 96.36 |
| 3 | 10.82 | 90.03 | 5.67 | 95.42 |
| 4 | 7.89 | 93.62 | 5.81 | 95.80 |
| 5 | 8.36 | 94.93 | 5.90 | 96.05 |
| 6 | 8.31 | 93.76 | 6.05 | 95.56 |
| average | 8.54 | 93.74 | 5.55 | 96.19 |

Table 7.1: The recognition accuracy using the baseline system and the new system. The test material of 350 Finnish words was used.

# Chapter 8

# Conclusion

## 8.1 Discussion

One main principle in feature selection is to try to get as general features as possible. This means that the chosen features should not be useful only to the particular data set. This can be partly avoided by having two separate data sets. One is used for training and another one for testing. But even this doesn't guarantee the generalizing ability. Although training set and testing set have been computed from different data in all experiments, the result of an experiment is always a combination of the particular training set and the particular testing set. Also the vocabulary of the sets have been the same. The danger of this is that if the time span for e.g. delta computation exceeds several phone durations, the delta features may adapt to the specific vocabulary and order of phonemes instead of single phones or phone pairs. Therefore one criterion in feature selection was not to use too many consequtive data frames for one feature. The duration of phones vary considerably, but crude average value can be considered to be 100 ms.

Different feature extraction methods can be considered as separate information sources, each of which producing an individual observation stream. One stream may be the main stream and another stream contain information about the context. For example having one stream of static mel-cepstra and one stream of concatenated delta mel-cepstra, the first stream can be considered as the main stream and the delta mel-cepstra stream as its context.

Parallel feature streams enable the development and testing of one feature separately from the effect of other streams. Combining multiple streams is then merely the finding of the proper stream weights. The effect of stream weighting could be achieved also by having only one concatenated feature vector and by weighting each vector component differently in distance calculations. However, by using the weighting in state probability computations, the tools of the probability theory can be used.

## 8.2 Conclusion

This study involved the developing of feature vectors in order to improve the recognition rate of the speech recognition system. Speech database consisted of 15 speakers and two speakers were chosen for this study. One of the speakers gave the best result according to the preliminary recognition test and one speaker represented the above average speaker.

Because the evaluation of the pattern recognition system is always determined by the combination of the training set and the testing set, too strict conclusions have been avoided to make from the results of the experiments. The results have been used mostly to see the effect of the parameters and not to get the best possible results for one speaker.

Several short time feature vector types were considered, and mel-cepstrum was chosen to be the building block of the long time feature vector computation. Combining several short time feature vectors into one long time feature vector brings the information about the context of the speech direct into the feature vector and the coarticulation effect can be taken into account. Several tests were done in order to find a feature, which could discriminate phone centers from phone transitions. 50 ms was found a good time span for computing deltas and concatenating static mel-cepstra.

Semi-continuous HMM was used in the final speech recognition task and SOM was used as a pdf approximation for each phoneme. Each SOM unit was a Gaussian kernel and the effect of the kernel width was experimented. The average squared quantization error of the feature vectors was found to be useful in choosing the kernel width.

Results using only delta features in the HMM were surprisingly good. Two concatenated delta mel-cepstra as a feature vector gave similar or even better results than one static mel-cepstrum vector. The improvement in the results was found to be most evident if one additional static or delta mel-cepstrum was concatenated to the single static or single delta mel-cepstrum. This shows the importance of the context in the feature vector.

The concept of multiple feature streams was also investigated. One of its benefits is that several different feature types of different magnitudes can be mixed without scaling of features. If different features are concatenated into a single feature vector, scaling is usually required. Good results were obtained having static mel-cepstra in one stream and delta mel-cepstra in other stream. Using both static and delta streams gave significantly better results than using only one of them. However, the results were quite similar when both vectors were concatenated into a single vector or they were fed parallelly into the HMM. But because the deltas were calculated from 50 ms time span, the magnitudes of them were close to the magnitudes of static features. More benefits could be expected if the streams would contain totally different features types.

Smallest error rates using one static mel-cepstrum and two concatenated delta mel-cepstrum in parallel streams obtained in the experiments were around 3 % for speaker 1 and around 5 % for speaker 2. In the test using speeches of six independent speakers, the average error rate was 5.55 % giving the error reduction of 35 % compared to the baseline system.

In order to take full advantage of multiple feature streams, a good stream weighting algorithm should be found. Some algorithms were developed and experimented in this study, but they did not improve the recognition results. This suggests further studies on the topic. Once a good algorithm is found, the system could then take many parallel feature streams and choose only those streams which have the most relevant information.

# Bibliography


Bahl, L., Brown, P., de Souza, P., and Mercer, R. (1988). A new algorithm for the estimation of hidden Markov model parameters. In *Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 493–496.

Baum, L. (1972). An inequality and associated maximization technique in statistical estimation of probabilistic functions of Markov chains. *Inequalities*, 3:1–8.

Bogert, B. P., Healy, M., and Tukey, J. (1963). The quefrency alanysis of time series of echoes: Cepstrum, pseudo-autocovariance, cross-cepstrum, and saphe cracking. In Rosenblatt, M., editor, *Proc. Symp. Time Series Analysis*, pages 209–243, New York. John Wiley & Sons, Inc.

Davis, S. B. and Mermelstein, P. (1980). Comparison of parametric representations for monosyllabic word recognition in continuously spoken sentences. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 28(4):357–366.

Forney, G. D. (1973). The Viterbi algorithm. In *Proceedings of the IEEE*, volume 6, pages 268–278.

Furui, S. (1986). Speaker-independent isolated word recognition using dynamic features of speech spectrum. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 34(1):52–59.

Gonzalez, R. C. and Woods, R. E. (1992). *Digital Image Processing.* Addison-Wesley.

Gupta, V., Lennig, M., and Mermelstein, P. (1987). Integration of acoustic information in a large vocabulary word recognizer. In *Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, volume 1, pages 697–700.

Hämäläinen, A. (1995). Self-organizing map and reduced kernel density estimation. Technical report, Rolf Nevanlinna Institute.

Huang, X. and Jack, M. (1989). Unified techniques for vector quantization and hidden Markov modeling using semi-continuous models. In *Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, volume 1, pages 639–642, New York.

Itakura, F., Singer, H., and Umezaki, T. (1990). Low bit quantization of the smoothed group delay spectrum for speech recognition. In *Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, volume 2, pages 761–764.

Itakura, F. and Umezaki, T. (1987). Distance measure for speech recognition based on the smoothed group delay spectrum. In *Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, volume 3, pages 1257–1260.

Jankowski Jr., C., Lippman, R., and Vo, H. (1995). A comparison of signal processing front ends for automatic word recognition. *IEEE Transactions on Speech and Signal Processing*, 3(4):286–292.

Jelinek, F. (1976). Continuous speech recognition by statistical methods. In *Proceedings of the IEEE*, volume 64, pages 532–536.

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

Juang, B.-H. and Rabiner, L. R. (1991). Hidden Markov models for speech recognition. *Technometrics*, 33(3):251–272.

Karjalainen, M. (1987). Puheen kuulemisen mallintaminen. Technical Report Akustiikan julkaisusarja, raportti 29, Helsinki University of Technology. (in Finnish).

Kirkpatric, S., Gelett Jr., C., and Vecchi, M. (1983). Optimization by simulated annealing. *Science*, 220(4598):671–680.

Kohonen, T. (1981). Automatic formation of topological maps of patterns in a self-organizing system. In *Proceedings of The Second Scandinavian Conference on Image Analysis*, pages 214–220, Helsinki, Finland.

Kohonen, T. (1986a). Dynamically expanding context, with application to the correction of symbol strings in recognition of continuous speech. In *Proceedings of the 8th International Conference on Pattern Recognition*, pages 1148–1151, Paris, France.

Kohonen, T. (1986b). Learning vector quantization for pattern recognition. Technical Report TKK-F-A601, Helsinki University of Technology.

Kohonen, T. (1988). The 'Neural' Phonetic Typewriter. *IEEE Computer*, 21(3):11–22.

Kohonen, T. (1991). The Hypermap Architecture. In Kohonen, T., Mäkisara, K., Simula, O., and Kangas, J., editors, *Artificial Neural Networks*, volume II, pages 1357–1360, Amsterdam, Netherlands. North-Holland.

Kohonen, T. (1995). *Self-Organizing Maps*. Springer-Verlag, Berlin-Heidelberg-New York.

Kohonen, T., Hynninen, J., Kangas, J., Laaksonen, J., and Torkkola, K. (1996). LVQ_PAK: The Learning Vector Quantization Program Package. Technical Report A31, Helsinki University of Technology, Laboratory of Computer and Information Science.

Koikkalainen, P. (1994). Progress with the tree-structured self-organizing map. In *Proceedings of ECAI 94, 11th European Conference on Artificial Intelligence*, pages 211–215.

Koikkalainen, P. and Oja, E. (1990). Self-organizing hierarchical feature maps. In *Proceedings of the International Joint Conference on Neural networks (IJCNN)*, volume 2, pages 297–284.

Kurimo, M. (1994). 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, Espoo, Finland.

Lee, K.-F. (1988). Large-vocabulary speaker-independent continuous speech recognition: The SPHINX system. Technical Report CS-88-148, Carnegie Mellon University.

Liu, L. (1995). Learning vector quantization and simulated annealing in automated feature selection for classification of voice disorders. Master's thesis, Helsinki University of Technology.

Mäntysalo, J. (1992). Some experiments on LVQ-based speech recognition with high-dimensional context vectors. Master's thesis, Helsinki University of Technology. (in Finnish).

Mäntysalo, J., Torkkola, K., and Kohonen, T. (1994). Mapping context dependent acoustic information into context independent form by LVQ. *Speech Communication*, 14(2):119–130.

Oppenheim, A. and Schafer, R. (1975). *Digital Signal Processing.* Prentice-Hall.

O'Shaughnessy, D. (1987). *Speech Communication.* Addison-Wesley.

Parzen (1962). An Estimation of a Probability Density Function and Mode. *The Annals of Mathematical Statistics*, 33:1065–1076.

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

Rabiner, L. and Schafer, R. (1978). *Digital Processing of Speech Signals.* Prentice-Hall.

Rabiner, L., Wilpon, J., and Juang, B. (1986). A segmental $K$-means training procedure for connected word recognition. *AT&T Technical Journal*, 64:21–40.

Rabiner, L., Wilpon, J., and Soong, F. (1988). High performance connected digit recognition using hidden Markov models. In *Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, volume 1, pages 119–122.

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

Rabiner, L. R. and Radar, C. M., editors (1972). *Digital Signal Processing*, chapter 2. IEEE Press.

Rogina, I. and Waibel, A. (1994). Learning state-dependent stream weights for multi-codebook HMM speech recognition systems. In *Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, volume 1, pages 217–220.

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