MEE-01-27

Speech Recognition using Hidden Markov Model

performance evaluation in noisy environment

Mikael Nilsson
Marcus Ejnarsson

Degree of Master of Science in Electrical Engineering
Supervisor: Mattias Dahl
Department of Telecommunications and Signal Processing
Blekinge Institute of Technology
March, 2002

Speech Recognition using Hidden Markov Model

performance evaluation in noisy environment

Mikael Nilsson
Marcus Ejnarsson

Department of Telecommunications and Signal Processing
Blekinge Institute of Technology
Ronneby, March 2002

Abstract

The purpose with this final master degree project was to develop a speech recognition tool, to make the technology accessible. The development includes an extensive study of hidden Markov model, which is currently the state of the art in the field of speech recognition. A speech recognizer is a complex machine developed with the purpose to understand human speech. In real life this speech recognition technology might be used to get a gain in traffic security or facilitate for people with functional disability. The technology can also be applied to many other areas. However in a real environment there exist disturbances that might influence the performance of the speech recognizer. The report includes an performance evaluation in different noise situations, in a car environment. The result shows that the recognition rate varies from 100%, in a noise free environment, to 75% in a more noisy environment.

MEE-01-27

Printed in Sweden
by Kaserntryckeriet AB
Karlskrona, 2002

Contact information:

Mikael Nilsson
Department of Telecommunications and Signal Processing
Blekinge Institute of Technology
SE-372 25 Ronneby
SWEDEN

Phone: +46 457 38 50 00
E-mail: Mikael.Nilsson@bth.se

Marcus Ejnarsson
Mobile phone: +46 709-621 321
E-mail: di97mej@student.bth.se

Contents

Chapter 1

Introduction

Speech recognition is a topic that is very useful in many applications and environments in our daily life. Generally speech recognizer is a machine which understand humans and their spoken word in some way and can act thereafter. It can be used, for example, in a car environment to voice control non critical operations, such as dialling a phone number. Another possible scenario is on-board navigation, presenting the driving route to the driver. Applying voice control the traffic safety will be increased.

A different aspect of speech recognition is to facilitate for people with functional disability or other kinds of handicap. To make their daily chores easier, voice control could be helpful. With their voice they could operate the light switch, turn off/on the coffee machine or operate some other domestic appliances. This leads to the discussion about intelligent homes where these operations can be made available for the common man as well as for handicapped.

With the information presented so far one question comes naturally: how is speech recognition done? To get a knowledge of how speech recognition problems can be approached today, a review of some research highlights will be presented.

The earliest attempts to devise systems for automatic speech recognition by machine were made in the 1950's, when various researchers tried to exploit the fundamental ideas of acoustic-phonetics. In 1952, at Bell Laboratories, Davis, Biddulph, and Balashek built a system for isolated digit recognition for a single speaker [12]. The system relied heavily on measuring spectral resonances during the vowel region of each digit. In 1959 another attempt was made by Forgie and Forgie, constructed at MIT Lincoln Laboratories. Ten vowels embedded in a /b/-vowel-/t/ format were recognized in a speaker independent manner [13]. In the 1970's speech recognition research achieved a number of significant milestones. First the area of isolated word or discrete utterance recognition became a viable and usable technology based on the fundamental studies by Velichko and

Zagoruyko in Russia [14], Sakoe and Chiba in Japan [15] and Itakura in the United States [16]. The Russian studies helped advance the use of pattern recognition ideas in speech recognition; the Japanese research showed how dynamic programming methods could be successfully applied; and Itakura's research showed how the ideas of linear predicting coding (LPC). At AT&T Bell Labs, began a series of experiments aimed at making speech recognition systems that were truly speaker independent [17]. They used a wide range of sophisticated clustering algorithms to determine the number of distinct patterns required to represent all variations of different words across a wide user population. In the 1980's a shift in technology from template-based approaches to statistical modelling methods, especially the hidden Markov model (HMM) approach [1].

The purpose with this master thesis is getting a deeper theoretical and practical understanding of a speech recognizer. The work started by examine a currently existing state of the art speech recognizer called hidden Markov toolkit (HTK)[1], used by many researchers. This is a complex distribution consisting of many modules implemented in the computer programming language C. All these modules was compiled and studied to understand each modules task. HTK is not easy to use, this because it is necessary to have theoretical understanding to get a fully functioning speech recognizer. Therefor it was desired to make a speech recognizer, based on HTK, accessible and simpler to use. A user interface was developed as a webpage, where training and testing files could be uploaded to evaluate the performance of the speech recognizer. The result was sent to the user via mail after completion.

With the experience from HTK, it was desired to have a system that could be easier to use. The webpage is limited because it is not possible to try different settings. This is why an implementation in the programming language Matlab has been done. This implementation is easy to use and can easily be extended with new features. This package is accessible for researchers and technological development in the field of speech recognition.

Applying this knowledge in a practical manner, the speech recognizer implemented in Matlab was used to simulated, as if, a speech recognizer was operating in a real environment. To do this simulation recordings in a car has been done to get real data.

In the future it could be possible to use this information to create a chip that could be used as a new interface to humans. For example it would be desired to get rid of all remote controls in the home and just tell the tv, stereo or any desired device what to do with the voice.

[1]More information can be found at htk.eng.cam.ac.uk

The outline of this thesis is as follows.

Chapter 2 - The Speech Signal
This chapter will discuss how the production and perception of speech is performed. Topics related to this chapter are *human vocal mechanism*, *speech representation* and *phonemes and phonetics*.

Chapter 3 - From Speech to Feature Vectors
In this chapter the fundamental signal processing applied to a speech recognizer. Some topics related to this chapter are *frame blocking and windowing*, *mel-cepstrum* and *delta and acceleration coefficients*.

Chapter 4 - Hidden Markov Model
Aspects of this chapter are theory and implementation of the set of statistical modelling techniques collectively referred to as hidden Markov modelling. Some topics related to this chapter are *observation densities*, *forward and backward algorithm*, *Viterbi algorithm* and *Baum-Welch*.

Chapter 5 - Speech Quality Assessment
This chapter presents different quality measurements applied to speech signals. Topics related to this chapter are *classical signal-to-noise ratio*, *segmental signal-to-noise ratio* and *Itakura distance*.

Chapter 6 - Practical Experimental Results
In this chapter the speech recognizer implemented in Matlab will be used. This to test the recognizer in different signal to noise ratios and with different noises.

Chapter 7 - Summary and Conclusions
This chapter will summarize the thesis and make suggestions to further work.

Chapter 2

The Speech Signal

As relevant background to the field of speech recognition, this chapter intend to discuss how the speech signal is produced and perceived by human beings. This is an essential subject that has to be considered before one can pursue and decide which approach to use for speech recognition.

Human communication is to be seen as a comprehensive diagram of the process from speech production to speech perception between the talker and listener, see Fig. 2.1.

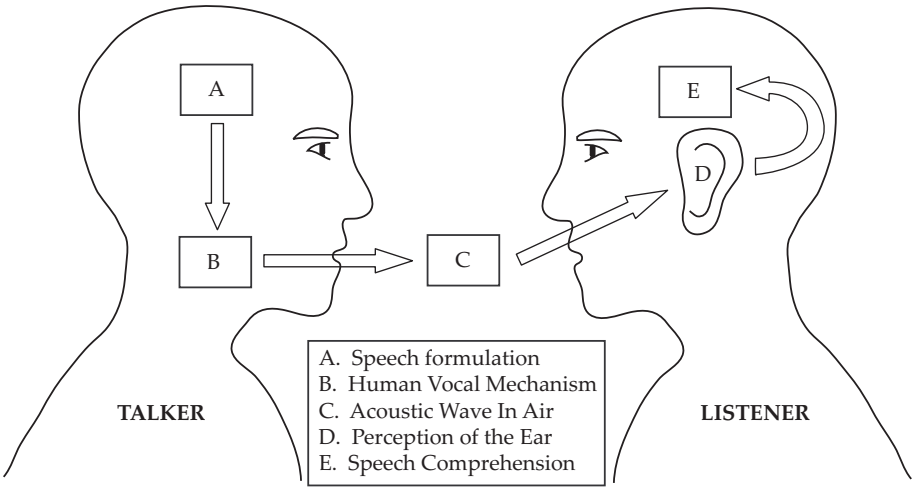

Figure 2.1: Schematic diagram of the speech production/perception process

Five different elements, A. Speech formulation, B. Human vocal mechanism, C. Acoustic air, D. Perception of the ear, E. Speech comprehension, will be examined more carefully in the following sections.

The first element (A. Speech formulation) is associated with the formulation of the speech signal in the talker's mind. This formulation is used by the human vocal mechanism (B. Human vocal mechanism) to produce the actual speech waveform. The waveform is transferred via the air (C. Acoustic air) to the listener. During this transfer the acoustic wave can be affected by external sources, for example noise, resulting in a more complex waveform. When the wave reaches the listener's hearing system (the ears) the listener percepts the waveform (D. Perception of the ear) and the listener's mind (E. Speech comprehension) starts processing this waveform to comprehend its content so the listener understands what the talker is trying to tell him or her.

One issue with speech recognition is to "simulate" how the listener process the speech produced by the talker. There are several actions taking place in the listeners head and hearing system during the process of speech signals. The perception process can be seen as the inverse of the speech production process. Worth mentioning is that the production and perception is highly a nonlinear process. This will be discussed further on in Ch. 3.

2.1 Speech Production

To be able to understand how the production of speech is performed one need to know how *the human's vocal mechanism* is constructed, see Fig. 2.2[1].

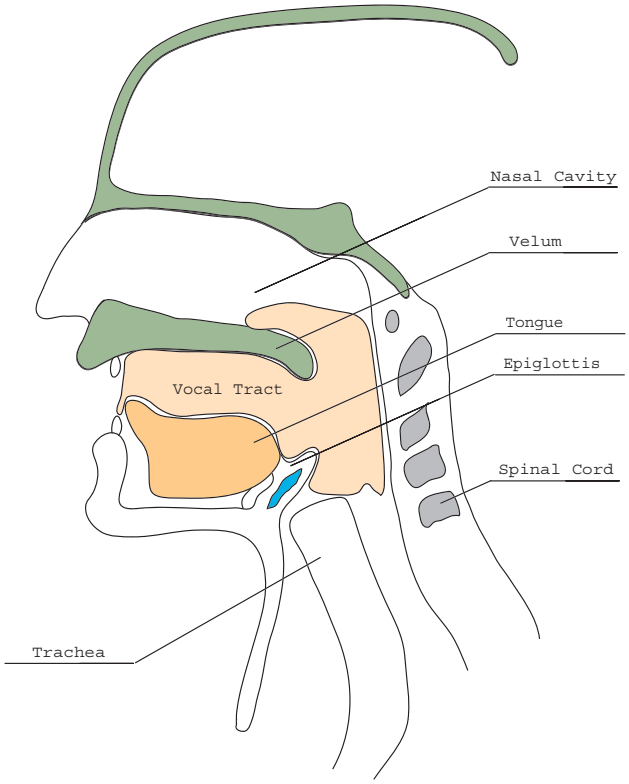

Figure 2.2: Human Vocal Mechanism

The most important parts of the human vocal mechanism are the *vocal tract* together with *nasal cavity*, which begins at the *velum*. The velum is a trapdoor-like mechanism that is used to formulate nasal sounds when needed. When the velum is lowered, the nasal cavity is coupled together with the vocal tract to formulate the desired speech signal. The cross-sectional area of the vocal tract is limited by the tongue, lips, jaw and velum and varies from 0-20 cm^2 [1].

When humans produce speech, air is expelled from the lungs through the *trachea*. The air flowing from the lungs causes the vocal cords to vibrate and by forming the vocal tract, lips, tongue, jaw and maybe using the nasal cavity, different sounds can be produced.

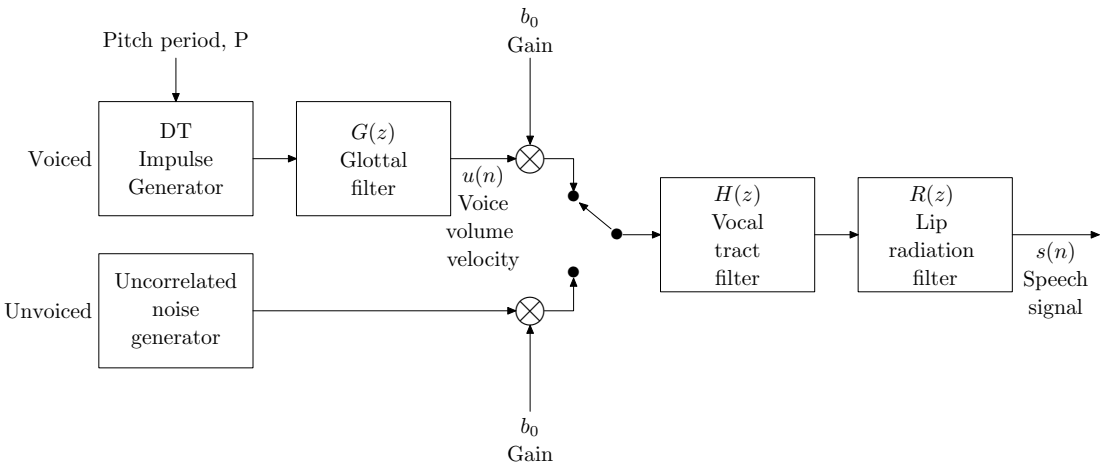

Figure 2.3: Discrete-Time Speech Production Model

Important parts of the *discrete-time speech production model*, in the field of speech recognition and signal processing, are: $u(n)$, gain b_0 and $H(z)$. The impulse generator acts like the lungs, exciting the glottal filter $G(z)$, resulting in $u(n)$. The $G(z)$ is to be regarded as the vocal cords in the human vocal mechanism[1]. The signal $u(n)$ can be seen as the excitation signal entering the vocal tract and the nasal cavity and is formed by exciting the vocal cords by air from the lungs. The gain b_0 is a factor that is related to the volume of the speech being produced. Larger gain b_0 gives louder speech and vice versa. The vocal tract filter $H(z)$ is a model over the vocal tract and the nasal cavity. The lip radiation filter $R(z)$ is a model of the formation of the human lips to produce different sounds.

2.2 Speech Representation

The speech signal and all its characteristics can be represented in two different domains, *the time* and *the frequency domain.*

A speech signal is a slowly time varying signal in the sense that, when examined over a short period of time (between 5 and 100 ms), its characteristics are short-time stationary. This is not the case if we look at a speech signal under a longer time perspective (approximately time T>0.5 s). In this case the signals characteristics are non-stationary, meaning that it changes to reflect the different sounds spoken by the talker.

To be able to use a speech signal and interpret its characteristics in a proper manner some kind of representation of the speech signal are preferred. The speech representation can exist in either the time or frequency domain, and in three

[1]Please recall Fig. 2.2

different ways[1]. These are a *three-state representation*, a *spectral representation* and the last representation is a *parameterization of the spectral activity*. These representations will be discussed in the following sections (2.2.1-2.2.3).

2.2.1 Three-state Representation

The three-state representation is one way to classify events in speech. The events of interest for the three-state representation are:

- Silence (S) - No speech is produced.
- Unvoiced (U) - Vocal cords are not vibrating, resulting in an aperiodic or random speech waveform.
- Voiced (V) - Vocal cords are tensed and vibrating periodically, resulting in a speech waveform that is quasi-periodic.

Quasi-periodic means that the speech waveform can be seen as periodic over a short-time period (5-100 ms) during which it is stationary.

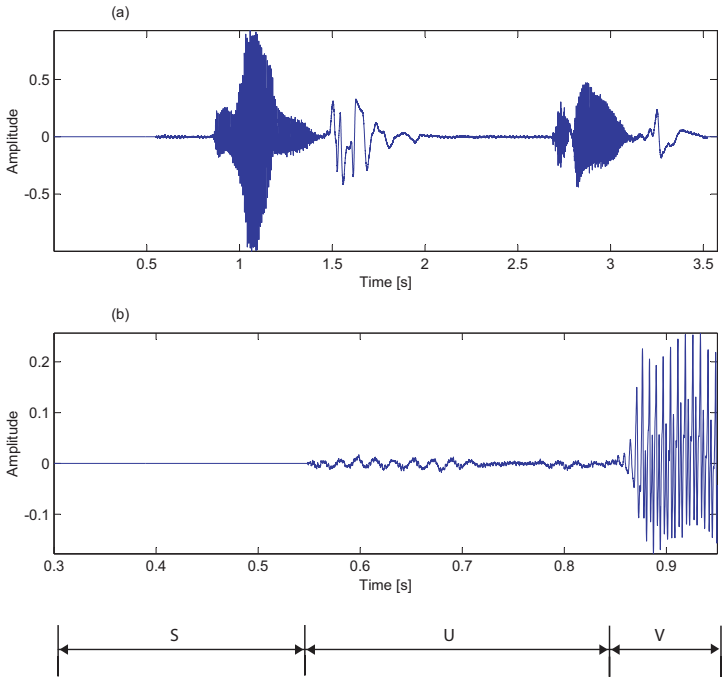

Figure 2.4: Three-state representation

The upper plot (a) contains the whole speech sequence and in the middle plot (b) a part of the upper plot (a) is reproduced by zooming an area of the whole speech sequence. At the bottom of Fig. 2.4 the segmentation into a three-state

representation, in relation to the different parts of the middle plot, is given.

The segmentation of the speech waveform into well-defined states is not straight forward. But this difficulty is not as a big problem as one can think. However, in speech recognition applications the boundaries between different states are not exactly defined and therefore non-crucial.

As complementary information to this type of representation it might be relevant to mention that these three states can be combined. These combinations result in three other types of excitation signals: *mixed, plosive and whisper*.

2.2.2 Spectral Representation

Spectral representation of speech intensity over time is very popular, and the most popular one is *the sound spectrogram*, see Fig. 2.5.

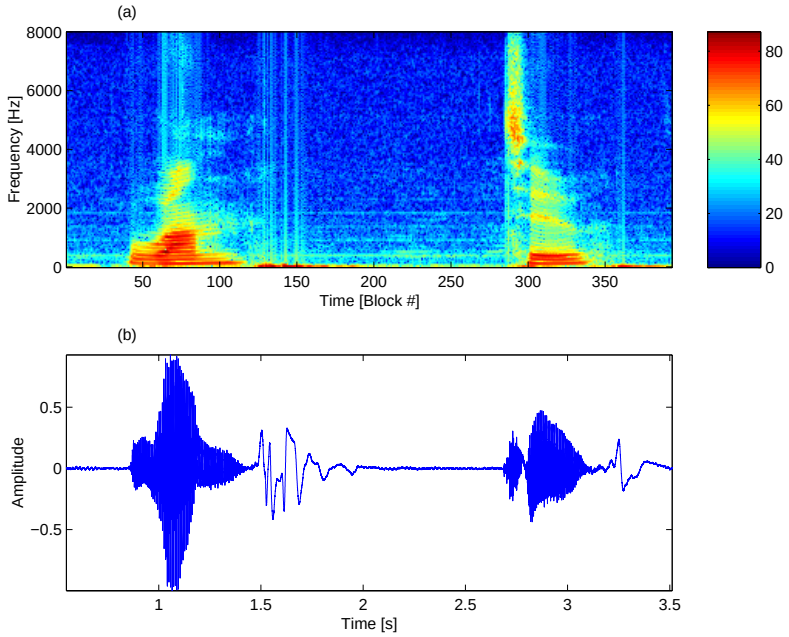

Figure 2.5: Spectrogram using Welch's Method (a) and speech amplitude (b)

Here the darkest (dark blue) parts represents the parts of the speech waveform[2] where no speech is produced and the lighter (red) parts represents intensity if speech is produced.

Fig. 2.5a shows a spectrogram in the frequency domain and in Fig. 2.5b the speech waveform is given in the time domain. For the spectrogram Welch's

[2]Please recall Fig. 2.5b

method is used, which uses averaging modified periodograms [3]. Parameters used in this method are blocksize $K = 320$, window type Hamming with 62.5% overlap resulting in blocks of 20 ms with a distance of 6.25 ms between blocks.

2.2.3 Parameterization of the Spectral Activity

When speech is produced in the sense of a time-varying signal, its characteristics can be represented via a parameterization of the spectral activity. This representation is based on the model of speech production [see Sec. 2.1].

The human vocal tract can (roughly) be described as a tube excited by air either at the end or at a point along the tube. From acoustic theory it is known that the transfer function of the energy from the excitation source to the output can be described in terms of natural frequencies or resonances of the tube, more known as *formants*. Formants represent the frequencies that pass the most acoustic energy from the source to the output. This representation is highly efficient, but is more of theoretical than practical interest. This because it is difficult to estimate the formant frequencies in low-level speech reliably and defining the formants for unvoiced (U) and silent (S) regions.

2.3 Phonemics and Phonetics

As discussed earlier in this chapter, the speech production begins in the humans mind, when he or she forms a thought that is to be produced and transferred to the listener. After having formed the desired thought, he or she constructs a phrase/sentence by choosing a collection of finite mutually exclusive sounds. The basic theoretical unit for describing how to bring linguistic meaning to the formed speech, in the mind, is called *phonemes*.

Phonemes can be seen as a way of how to represent the different parts in a speech waveform, produced via the human vocal mechanism[3] and divided into continuant (stationary) or non-continuant parts, see Fig. 2.6.

A phoneme is continuant if the speech sound is produced when the vocal tract is in a steady-state. In opposite of this state, the phoneme is non-continuant when the vocal tract changes its characteristics during the production of speech. For example if the area in the vocal tract changes by opening and closing the mouth or moving your tongue in different states, the phoneme describing the speech produced is non-continuant.
Phonemes can be grouped based on the properties of either the time waveform or frequency characteristics and classified in different sounds produced by the human

[3]Please recall Fig. 2.2

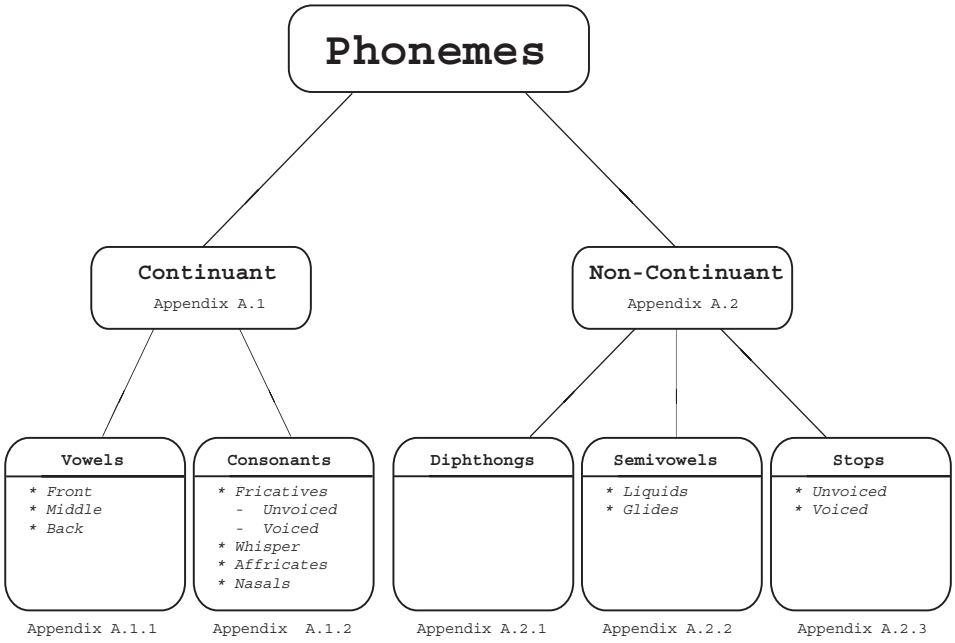

Figure 2.6: Phoneme classification

vocal tract. The classification, may also be seen as a division of the sections in Appendix A, see Fig. 2.6.

2.4 Summary

The purpose with this chapter has been to present an overview of the fundamentals of speech science and to provide sufficient background to pursue applications of digital signal processing to speech. The speech is produced through the careful movement of the vocal-tract articulators in response to an excitation signal. The excitation signal may be periodic at the glottis, or noise-like due to a major construction along the vocal tract, or a combination. Phonemes, can be classified in terms of the place of articulation, manner of articulation, or a spectral characterization. Each phoneme has a distinct set of features that may or may not require articulator movement for proper sound production. This knowledge is useful, in the field of speech recognition, to label the (synthetic?) speech waveform, being analyzed, in a linguistic sense. Further, the speech perception process, by the human hearing system, might be useful to model appropriately. The main thing to model is the nonlinear character of the human hearing system. A more detailed treatment of this subject, together with others, is found in the following chapter.

Chapter 3

From Speech To Feature Vectors

This chapter describes how to extract information from a speech signal, which means creating feature vectors from the speech signal. A wide range of possibilities exist for parametrically representing a speech signal and its content. The main steps for extracting information are *preprocessing*, *frame blocking and windowing*, *feature extraction* and *postprocessing*, see Fig. 3.1.

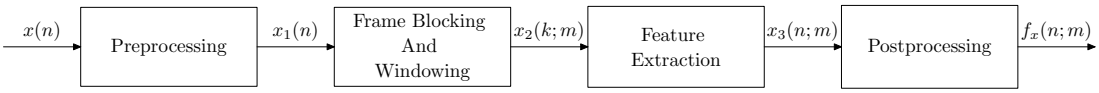

Figure 3.1: Main steps in Feature Extraction

From a correct bandlimited and sampled speech signal, $x(n)$, one will finally yield the feature vectors $f_x(n;m)$[1]. The notation $f_x(n;m)$, where $m = 0, 1, \ldots, M-1$ and $n = 0, 1, \ldots, N-1$, means M vectors, each of size N. In the following sections the steps in Fig. 3.1 will be explained.

3.1 Preprocessing

This step is the first step to create feature vectors. The objective in the preprocessing is to modify the speech signal, $x(n)$, so that it will be "more suitable" for the feature extraction analysis. The preprocessing operations noise cancelling, preemphasis and voice activation detection can be seen in Fig. 3.2.

The first thing to consider is if the speech, $x(n)$, is corrupted by some noise, $d(n)$, for example an additive disturbance $x(n) = s(n) + d(n)$, where $s(n)$ is the clean speech signal. There are several approaches to perform noise reduction on a noisy speech signal. Two commonly used noise reduction algorithms in the field of speech recognition context is *spectral subtraction* and *adaptive noise*

[1]Note that the index n in $f_x(n;m)$ is a vector index and not a sample index as in $x(n)$

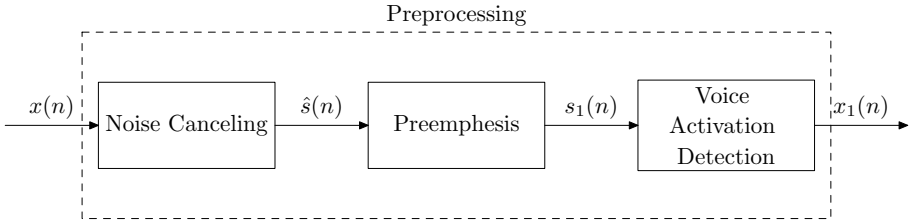

Figure 3.2: Steps in Preprocessing

cancellation [2]. A low *signal to noise ratio* (SNR) decrease the performance of the recognizer in a real environment. Some changes to make the speech recognizer more noise robust will be presented later. Note that the order of the operations might be reordered for some tasks. For example the noise reduction algorithm, *spectral subtraction*, is better placed last in the chain (it needs the voice activation detection).

3.1.1 Preemphasis

The preemphasizer is used to spectrally flatten the speech signal. This is usually done by a highpass filter. The most commonly used filter for this step is the FIR filter described below:

$$H(z) = 1 - 0.95z^{-1} \tag{3.1}$$

The filter response for this FIR filter can be seen in Fig. 3.3. The filter in the time domain will be $h(n) = \{\underset{\uparrow}{1}, -0.95\}$ and the filtering in the time domain will give the preemphasized signal $s_1(n)$:

$$s_1(n) = \sum_{k=0}^{M-1} h(k)\hat{s}(n-k) \tag{3.2}$$

3.1.2 Voice Activation Detection (VAD)

The problem of locating the endpoints of an utterance in a speech signal is a major problem for the speech recognizer. An inaccurate endpoint detection will decrease the performance of the speech recognizer. The problem of detecting endpoints seem to be relatively trivial, but it has been found to be very difficult in practice. Only when a fair SNR is given, the task is made easier. Some commonly used measurements for finding speech are short-term energy estimate E_{s_1}, or short-term power estimate P_{s_1}, and short term zero crossing rate Z_{s_1}. For the speech signal $s_1(n)$ these measures are calculated as follows:

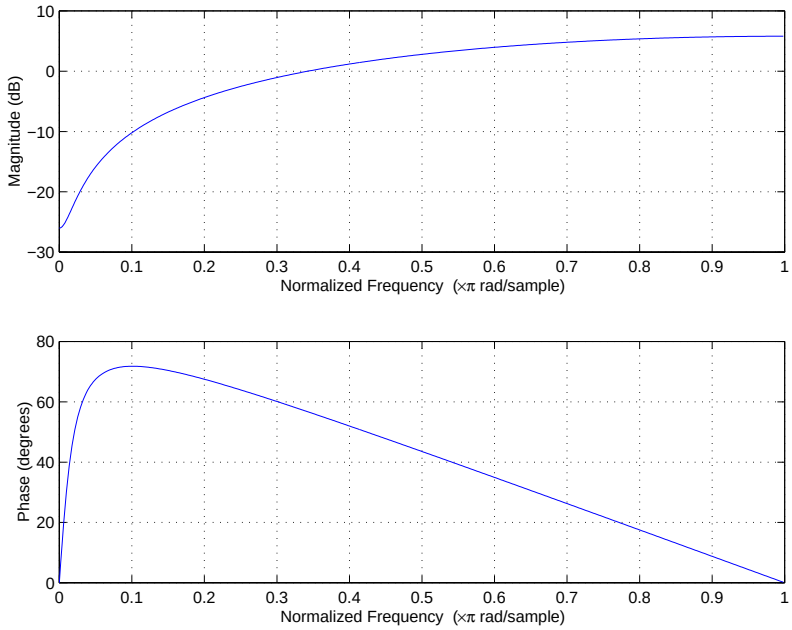

Figure 3.3: Preemphasis filter

$$E_{s_1}(m) = \sum_{n=m-L+1}^{m} s_1^2(n) \tag{3.3}$$

$$P_{s_1}(m) = \frac{1}{L} \sum_{n=m-L+1}^{m} s_1^2(n) \tag{3.4}$$

$$Z_{s_1}(m) = \frac{1}{L} \sum_{n=m-L+1}^{m} \frac{|\mathrm{sgn}(s_1(n)) - \mathrm{sgn}(s_1(n-1)|}{2} \tag{3.5}$$

Where:

$$\mathrm{sgn}(s_1(n)) = \begin{cases} +1, & s_1(n) \geq 0 \\ -1, & s_1(n) < 0 \end{cases} \tag{3.6}$$

For each block of L samples these measures calculate some value. Note that the index for these functions is m and not n, this because these measures do not have to be calculated for every sample (the measures can for example be calculated in every 20 ms). The short-term energy estimate will increase when speech is present in $s_1(n)$. This is also the case with the short-term power estimate, the only thing that separates them is scaling with $\frac{1}{L}$ when calculating the short-term power estimate. The short term zero crossing rate gives a measure of how many times the signal, $s_1(n)$, changes sign. This short term zero crossing rate tends to be larger during unvoiced regions [2].

These measures will need some triggers for making decision about where the utterances begin and end. To create a trigger, one need some information about

the background noise. This is done by assuming that the first 10 blocks are background noise. With this assumption the mean and variance for the measures will be calculated. To make a more comfortable approach the following function is used:

$$W_{s_1}(m) = P_{s_1}(m) \cdot (1 - Z_{s_1}(m)) \cdot S_c \tag{3.7}$$

Using this function both the short-term power and the zero crossing rate will be taken into account. S_c is a scale factor for avoiding small values, in a typical application is $S_c = 1000$. The trigger for this function can be described as:

$$t_W = \mu_W + \alpha\delta_W \tag{3.8}$$

The μ_W is the mean and δ_W is the variance for $W_{s_1}(m)$ calculated for the first 10 blocks. The α term is a constant that have to be fine tuned according to the characteristics of the signal. After some testing the following approximation of α will give a pretty good voice activation detection in various level of additive background noise:

$$\alpha = 0.2 \cdot \delta_W^{-0.8} \tag{3.9}$$

The voice activation detection function, VAD(m), can now be found as:

$$\text{VAD}(m) = \begin{cases} 1, & W_{s_1}(m) \geq t_W \\ 0, & W_{s_1}(m) < t_W \end{cases} \tag{3.10}$$

VAD(n) is found as VAD(m) in the block of measure. For example if the measures is calculated every 320 sample (block length L=320), which corresponds to 20 ms if the sampling rate is 16 kHz. The first 320 samples of VAD(n) found as VAD(m) then $m = 1$. Using these settings the VAD(n) is calculated for the speech signal containing the words four and five, see Fig. 3.4. If some other sampling rate is used, strive for a block length of 20 ms.

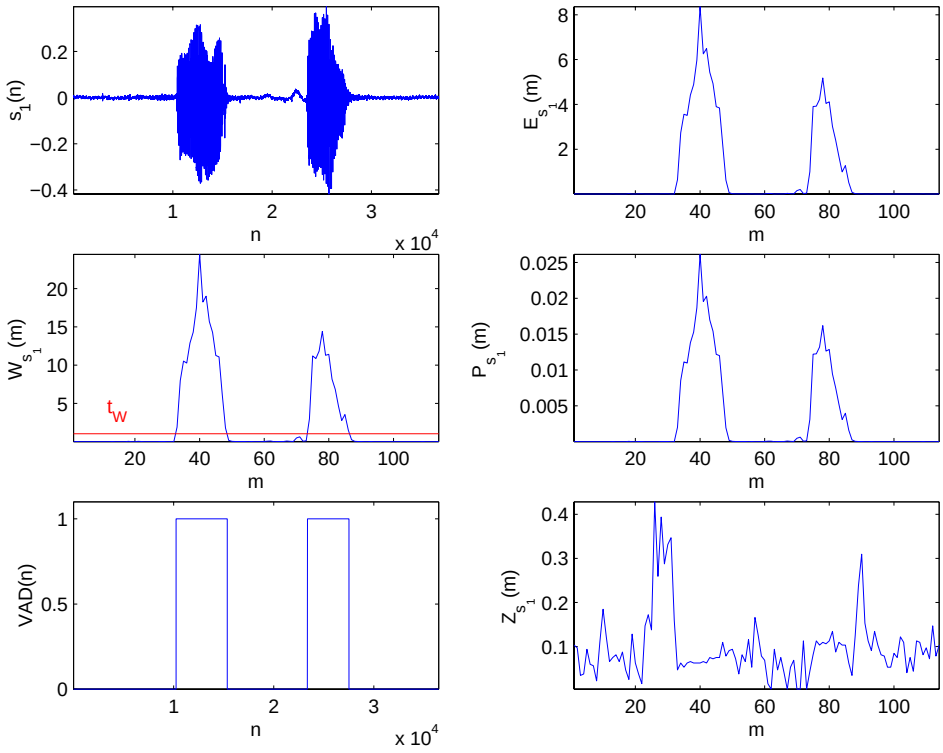

Figure 3.4: Different measures used to find speech content, VAD(n)

With the function, VAD(n), the calculation of $x_1(n)$ is simply $s_1(n)$ when VAD(n) is one, see Fig. 3.5. After this step the preprocessing is ready and $x_1(n)^2$ is prepared and calculated for the next step.

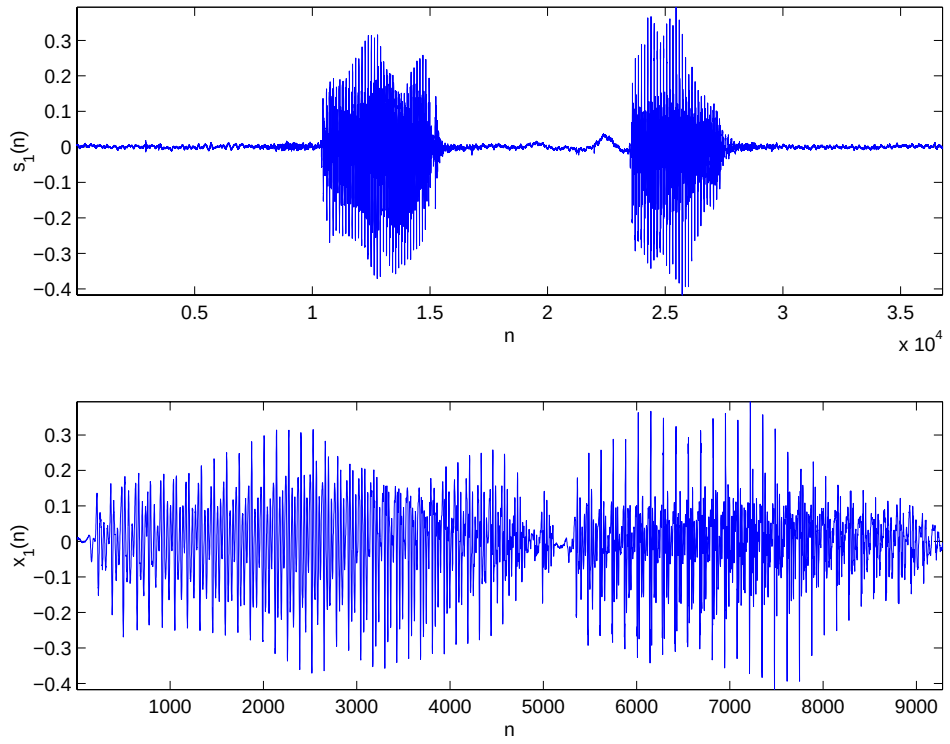

Figure 3.5: From speech with silence, $s_1(n)$, to speech with no silence, $x_1(n)$, using decision function VAD(n)

[2]The sample index n, for $x_1(n)$ is here different from the index for $s_1(n)$

3.2 Frame Blocking and Windowing

The next thing to do with $x_1(n)$ is to divide it into speech frames and apply a window to each frame, see Fig. 3.6.

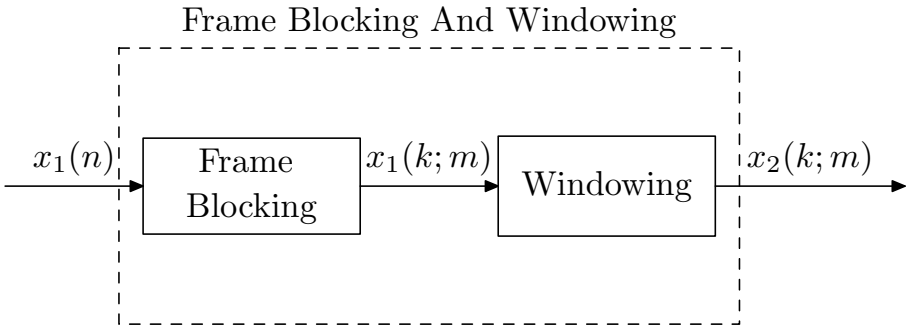

Figure 3.6: Steps in Frame Blocking and Windowing

Each frame is K samples long, with adjacent frames being separated by P samples, see Fig. 3.7.

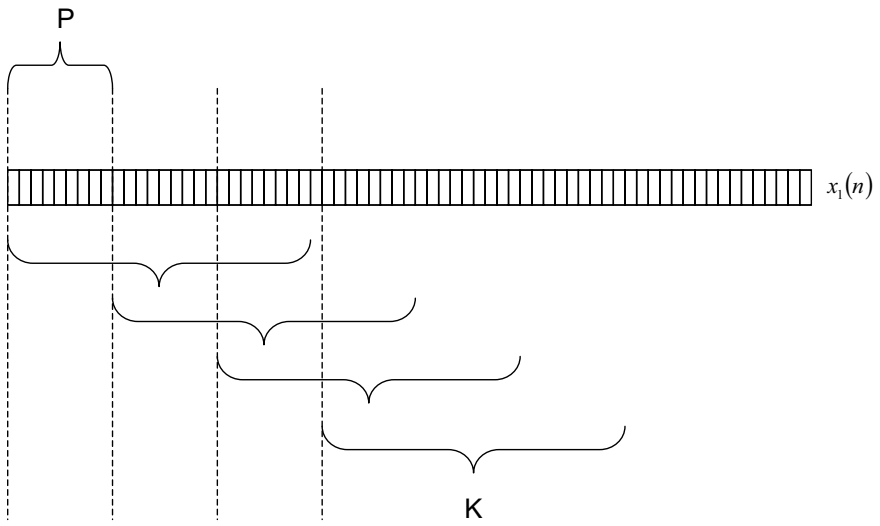

Figure 3.7: Frame blocking of a sequence $x_1(n)$

Typically values for K and P are 320 samples and 100 samples (62.5% overlap) which correspond to 20 ms frames, separated by 6.25 ms when the sampling rate of the speech is 16 kHz. By choosing frames of 20 ms one can assume that the speech is stationary within each frame [2]. By applying the frame blocking to $x_1(n)$ one will get M vectors of length K, which correspond to $x_1(k;m)$ where $k = 0, 1, \ldots, K-1$ and $m = 0, 1, \ldots, M-1$. The next thing to do is to apply a

window to each frame in order to reduce signal discontinuity at either end of the block. A commonly used window is the Hamming window. It is calculated as:

$$w(k) = 0.54 - 0.46cos(\frac{2\pi k}{K-1}) \tag{3.11}$$

By applying $w(k)$ to $x_1(k;m)$ for all blocks, $x_2(k;m)$ is calculated.

3.3 Feature Extraction

The next step is an important one, namely to extract relevant information from the speech blocks. A variety of choices for this task can be applied. Some commonly used methods for speech recognition is linear prediction and mel-cepstrum [1]. These measures has been widely used and here are some reasons why [1]:

- These measures provides a good model of the speech signal. This is particulary true in quasi steady state voiced region of speech.
- The way these measures is calculated leads to a reasonable source-vocal tract separation. This property leads to a fairly good representation of the vocal tract characteristics (which is directly related to the speech sound being produced).
- The measures has a analytically tractable model.
- Experience has found that these measures works well in recognition application.

Other measures to add to the feature vectors are energy measures and also the calculation of delta and acceleration coefficients. The delta coefficients means that a derivative approximation of some measure (e.g. linear prediction coefficients) is added and the acceleration coefficients is the second derivative approximation of some measures.

3.3.1 Linear Prediction

The main idea behind linear prediction is to extract the vocal tract parameters. A model of the vocal tract can be seen in Fig. 3.8.
Given a speech sample at time n, $s(n)$ can be modelled as a linear combination of the past p speech samples, such that:

$$s(n) = b_0u(n) + a_1s(n-1) + a_2s(n-2) + \cdots + a_ps(n-p) \tag{3.12}$$

Or equivalently:

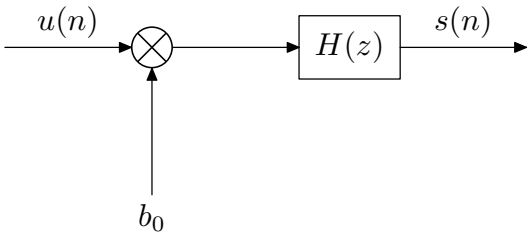

Figure 3.8: Model of the vocal tract

$$s(n) = b_0 u(n) + \sum_{i=1}^{p} a_i s(n-i) \tag{3.13}$$

where $u(n)$ is a normalized excitation signal, b_0 the gain of the excitation signal and the coefficients $a_1, a_1, \ldots, a_p$ are the weights for previous sound samples. All these coefficients are assumed constant over the speech analysis frame. An other way to look at this is to express $s(n)$ in the z-domain:

$$S(z) = b_0 U(z) + \sum_{i=1}^{p} a_i S(z) z^{-i} \tag{3.14}$$

And hereby the transfer function becomes:

$$H(z) = \frac{S(z)}{U(z)} = \frac{b_0}{1 - \sum_{i=1}^{p} a_i z^{-i}} \tag{3.15}$$

As (3.15) shows, one can see that this problem is to create an all pole model of the vocal tract. The calculation of the coefficients are applied when the speech is assumed stationary, $x_2(k;m)$ are the frames of speech where the speech is assumed stationary. The calculation of these coefficients for each block in $x_2(k;m)$ can be done in different ways by using the *autocorrelation method*, the *covariance method*, the *Levinson-Durbin recursion* etc [3]. This report focus on the Levinson-Durbin recursion method. The steps in linear prediction can be seen in Fig. 3.9.

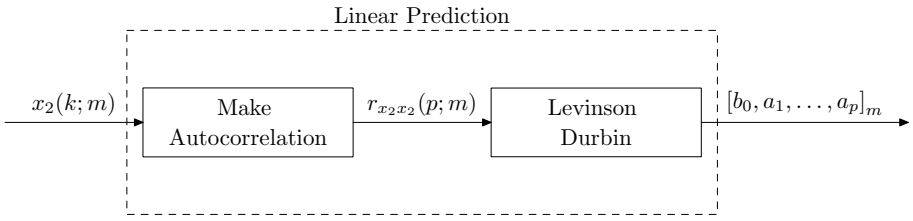

Figure 3.9: Steps in Linear Prediction

To get a deeper understanding of the different ways to extract the predictor coefficients see [3]. In a typical application is $p = 13$.

3.3.2 Mel-Cepstrum

Instead of using linear prediction, another method is often used in speech recognition, namely the *mel-cepstrum*. This method consists of two parts: the cepstrum calculation and a method called mel scaling. The parts will be described in the following two sections and finally the mel-cepstrum will be summarized.

Cepstrum

The cepstrum method is a way of finding the vocal tract filter $H(z)$ with "homomorphic processing". Homomorphic signal processing is generally concerned with the transformation to linear domain of signals combined in a nonlinear way. In this case the two signals are not combined linearly (a convolution can't be described as an simple linear combination). As Fig. 3.8 shows, the speech signal, $s(n)$, can be seen as the result of a convolution between $u(n)$ and $h(n)$:

$$s(n) = b_0 \cdot u(n) * h(n) \tag{3.16}$$

In the frequency domain:

$$S(z) = b_0 \cdot U(z)H(z) \tag{3.17}$$

Since the excitation, $U(z)$, and the vocal tract, $H(z)$, are combined multiplicative it is difficult to separate them. If the log[3] operation is applied the task will become additive:

$$\log S(z) = \log(b_0 \cdot U(z)H(z)) = \log(b_0 \cdot U(z)) + \log(H(z)) \tag{3.18}$$

The additive property of the log spectrum also applies when an inverse transforming is applied to it. The result of this operation is called a *cepstrum*. To avoid taking logarithm of complex numbers, an abs operation is applied to $S(z)$, this is the definition of a "real cepstrum". The steps to create a real cepstrum can be seen in Fig. 3.10.

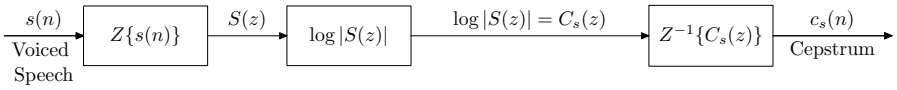

Figure 3.10: Steps to get real cepstrum

Note that the real cepstrum is an even sequence on the index n, since $C_s(z) = \log|S(z)|$, is real and even. These properties gives that the inverse cosine transform can be applied to $C_s(z)$ to get $c_s(n)$. The index n in $c_s(n)$ is now the so called quefrency - high-quefrency equals high n and vice versa. Now that $c_s(n)$ has been calculated one will get hold of $c_h(n)$, which is the cepstrum of the vocal

[3]By logarithm the natural logarithm is assumed if not stated otherwise

tract filter. The vocal tract filter has "slow" spectral variations and the excitation signal has "fast" variations. This property corresponds to low-quefrency for the vocal tract filter and high-quefrency for the excitation signal. Fig. 3.11 shows the quefrency domain operations. Note that the P in the cepstrum for the excitation is the corresponding pitch.

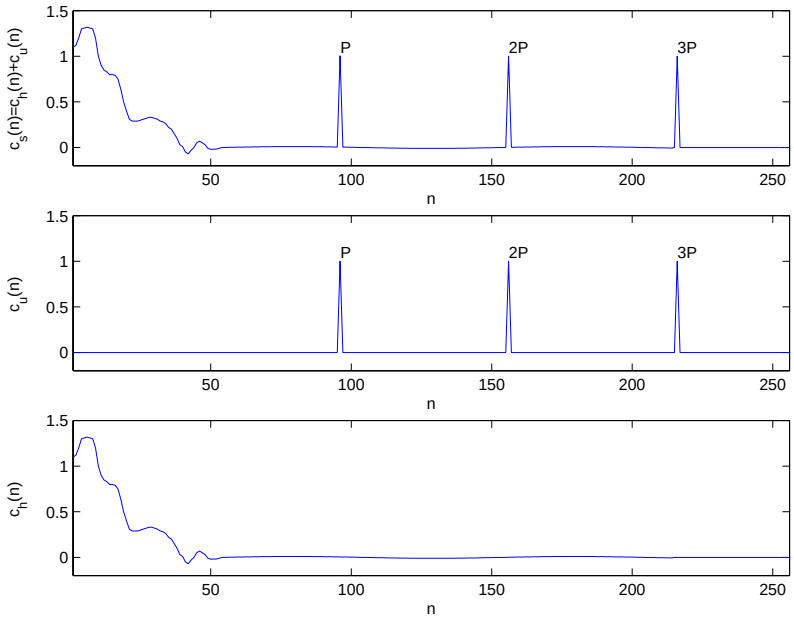

Figure 3.11: Quefrency functions for vocal tract model

To extract the vocal tract cepstrum one can apply a low-pass "lifter" to $c_s(n)$. Liftering is filtering in cepstrum domain. One easy way to do this is to drop some of the cepstrum coefficients at the end. This can be formulated as a window which $c_s(n)$ is to be multiplicated with:

$$l_1(n) = \begin{cases} 1, & n = 0, 1, \ldots, L-1 \\ 0, & else \end{cases} \tag{3.19}$$

Where the length, L, is the chosen to extract $c_h(n)$. In Fig. 3.11 a good choice of L would be 75. An other lifter which has also been proven to give a good recognition result is [1]:

$$l_2(n) = \begin{cases} 1 + \frac{L-1}{2} \sin\left(\frac{\pi n}{L-1}\right), & n = 0, 1, \ldots, L-1 \\ 0, & else \end{cases} \tag{3.20}$$

Both lifters can be seen in Fig. 3.12.
Now the cepstrum for the vocal tract, $c_h(n)$, can be calculated by (using second lifter):

$$c_h(n) \approx c_s(n) \cdot l_2(n) \tag{3.21}$$

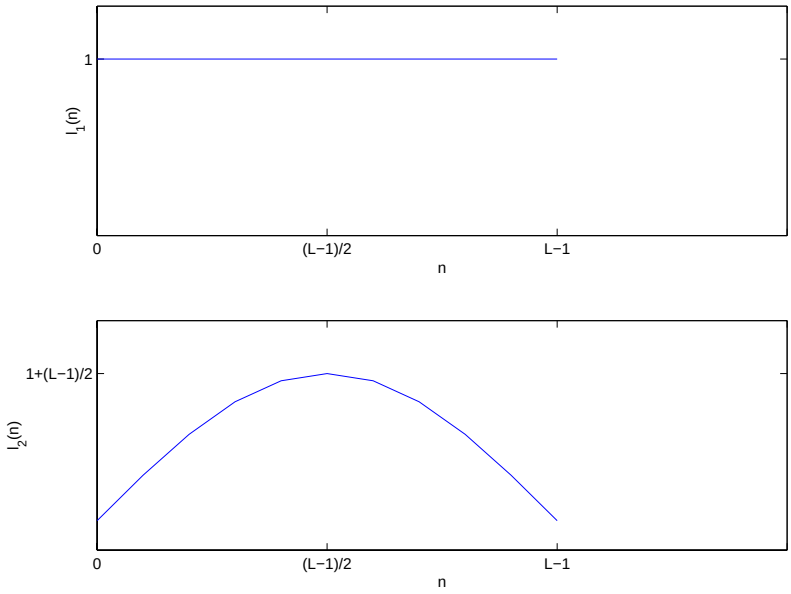

Figure 3.12: Lifters $l_1(n)$ and $l_2(n)$

Mel scale

As mentioned in Ch. 2, psychophysical studies (studies of human auditory perception [2]) has shown that human perception of the frequency contents of sound, for speech signals does not follow a linear scale. Thus for each tone with an actual frequency, F, measured in Hz, a subjective pitch is measured on a scale called the "mel" scale[4]. As reference for the mel scale, 1000 Hz is usually said to be 1000 mels. As a nonlinear transformation of the frequency scale, the following formula is used:

$$F_{\text{mel}} = 2595 \cdot \log_{10}(1 + \frac{F_{\text{Hz}}}{700}) \tag{3.22}$$

This nonlinear transformation can be seen in Fig. 3.13.

To apply the mel scale to the cepstrum, a filterbank of K triangular bandpass filters is applied to $|S(z)|$. These triangular bandpass filters has centrum frequencies in K equally spaced mel values. The equally spaced mel values correspond to different frequency values, that is the inverse of (3.22):

$$F_{\text{Hz}} = 700 \cdot (10^{\frac{F_{\text{mel}}}{2595}} - 1) \tag{3.23}$$

If for example one wants K mel scaled coefficients in the range 0-5000 Hz (Nyquist range), the first thing to do is to use (3.22) to get F_{mel} corresponding to $F_{\text{Hz}} =$

[4]From the experiments by Stevens and Volkman (1940) [2]

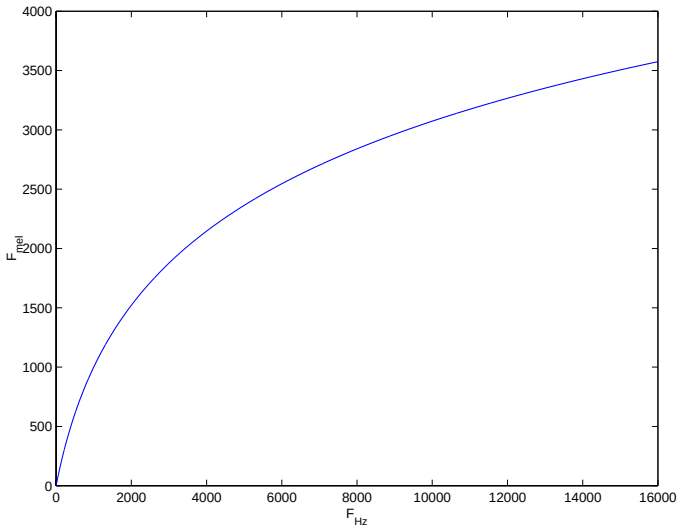

Figure 3.13: Transformation from Hz to *mels*

5000. Now the calculation of the centrum frequency in the equally spaced mel scale can easily be done by dividing the calculated F_{mel} with K. Now the equally spaced mel values are calculated and the reverse operation is done to get back to F_{Hz}. In Fig. 3.14 this is illustrated for $K = 10$.

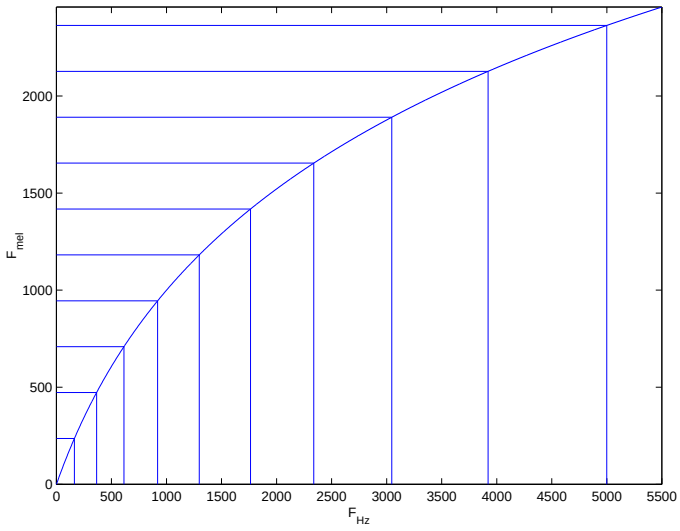

Figure 3.14: Equally spaced mel values

Now the centrum of the triangular filter is found in Hz and the triangular band-pass filters can be found. The width of each filter is just the distance to the previous centrum times 2 (the first centrum frequency has zero as its previous centrum value). Fig. 3.15 shows the mel scaled filter bank when $K = 10$.

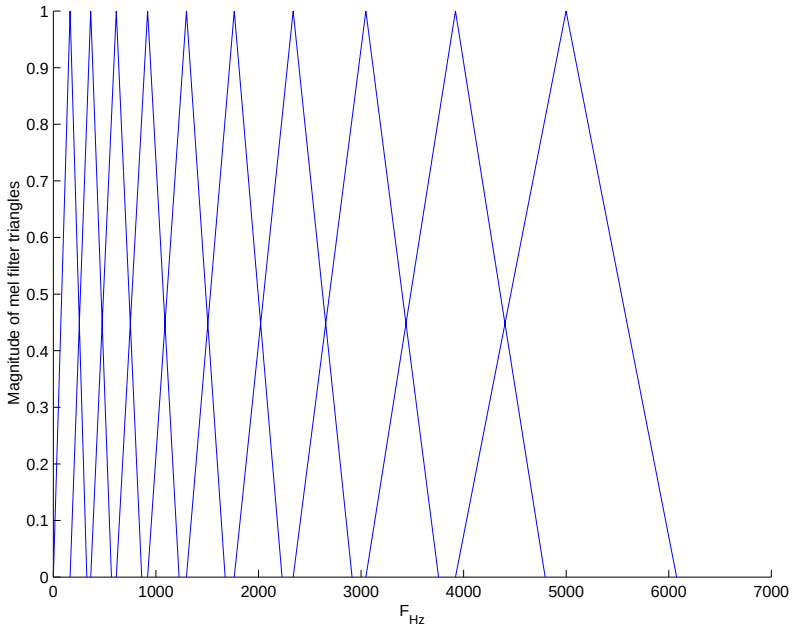

Figure 3.15: Mel scale filter bank

The bandlimits for these filters, when $K = 10$ (i.e. the bandlimits in Fig. 3.15) and when $K = 20$ can be seen in Tab. 3.1 and Tab. 3.2.

Filter	Lower limit [Hz]	Upper limit [Hz]
1	0	326.6253
2	163.3126	566.1408
3	364.7267	861.5363
4	613.1315	1225.8486
5	919.4901	1675.1564
6	1297.3232	2229.2893
7	1763.3063	2912.7036
8	2338.0049	3755.561
9	3046.7829	4795.0602
10	3920.9215	6077.0785

Table 3.1: Bandlimits when $K = 10$

Every triangular filter now will give one new mel spectrum coefficient, m_k, by summing up the filtered result. The chain of calculating a mel-cepstum for a speech frame will be described in the following section.

Filter	Lower limit [Hz]	Upper limit [Hz]
1	0	154.759
2	77.3795	249.2458
3	163.3126	354.1774
4	258.745	470.7084
5	364.7267	600.121
6	482.4239	743.8391
7	613.1315	903.4442
8	758.2878	1080.6923
9	919.4901	1277.5338
10	1098.5119	1496.1345
11	1297.3232	1738.8999
12	1518.1115	2008.501
13	1763.3063	2307.9044
14	2035.6053	2640.4045
15	2338.0049	3009.6599
16	2673.8324	3419.7335
17	3046.7829	3875.1375
18	3460.9602	4380.8829
19	3920.9215	4942.5344
20	4431.728	5566.272

Table 3.2: Bandlimits when $K = 20$

Mel-Cepstrum

All steps to create mel-cepstrum coefficients from a speech frame will be described in this section. The sampling rate used in this description is F_s =16 kHz and the block size $K = 320$. The steps for this operation can be viewed in Fig. 3.16.

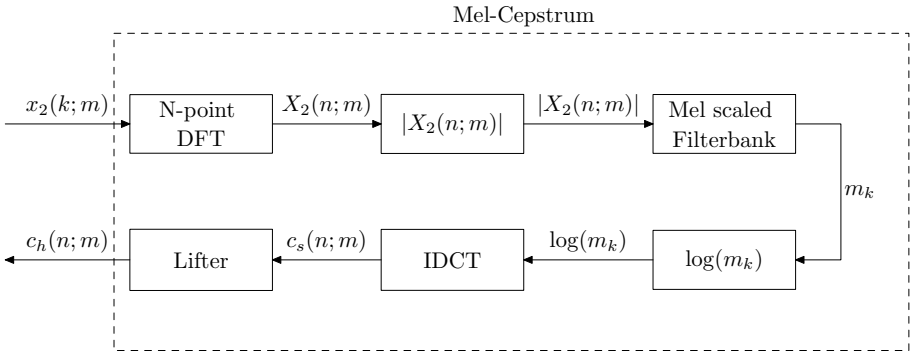

Figure 3.16: The steps in creation of Mel-Cepstrum

The first step is to make the block length to a power of 2 length ($N = 2^i$) which enables a fast radix-2 algorithm to be used for calculating the FFT of the block. In this case $K = 320$ gives an FFT of length 512, which means that the block will be zeropadded with 192 zeros. After applying an N-point FFT to $x_2(k;m)$ one will get $X_2(n;m)$. After this, the magnitude of $X_2(n;m)$ is calculated and is used with the mel scale filter bank. The mel spectrum coefficients are then the sum of the filtered result. This can be described by:

$$m_k = \sum_{n=0}^{N-1} |X_2(n;m)| H_k^{mel}(n) \tag{3.24}$$

Where $H_k^{mel}(n)$ is one triangular filter. Here 20 filters[5] are used in the range 0-5000 Hz and these filters are illustrated in Fig. 3.17 as FFT vectors.
After the mel spectrum coefficients are calculated, as in (3.24), the logarithm is taken and the inverse discrete cosine transform is applied as:

$$c_s(n;m) = \sum_{k=0}^{N-1} \alpha_k \cdot \log(m_k) \cos\left(\frac{\pi(2n+1)k}{2N}\right),\ n = 0, 1, \ldots, N-1 \tag{3.25}$$

Where:

$$\begin{cases} \alpha_0 = \sqrt{\frac{1}{N}} \\ \alpha_k = \sqrt{\frac{2}{N}},\ 1 \le k \le N-1 \end{cases} \tag{3.26}$$

[5]Please recall Tab. 3.2

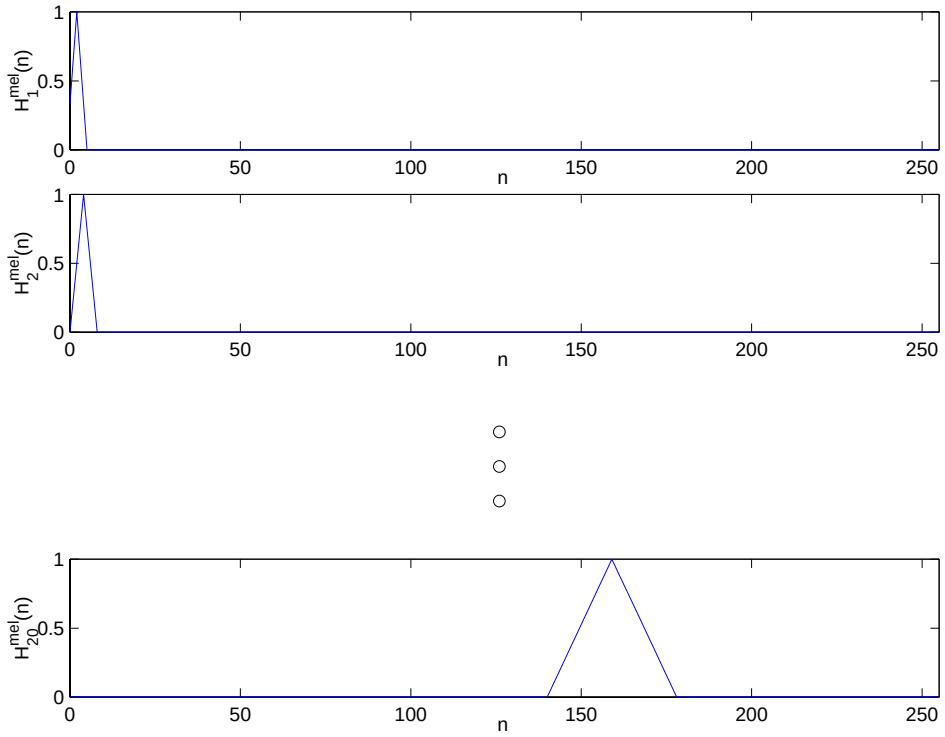

Figure 3.17: Mel scale filters as FFT vectors

Note that N is now the number of wanted cepstrum coefficients, not the FFT length. Note also that only K values, m_k, are available for the calculation. Usually $N = K$, otherwise one need to zeropad or truncate the m_k values to N wanted values. N values of the frame mel cepstrum are extracted, $c_s(n;m)$, and the liftering takes place. The liftering length, L, has to be chosen and a good choice for removing the pitch is $L = \frac{2}{3}N$. This can be described by (using second lifter):

$$c_h(n;m) = c_s(n;m) \cdot \left(1 + \frac{L-1}{2} \sin\left(\frac{\pi n}{L-1}\right)\right), \; n = 0, 1, \ldots, L-1 \qquad (3.27)$$

Note that this operation zeros out (or cuts away) some of the last mel cepstrum coefficients[6]. After this step the final mel cepstrum values are found. A way to interpret these values visually, is to make an FFT of $c_h(n;m)$ to see its spectral information. The spectral information given is actually $\log|H(n;m)|$, which is the log of the magnitude for the vocal tract in mel scale. To get the magnitude for the vocal tract filter the following formula can be used:

$$|H(n;m)| = \left|e^{FFT_N\{c_h(n;m)\}}\right| \qquad (3.28)$$

[6]Please recall Fig. 3.12

Note that the N-points in $|H(n;m)|$ are now represented in melscale, so $|H(n;m)|$ is actually in mel scale and not ordinary frequency scale. This is illustrated for a speech file in Fig. 3.18, sampling rate was 16 kHz, the range used for the mel filterbank was 0-5000 Hz and an 512 point FFT was used.

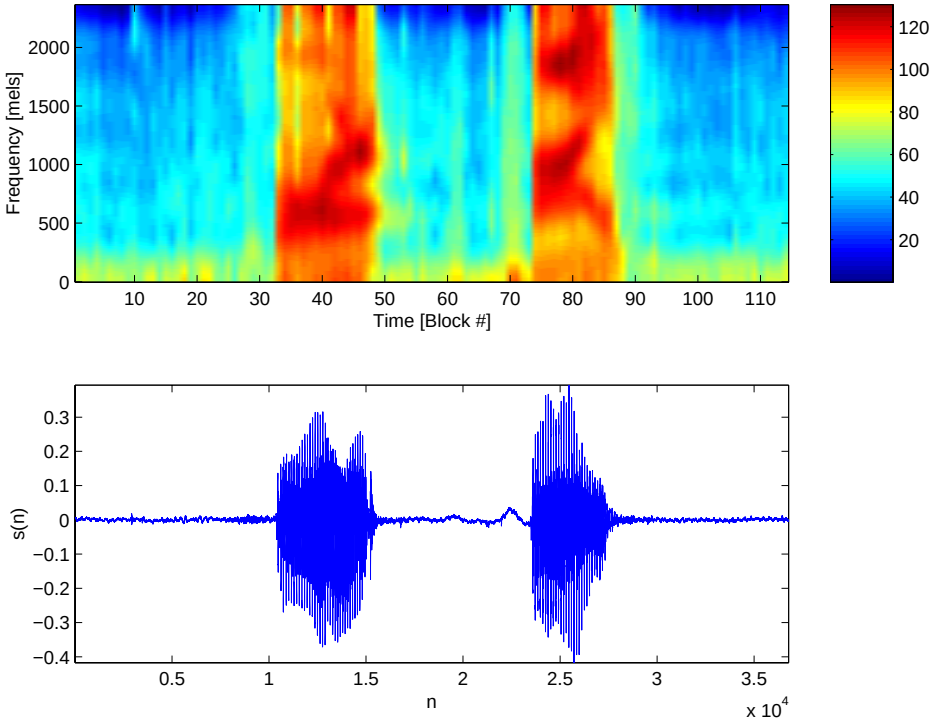

Figure 3.18: Spectral representation of cepstrum coefficients

3.3.3 Energy measures

An extra measure to augment the coefficients derived from linear prediction or mel-cepstrum, is the *log of signal energy*. This means that, for every frame, an extra energy term is added. This energy term is calculated as:

$$E_m = \log \sum_{k=0}^{K-1} x_2^2(k;m) \tag{3.29}$$

3.3.4 Delta and Acceleration Coefficients

Spectral transitions are believed to play an important role in human speech perception. Therefore it is desired to add information about time difference, or delta coefficients and also acceleration coefficients (second time derivative). The notation for mel-cepstrum will be used in this section, but the same equations

can also be applied to linear prediction coefficients. One direct way to get delta coefficients is:

$$\Delta c_h(n;m) = c_h(n;m+1) - c_h(n;m) \tag{3.30}$$

This delta approximation will be quite noisy and some way to smooth it is desired. This is why the time derivative, $\partial c_h(n;m)/\partial m$, is normally obtained by polynomial approximation. The approximation is done by fitting a polynomial over a segment of the mel-cepstral trajectory. The polynomial fit has to be done individually for each of the mel-cepstral coefficients $c_h(n;m)$, $n = 0, 2, \ldots, N-1$. Consider fitting a segment of mel-cepstrum trajectory, $c_h(n;m+p)$, $p = -P, -P+1, \ldots, P$ by a second-order polynomial $h_1(n;m) + h_2(n;m)p + h_3(n;m)p^2$. The parameters $h_1(n;m)$,$h_2(n;m)$ and $h_3(n;m)$ has to be chosen to minimize some criterium. The criterium to be minimized, is the least square. This gives that the following function has to be minimized:

$$E = \sum_{p=-P}^{P} \Big[c_h(n;m+p) - \Big(h_1(n;m) + h_2(n;m)p + h_3(n;m)p^2 \Big) \Big]^2 \tag{3.31}$$

Differentiating E with respect to $h_1(n;m)$,$h_2(n;m)$ and $h_3(n;m)$ and setting the result to zero, lead to three simultaneous equations (The index $(n;m)$ for h_1,h_2 and h_3 is dropped for simplicity):

$$\begin{cases} \dfrac{\partial E}{\partial h_1} = -2 \cdot \displaystyle\sum_{p=-P}^{P} \Big[c_h(n;m+p) - h_1 - h_2 p - h_3 p^2 \Big] = 0 \\ \dfrac{\partial E}{\partial h_2} = -2 \cdot \displaystyle\sum_{p=-P}^{P} \Big[c_h(n;m+p)p - h_1 p - h_2 p^2 - h_3 p^3 \Big] = 0 \\ \dfrac{\partial E}{\partial h_3} = -2 \cdot \displaystyle\sum_{p=-P}^{P} \Big[c_h(n;m+p)p^2 - h_1 p^2 - h_2 p^3 - h_3 p^4 \Big] = 0 \end{cases} \tag{3.32}$$

The solution to (3.32), can after some calculations, be found as:

$$h_2 = \frac{\displaystyle\sum_{p=-P}^{P} c_h(n;m+p)p}{T_P} \tag{3.33}$$

$$h_3 = \frac{T_P \displaystyle\sum_{p=-P}^{P} c_h(n;m+p) - (2P+1) \sum_{p=-P}^{P} c_h(n;m+p)p^2}{T_P^2 - (2P+1) \displaystyle\sum_{p=-P}^{P} p^4} \tag{3.34}$$

$$h_1 = \frac{1}{2P+1}\left[\sum_{p=-P}^{P} c_h(n;m+p) - h_3 T_P\right] \tag{3.35}$$

Where:

$$T_P = \sum_{p=-P}^{P} p^2 \tag{3.36}$$

To illustrate the approximation on a typical mel-cepstrum trajectory, see Fig. 3.19.

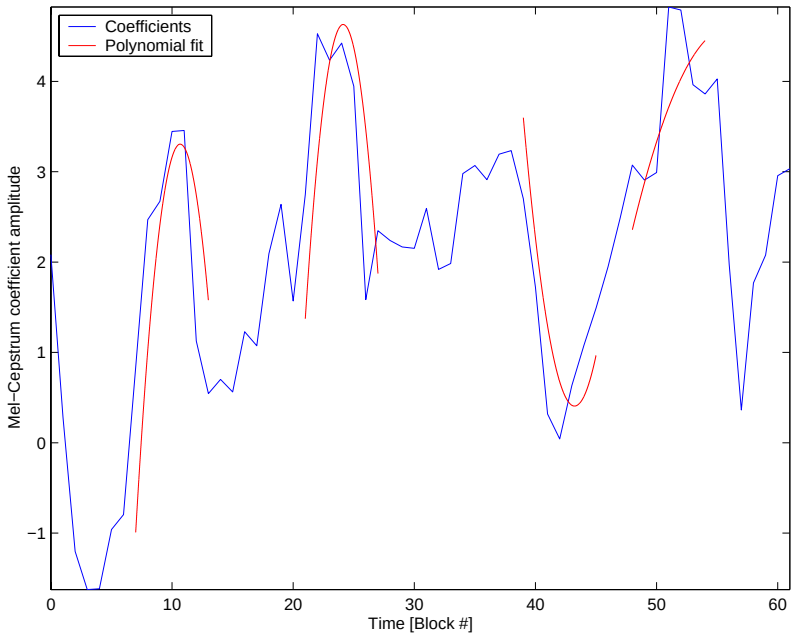

Figure 3.19: 2nd-order polynomial approximation, fitting width is 7 ($P = 3$)

With the approximation curve, $h_1+h_2p+h_3p^2$, an approximation of the derivative can be found as:

$$\left.\frac{\partial c_h(n;m+p)}{\partial p}\right|_{p=0} \simeq \left.\frac{\partial\left(h_1+h_2p+h_3p^2\right)}{\partial p}\right|_{p=0} = \lim_{p\to 0}\left(h_2+2h_3p\right) = h_2 \tag{3.37}$$

From this point on the estimate $\left.\frac{\partial c_h(n;m+p)}{\partial p}\right|_{p=0}$ is denoted $\delta^{[1]}(n;m)$, which is the same as the delta coefficients. An explicit expression for the delta coefficients is given by adding (3.33) to (3.37):

$$\delta^{[1]}(n;m) = h_2(n;m) = \frac{\sum_{p=-P}^{P} c_h(n;m+p)p}{T_P} \tag{3.38}$$

Knowing the polynomial approximation it is also possible to get the second derivative, the acceleration coefficients, by taking the second derivative of the approximation polynomial. This leads to the second derivative approximation:

$$\left.\frac{\partial^2 c_h(n;m+p)}{\partial p^2}\right|_{p=0} \simeq \left.\frac{\partial^2 \left(h_1 + h_2 p + h_3 p^2\right)}{\partial p^2}\right|_{p=0} = \lim_{p\to 0}\left(2h_3\right) = 2h_3 \tag{3.39}$$

The acceleration coefficients will be denoted $\delta^{[2]}(n;m)$ and by using (3.34) and (3.39), the following formula, for the acceleration coefficients will be used:

$$\delta^{[2]}(n;m) = 2h_3(n;m) = \frac{2\left(T_P \sum_{p=-P}^{P} c_h(n;m+p) - (2P+1)\sum_{p=-P}^{P} c_h(n;m+p)p^2\right)}{T_P^2 - (2P+1)\sum_{p=-P}^{P} p^4} \tag{3.40}$$

The delta and acceleration coefficients can now be augmented to the feature vectors to get more relevant information about the speech. The choice of the fitting width (fitting width $= 2P+1$) adds a delay of P blocks to the system. A good choice is $P = 3$, which gives a good approximation and not too long delay [1].

3.3.5 Summary

Different measures for each block has been presented and the final choice of measures will be done, representing the step from $x_2(k;m)$ to $x_3(n;m)$ will be described here, see Fig. 3.1. The coefficients chosen are the energy measure, mel-cepstrum coefficients and the delta and acceleration coefficients calculated from the mel-cepstrum coefficients. The vectors $x_3(n;m)$ are build up as in Fig. 3.20.

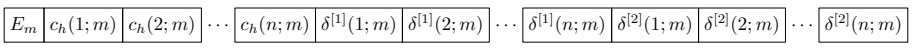

Figure 3.20: Measures saved in vector for block m

The process to extract the features can be viewed in Fig. 3.21.
The outputs seen in Fig. 3.21 are the desired features, $x_3(n;m)$, saved in a vector (see Fig. 3.20).

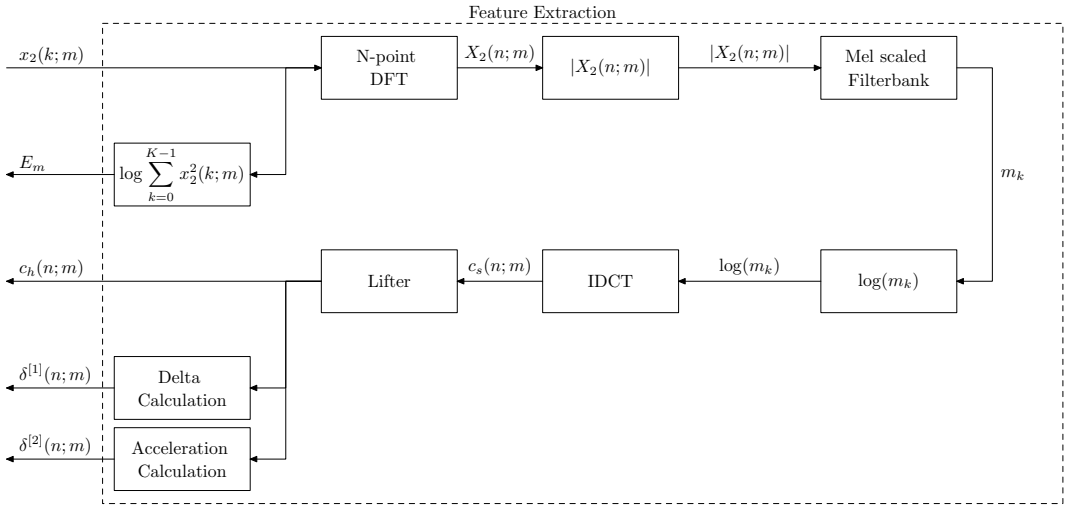

Figure 3.21: Steps in Feature Extraction

3.4 Postprocessing

The final touch of the features will be done in this section. Two more steps are included to get the final feature vectors. These steps can be viewed in Fig. 3.22.

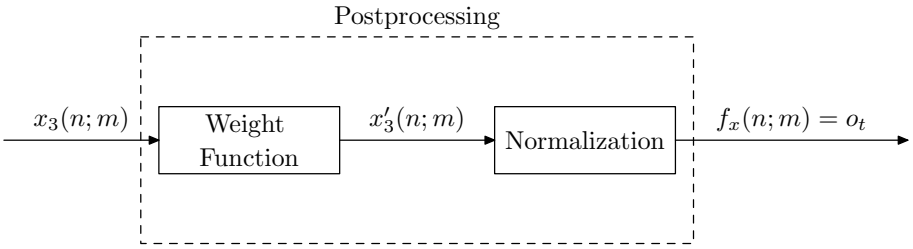

Figure 3.22: Steps in Postprocessing

After the desired features are chosen, these can be weighted with some weight function to give some features more or less influence. For example it might be desired to weight down some of the first mel-cepstrum coefficients, if a low frequency noise is present. This is done by a weight function:

$$x'_3(n;m) = x_3(n;m) \cdot w(n) \; n = 0, 2, \ldots, N-1 \tag{3.41}$$

The weight function, $w(n)$, is a way to be able to fine tune the speech recognizer for a given task. Assume $w(n) = 1$, but keep in mind that this window can be adjusted to get better speech representation for some tasks.

The next step is the normalization, meaning that the feature vectors are normalized over time to get zero mean. The mean vector, called $f_{\hat{\mu}}(n)$, can be calculated as:

$$f_{\hat{\mu}}(n) = \frac{1}{M} \sum_{m=0}^{M-1} x'_3(n;m) \tag{3.42}$$

To normalize the feature vectors, the following operation is applied:

$$f_x(n;m) = x'_3(n;m) - f_{\hat{\mu}}(n) \tag{3.43}$$

For training purpose can equations (3.42) and (3.43) be applied for some fix number of feature vectors, but when the recognition phase is done in real time it is necessary to calculate the mean vector online. Online calculation is done by setting the start mean vector to zero and update the mean for each new feature vector. The normalization procedure will force the feature vectors to the same numerical range. This property is found to make the speech recognition system more robust to noise and also gives less mismatch between training and recognition environments.

This will conclude the feature extraction process. To simplify the notation in Ch. 4, the feature vectors, $f_x(n;m)$, will be denoted $\mathbf{O} = (\mathbf{o}_1, \mathbf{o}_2, \ldots, \mathbf{o}_T)$ where $\mathbf{o}_t \in \mathcal{R}^D$. That is the dimension of the vectors is N for $f_x(n;m)$ and D for $\mathbf{o}_t$. The number of vectors M in $f_x(n;m)$ corresponds to T for $\mathbf{O}$. Note that for a single m will $f_x(n;m)$ correspond to $\mathbf{o}_t$ (single observation).

Chapter 4

Hidden Markov Model

This chapter will describe a method to train and recognize speech utterance from given observations, $\mathbf{o}_t \in \mathcal{R}^D$, where t is a time index and D is the vector dimension. A complete sequence of observations used to describe the utterance will be denoted $\mathbf{O} = (\mathbf{o}_1, \mathbf{o}_2, \ldots, \mathbf{o}_T)$. The utterance may be a word, a phoneme, or, in principle, a complete sentence or paragraph. The method described here is the *Hidden Markov Model* or HMM. The HMM is an stochastic approach which models the given problem as a "doubly stochastic process" in which the observed data are thought to be the result of having passed the "true" (hidden) process through a second process. Both processes are to be characterized using only the one that could be observed. The problem with this approach, is that one do not know anything about the Markov chains that generate the speech. The number of states in the model is unknown, there probabilistic functions are unknown and one can not tell from which state an observation was produced. These properties are *hidden*, and thereby the name *hidden* Markov model.

4.1 Discrete-Time Markov Model

This section will describe the theory of Markov chains, here the *hidden* part is uncovered. The system in this section is thereby an *Observable Markov Model*. Consider a system that may be described at any time being one of a set of N distinct states index by $1, 2, \ldots, N$. At regular spaced, discrete times, the system undergoes a change of state (possibly back to same state) according to a set of probabilities associated with the state. The time instances for a state change is denoted t and the actual state at time t as q_t. In the case of a first order Markov chain, the state transition probabilities do not depend on the whole history of the process, just the preceding state is taken into account. This is the Markov property and is defined as:

$$P\left(q_t = j | q_{t-1} = i, q_{t-2} = k, \ldots\right) = P\left(q_t = j | q_{t-1} = i\right) \tag{4.1}$$

Also consider that the right hand of (4.1) is independent of time, which leads to a set of state transitions probabilities, a_{ij}, of the form:

$$a_{ij} = P\left(q_t = j | q_{t-1} = i\right), \ 1 \leq i, j \leq N \tag{4.2}$$

These state probabilities, a_{ij}, has the following properties (due to standard stochastic constrains) :

$$\begin{aligned} a_{ij} &\geq 0 \qquad \forall j, i \\ \sum_{j=1}^{N} a_{ij} &= 1 \quad \forall i \end{aligned} \tag{4.3}$$

The state transition probabilities for all states in a model can be described by a transition probability matrix:

$$\mathbf{A} = \begin{bmatrix} a_{11} & a_{12} & \dots & a_{1N} \\ a_{21} & a_{22} & \dots & a_{2N} \\ \vdots & \vdots & \ddots & \vdots \\ a_{N1} & a_{N2} & \dots & a_{NN} \end{bmatrix}_{N \times N} \tag{4.4}$$

The only thing remaining to describe the system is the initial state distribution vector (the probability to start in some state). And this vector is described by:

$$\boldsymbol{\pi} = \begin{bmatrix} \pi_1 = P(q_1 = 1) \\ \pi_2 = P(q_1 = 2) \\ \vdots \\ \pi_N = P(q_1 = N) \end{bmatrix}_{N \times 1} \tag{4.5}$$

The stochastic property for the initial state distribution vector is:

$$\sum_{i=1}^{N} \pi_i = 1 \tag{4.6}$$

Where the π_i is defined as:

$$\pi_i = P(q_1 = i), \quad 1 \leq i \leq N \tag{4.7}$$

These are the properties and equations for describing a Markov process. The Markov model can be described by $\mathbf{A}$ and $\boldsymbol{\pi}$.

4.1.1 Markov Model of Weather

In this section an example of a discrete time Markov process will be presented to set ideas about Markov chains. Here a model of the weather will be presented, consider a four state Markov model of the weather, see Fig. 4.1.
Assume that once a day (e.g. in the morning), the weather is observed as being one of the following:

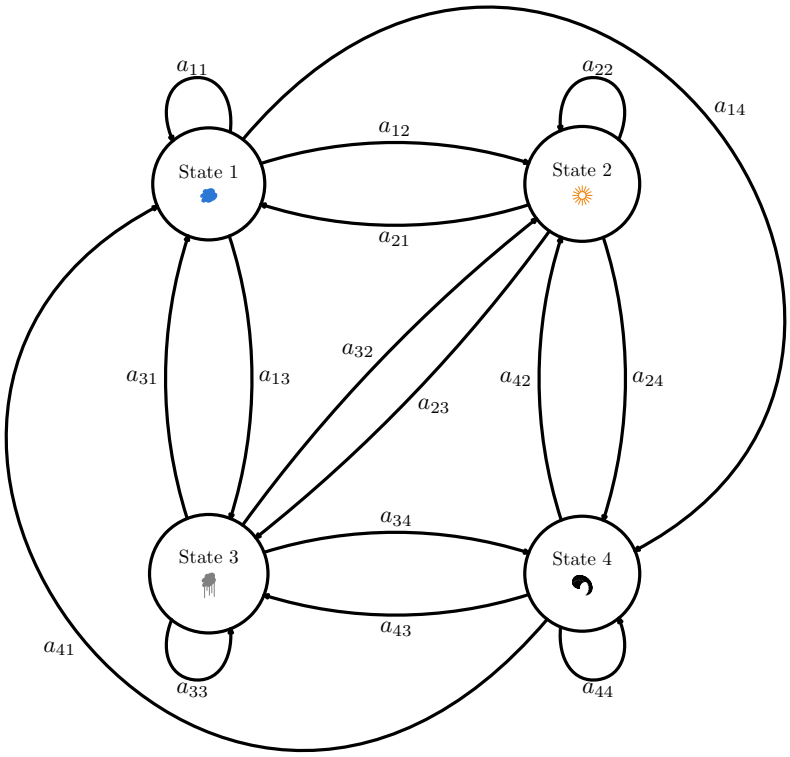

Figure 4.1: Markov model of the weather

- **State 1:** cloudy
- **State 2:** sunny
- **State 3:** rainy
- **State 4:** windy

Given the model in Fig. 4.1, it is now possible to answer several interesting questions about the weather patterns over time. For example, what is the probability to get the sequence "sunny, rainy, sunny, windy, cloudy, cloudy" in six consecutive days? The first thing to do is to define the state sequence, $\mathbf{O}$, as:

$$\mathbf{O} = (\text{sunny}, \text{rainy}, \text{sunny}, \text{windy}, \text{cloudy}, \text{cloudy}) = (2,\ 3,\ 2,\ 4,\ 1,\ 1)$$

Given this sequence and the model of Fig. 4.1, the calculation of the probability of the observation sequence given a Markov model, $P(\mathbf{O}|\mathbf{A}, \boldsymbol{\pi})$, can directly be determined as:

$$\begin{aligned} P(\mathbf{O}|\mathbf{A}, \boldsymbol{\pi}) &= P(2, 3, 2, 4, 1, 1|\mathbf{A}, \boldsymbol{\pi}) \\ &= P(2)P(3|2)P(2|3)P(4|2)P(1|4)P(1|1) \\ &= \pi_2 \cdot a_{23} \cdot a_{32} \cdot a_{24} \cdot a_{41} \cdot a_{11} \end{aligned}$$

In a more general case, this calculation of the probability for a state sequence $\mathbf{q} = (q_1, q_2, \ldots, q_T)$ will be:

$$P(\mathbf{q}|\mathbf{A}, \boldsymbol{\pi}) = \pi_{q_1} \cdot a_{q_1 q_2} \cdot a_{q_2 q_3} \cdot \ldots \cdot a_{q_{T-1} q_T} \tag{4.8}$$

Another question of interest is to find the probability that a the system stays in the same state, given that the system is in a known state, for exactly d days. This corresponds to the following observation sequence:

$$\begin{array}{rcccccccc} \mathbf{O} & = & (i, & i, & i, & \ldots, & i, & j \neq i) \\ \text{day} & & 1 & 2 & 3 & & d & d+1 \end{array}$$

And the following probability (using Bayes rule):

$$\begin{aligned} P(\mathbf{O}|\mathbf{A}, \boldsymbol{\pi}, q_1 = i) &= P(\mathbf{O}, q_1 = i|\mathbf{A}, \boldsymbol{\pi})/P(q_1 = i) \\ &= \pi_i (a_{ii})^{d-1}(1 - a_{ii})/\pi_i \\ &= (a_{ii})^{d-1}(1 - a_{ii}) \\ &= p_i(d) \end{aligned} \tag{4.9}$$

This probability, $p_i(d)$, can be seen as the state duration distribution. Here the characteristic distribution in a Markov chain appears, namely the exponential distribution. Based on $p_i(d)$ it is also possible to find the expected number of observations (duration) in a state, conditioned on starting in that state as:

$$\begin{aligned} E[d_i] &= \sum_{d=1}^{\infty} d \cdot p_i(d) \\ &= \sum_{d=1}^{\infty} d \cdot a_{ii}^{d-1}(1 - a_{ii}) \\ &= (1 - a_{ii}) \frac{\partial}{\partial a_{ii}} \left(\sum_{d=1}^{\infty} a_{ii}^d \right) \\ &= (1 - a_{ii}) \frac{\partial}{\partial a_{ii}} \left(\frac{a_{ii}}{1 - a_{ii}} \right) \\ &= \frac{1}{(1 - a_{ii})} \end{aligned} \tag{4.10}$$

4.2 Discrete-Time Hidden Markov Model

The discrete-time Markov model described in previous section is too restrictive to be applicable to many problems of interest. Therefore it will be extended to

the *discrete-time hidden Markov model*. The extension done is that every state will now not be deterministic (e.g. sunny), instead it will be probabilistic. This means that every state generates a observation, $\mathbf{o}_t$, according to some probabilistic function. The production of observations in this stochastic approach is characterized by a set of observation probability measures, $B = \{b_j(\mathbf{o}_t)\}_{j=1}^{N}$, in which the probabilistic function for each state, j, is:

$$b_j(\mathbf{o}_t) = P\left(\mathbf{o}_t | q_t = j\right) \tag{4.11}$$

To set ideas of how the hidden Markov model works, the so called *urn and ball model* will be presented.

4.2.1 The Urn and Ball Model

Assume that there are N large glass urns in a room. Within each urn are a quantity of colored balls. Assume that there are M distinct colors of the balls. Lets make an example, consider a set of N urns containing colored balls of M = 6 different colors (R = red, O=orange, B=black, G=green, B=blue, P=purple), see Fig. 4.2.

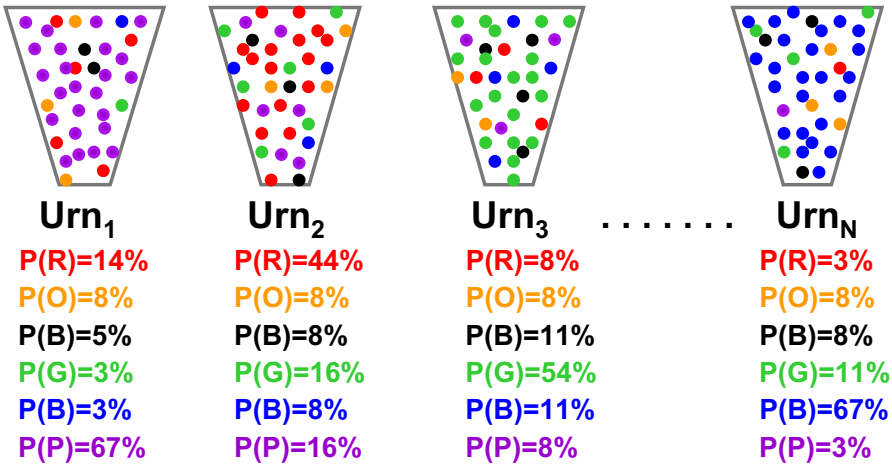

Figure 4.2: Urn and Ball example

The steps for generating an observation sequence is as follows for this urn and ball example:

1. Choose an initial state (here state equals urn) $q_1 = i$ according to the initial state distribution $\boldsymbol{\pi}$.

2. Set $t = 1$ (clock, $t = 1, 2, \ldots, T$).

3. Choose a ball from the selected urn (state) according to the symbol probability distribution in state i, i.e., $b_j(\mathbf{o}_t)$ (for example is the probability for a purple ball in the first urn 0,67, see Fig. 4.2). This colored ball represents the observation $\mathbf{o}_t$. Put the ball back to the urn.

4. Transit to a new state (urn) $q_{t+1} = j$ according to the state-transition probability distribution for state i, i.e., a_{ij}.

5. Set $t = t + 1$; return to step 3 if $t < T$; otherwise, terminate the procedure.

This steps describes how an hidden Markov model works when it is generating the observation sequence. It should be noted that the colors of the balls in each urn may be the same, and the distinction among various urns is the way the collection of the colored balls is composed. Therefore, an isolated observation of a particular colored ball does not immediately tell which urn it is drawn from. Note that the link between the urn and ball example and an example in a speech recognition task, is that an urn is equal to a state and a color is equal to a feature vector (the observation).

4.2.2 Discrete Observation Densities

The urn and ball example described in previous section is an example of a discrete observation density HMM. This because there are M distinct colors. In general the discrete observation density HMMs are based on partitioning the probability density function (pdf) of observations into a discrete set of small cells and symbols $\mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_M$ one symbol representing each cell. This partitioning subject is usually called *vector quantization* and there exists several analysis methods for this topic [1]. After a vector quantization is performed, a codebook is created of the mean vectors for every cluster.

The corresponding symbol for the observation is determined by the nearest neighbor rule, i.e. select the symbol of the cell with the nearest codebook vector. To make a parallel to the urn and ball model, this means that if a dark gray ball is observed, will it probably be closest to the black color. In this case the symbols $\mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_M$ are represented by one color each (e.g. $\mathbf{v}_1$ =red). The observation symbol probability distribution, $B = \{b_j(\mathbf{o}_t)\}_{j=1}^{N}$ will now have the symbol distribution in state j, $b_j(\mathbf{o}_t)$, defined as:

$$b_j(\mathbf{o}_t) = b_j(k) = P\left(\mathbf{o}_t = \mathbf{v}_k | q_t = j\right), \quad 1 \leq k \leq M \tag{4.12}$$

The estimation of the probabilities $b_j(k)$ is normally accomplished in two steps, first the determination of the codebook and then the estimation of the sets of observation probabilities for each codebook vector in each state. The major problem

with discrete output probability is that the vector quantization operation partitions the continuous acoustic space into separate regions destroying the original signal structure. As the output distributions of different states are typically highly overlapping the partitioning introduces small errors. The unreliability of parameter estimates due to the finiteness of the training data prevents the presentation of very large codebooks. This introduces quantization errors which may cause degradation in the discrete HMM performance when the observed feature vectors are intermediate between two codebook symbols [1]. This is why *continuous observation densities* can be used instead to increase the recognition rate, but this approach also gives that more calculations is needed.

4.2.3 Continuous Observation Densities

To create continuous observation density HMMs, $b_j(\mathbf{o}_t)$:s are created as some parametric probability density functions or mixtures of them. To be able to reestimate the parameters of the probability density function (pdf) some restriction must be placed on the model for the pdf. This restriction is that the pdf is any log-concave or elliptically symmetric density [1]. The most general representation of the pdf, for which a reestimation procedure has been formulated, is a finite mixture of the form:

$$b_j(\mathbf{o}_t) = \sum_{k=1}^{M} c_{jk} b_{jk}(\mathbf{o}_t), \quad j = 1, 2, \ldots, N \tag{4.13}$$

Where M is the number of mixtures and the following stochastic constraints for the mixture weights, c_{jk}, holds:

$$\begin{aligned} &\sum_{k=1}^{M} c_{jk} = 1 \quad && j = 1, 2, \ldots, N \\ &c_{jk} \geq 0 && j = 1, 2, \ldots, N, \ k = 1, 2, \ldots, M \end{aligned} \tag{4.14}$$

And $b_{jk}(\mathbf{o}_t)$ is a D-dimensional log-concave or elliptically symmetric density with mean vector $\boldsymbol{\mu}_{jk}$ and covariance matrix $\boldsymbol{\Sigma}_{jk}$:

$$b_{jk}(\mathbf{o}_t) = \mathcal{N}\left(\mathbf{o}_t, \boldsymbol{\mu}_{jk}, \boldsymbol{\Sigma}_{jk}\right) \tag{4.15}$$

The most used D-dimensional log-concave or elliptically symmetric density, is the Gaussian density. The Gaussian density can be found as:

$$b_{jk}(\mathbf{o}_t) = \mathcal{N}\left(\mathbf{o}_t, \boldsymbol{\mu}_{jk}, \boldsymbol{\Sigma}_{jk}\right) = \frac{1}{(2\pi)^{D/2}|\boldsymbol{\Sigma}_{jk}|^{1/2}} e^{-\frac{1}{2}\left(\mathbf{o}_t - \boldsymbol{\mu}_{jk}\right)^T \boldsymbol{\Sigma}_{jk}^{-1} \left(\mathbf{o}_t - \boldsymbol{\mu}_{jk}\right)} \tag{4.16}$$

To approximate simple observation sources, the mixture Gaussians provide an easy way to gain a considerable accuracy due to the flexibility and convenient

estimation of the pdfs. If the observation source generates a complicated high-dimensional pdf, the mixture Gaussians become computationally difficult to treat, due to the excessive number of parameters and large covariance matrices.

When huge number of parameters are to be estimated, the practical problem is the availability of the necessary amount of well representative training data. With insufficient training data some parameters will get more or less arbitrary values and especially for the covariance matrices, this may have drastical consequences.

As the length of the feature vectors are increased, the size of the covariance matrices increases in square proportional to the vector dimension. If feature vectors are designed to avoid redundant components, the off diagonal elements of the covariance matrices are usually small. This suggest to the covariance approximation by diagonal matrices. The diagonality also provides a simpler and faster implementation for the probability computation reducing (4.16) to:

$$\begin{aligned} b_{jk}(\mathbf{o}_t) &= \mathcal{N}\left(\mathbf{o}_t, \boldsymbol{\mu}_{jk}, \boldsymbol{\Sigma}_{jk}\right) = \frac{1}{(2\pi)^{D/2}|\boldsymbol{\Sigma}_{jk}|^{1/2}} e^{-\frac{1}{2}\left(\mathbf{o}_t - \boldsymbol{\mu}_{jk}\right)^T \boldsymbol{\Sigma}_{jk}^{-1}\left(\mathbf{o}_t - \boldsymbol{\mu}_{jk}\right)} \\ &= \frac{1}{(2\pi)^{D/2}\left(\prod_{l=1}^{D}\sigma_{jkl}\right)^{1/2}} e^{-\sum_{l=1}^{D}\frac{\left(\mathbf{o}_{tl} - \boldsymbol{\mu}_{jkl}\right)^2}{2\sigma_{jkl}^2}} \end{aligned} \tag{4.17}$$

Where the variance terms $\sigma_{jk1}, \sigma_{jk2}, \ldots, \sigma_{jkD}$ are the diagonal elements of the covariance matrix $\boldsymbol{\Sigma}_{jk}$.

4.2.4 Types of Hidden Markov Models

Different kinds of structures for HMMs can be used. The structure is defined by the transition matrix, $\mathbf{A}$. The most general structure is the *ergodic* or fully connected HMM. In this model can every state be reached from every other state of the model. As shown in Fig. 4.3a, for an $N = 4$ state model, this model has the property $0 < a_{ij} < 1$ (the zero and the one has to be excluded, otherwise is the ergodic property not fulfilled). The state transition matrix, $\mathbf{A}$, for an ergodic model, can be described by:

$$\mathbf{A} = \begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ a_{31} & a_{32} & a_{33} & a_{34} \\ a_{41} & a_{42} & a_{43} & a_{44} \end{bmatrix}_{4\times 4} \tag{4.18}$$

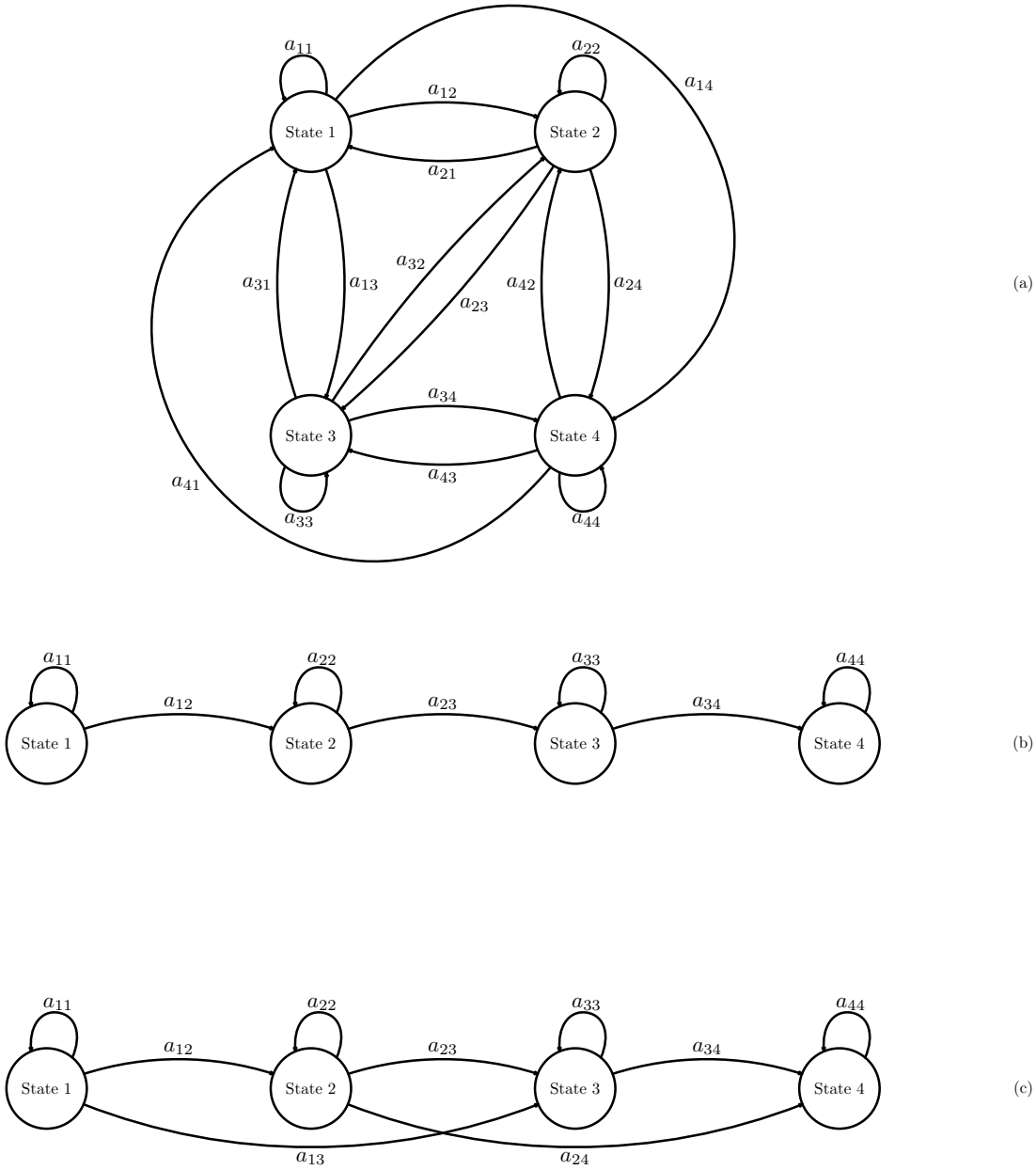

Figure 4.3: Different structures for HMMs

In speech recognition, it is desirable to use a model which models the observations in a successive manner - since this is the property of speech. The models that fulfills this modelling technique, is the left-right model or Bakis model, see Figs. 4.3b,c. The property for a left-right model is:

$$a_{ij} = 0, \quad j < i \tag{4.19}$$

That is, no jumps can be made to a previous states. The lengths of the transitions are usually restricted to some maximum length, typical two or three:

$$a_{ij} = 0, \quad j > i + \Delta \tag{4.20}$$

Note that, for a left-right model, the state transitions coefficients for the last state has the following property:

$$\begin{aligned} a_{NN} &= 1 \\ a_{Nj} &= 0, \quad j < N \end{aligned} \tag{4.21}$$

In Fig. 4.3b and Fig. 4.3c two left-right models are presented. In Fig. 4.3b is $\Delta = 1$ and the state transition matrix, $\mathbf{A}$, will be:

$$\mathbf{A} = \begin{bmatrix} a_{11} & a_{12} & 0 & 0 \\ 0 & a_{22} & a_{23} & 0 \\ 0 & 0 & a_{33} & a_{34} \\ 0 & 0 & 0 & a_{44} \end{bmatrix}_{4\times4} \tag{4.22}$$

And in Fig. 4.3c is $\Delta = 2$ and the state transition matrix, $\mathbf{A}$, will be:

$$\mathbf{A} = \begin{bmatrix} a_{11} & a_{12} & a_{13} & 0 \\ 0 & a_{22} & a_{23} & a_{24} \\ 0 & 0 & a_{33} & a_{34} \\ 0 & 0 & 0 & a_{44} \end{bmatrix}_{4\times4} \tag{4.23}$$

The choice of constrained model structure like a left-right model requires no modification in training algorithms (described later in Problem 3 - Parameter Estimation). This because any state probability set to zero will remain zero in the training algorithms.

4.2.5 Summary of elements for an Hidden Markov Model

The elements of a discrete-time hidden Markov model will now be summarized. These elements will be used throughout the thesis:

1. Number of states N. Although the states are hidden, for many practical applications there is often some physical significance attached to the states or to sets of states of the model [1]. For instance in the urn and ball model,

the states corresponds to the urns. The labels for the individual states is $\{1, 2, \ldots, N\}$, and the state at time t is denoted q_t.

2. Model parameter M. If discrete observation densities are used, the parameter M is the number of classes or cells that should be used, e.g. M equals the number of colors in the urn and ball example. If continuous observation densities are used, M is represented by the number of mixtures in every state.

3. Initial state distribution $\boldsymbol{\pi} = \{\pi_i\}_{i=1}^{N}$, in which π_i is defined as:

$$\pi_i = P(q_1 = i) \tag{4.24}$$

4. State transition probability distribution $\mathbf{A} = [a_{ij}]$ where:

$$a_{ij} = P\left(q_{t+1} = j | q_t = i\right), \quad 1 \leq i, j \leq N \tag{4.25}$$

5. Observation symbol probability distribution, $\mathbf{B} = \{b_j(\mathbf{o}_t)\}_{j=1}^{N}$, in which the probabilistic function for each state, j, is:

$$b_j(\mathbf{o}_t) = P\left(\mathbf{o}_t | q_t = j\right) \tag{4.26}$$

The calculation of $b_j(\mathbf{o}_t)$ can be found with discrete- or continuous observation densities. In this thesis, are the continuous observation densities used.

It should now be clear that a complete specification of an HMM requires two model parameters N and M. The specification of the three sets of probability measures $\boldsymbol{\pi}$, $\mathbf{A}$ and $\mathbf{B}$ are also necessary. For convenience will these probability measures use the notation, λ:

$$\lambda = (\mathbf{A}, \mathbf{B}, \boldsymbol{\pi}) \tag{4.27}$$

4.3 Three Basic Problems for Hidden Markov Models

Given the basics of an HMM from the previous section, three basic problems arise for applying the model in a speech recognition task:

Problem 1

Given the observation sequence $\mathbf{O} = (\mathbf{o}_1, \mathbf{o}_2, \ldots, \mathbf{o}_T,)$, and the model $\lambda = (\mathbf{A}, \mathbf{B}, \boldsymbol{\pi})$, how is the probability of the observation sequence,

given the model, computed? That is, how is $P(\mathbf{O}|\lambda)$ computed efficiently?

Problem 2

Given the observation sequence $\mathbf{O} = (\mathbf{o}_1, \mathbf{o}_2, \ldots, \mathbf{o}_T,)$, and the model $\lambda = (\mathbf{A}, \mathbf{B}, \boldsymbol{\pi})$, how is a corresponding state sequence, $\mathbf{q} = (q_1, q_2, \ldots, q_T)$, chosen to be optimal in some sense (i.e., best "explains" the observations)?

Problem 3

How are the probability measures, $\lambda = (\mathbf{A}, \mathbf{B}, \boldsymbol{\pi})$, adjusted to maximize $P(\mathbf{O}|\lambda)$?

The first problem can be seen as the recognition problem. With some trained models, each model represents a word, which model is the most likely if an observation is given (i.e. what word is spoken)? In the second problem the hidden part of the model is attempted to be uncovered. It should be clear, that for all except the case of degenerated models, there is no "correct" state sequence to be found. Thereby it is a problem to be solved best possible with some optimal criteria. The third problem can be seen as the training problem. That is given the training sequences, create a model for each word. The training problem is the crucial one for most applications of HMMs, because it will optimally adapt the model parameters to observed training data - i.e., it will create the best models for real phenomena [1].

4.4 Solution to Problem 1 - Probability Evaluation

The aim of this problem is to find the probability of the observation sequence, $\mathbf{O} = (\mathbf{o}_1, \mathbf{o}_2, \ldots, \mathbf{o}_T,)$, given the model λ, i.e., $P(\mathbf{O}|\lambda)$. Because the observations produced by states are assumed to be independent of each other and the time t, the probability of observation sequence $\mathbf{O} = (\mathbf{o}_1, \mathbf{o}_2, \ldots, \mathbf{o}_T)$ being generated by a certain state sequence $\mathbf{q}$ can be calculated by a product:

$$P(\mathbf{O}|\mathbf{q}, B) = b_{q_1}(\mathbf{o}_1) \cdot b_{q_2}(\mathbf{o}_2) \cdot \ldots \cdot b_{q_T}(\mathbf{o}_T) \tag{4.28}$$

And the probability of the state sequence, $\mathbf{q}$ can be found as[1]:

$$P(\mathbf{q}|\mathbf{A}, \boldsymbol{\pi}) = \pi_{q_1} \cdot a_{q_1 q_2} \cdot a_{q_2 q_3} \cdot \ldots \cdot a_{q_{T-1} q_T} \tag{4.29}$$

The joint probability of $\mathbf{O}$ and $\mathbf{q}$, i.e., the probability that $\mathbf{O}$ and $\mathbf{q}$ occur simultaneously, is simply the product of the above two terms, i.e.:

[1] Explained earlier in Sec. 4.1.1

$$\begin{aligned} P(\mathbf{O}, \mathbf{q}|\lambda) &= P(\mathbf{O}|\mathbf{q}, B) \cdot P(\mathbf{q}|\mathbf{A}, \boldsymbol{\pi}) \\ &= \pi_{q_1} b_{q_1}(\mathbf{o}_1) a_{q_1 q_2} b_{q_2}(\mathbf{o}_2) \cdot \ldots \cdot a_{q_{T-1} q_T} b_{q_T}(\mathbf{o}_T) \\ &= \pi_{q_1} \prod_{t=2}^{T} a_{q_{t-1} q_t} b_{q_t}(\mathbf{o}_t) \end{aligned} \tag{4.30}$$

The aim was to find $P(\mathbf{O}|\lambda)$, and this probability of $\mathbf{O}$ (given the model λ) is obtained by summing the joint probability over all possible state sequences $\mathbf{q}$, giving:

$$\begin{aligned} P(\mathbf{O}|\lambda) &= \sum_{\text{all } \mathbf{q}} P(\mathbf{O}|\mathbf{q}, B) \cdot P(\mathbf{q}|\mathbf{A}, \boldsymbol{\pi}) \\ &= \sum_{q_1, q_2, \ldots, q_T} \pi_{q_1} b_{q_1}(\mathbf{o}_1) a_{q_1 q_2} b_{q_2}(\mathbf{o}_2) \cdot \ldots \cdot a_{q_{T-1} q_T} b_{q_T}(\mathbf{o}_T) \\ &= \sum_{q_1, q_2, \ldots, q_T} \pi_{q_1} \prod_{t=2}^{T} a_{q_{t-1} q_t} b_{q_t}(\mathbf{o}_t) \end{aligned} \tag{4.31}$$

The interpretation of the computation in 4.31 is the following. Initially at time $t = 1$ the process starts by jumping to state q_1 with probability π_{q_1}, and generate the observation symbol $\mathbf{o}_1$ with probability $b_{q_1}(\mathbf{o}_1)$. The clock changes from t to $t+1$ and a transition from q_1 to q_2 will occur with probability $a_{q_1 q_2}$, and the symbol $\mathbf{o}_2$ will be generated with probability $b_{q_2}(\mathbf{o}_2)$. The process continues in this manner until the last transition is made (at time T), i.e., a transition from q_{T-1} to q_T will occur with probability $a_{q_{T-1} q_T}$, and the symbol $\mathbf{o}_T$ will be generated with probability $b_{q_T}(\mathbf{o}_T)$.

This direct computation has one major drawback. It is infeasible due to the exponential growth of computations as a function of sequence length T. To be precise, it needs $(2T-1)N^T$ multiplications, and $N^T - 1$ additions [1]. Even for small values of N and T; e.g., for $N = 5$ (states), $T = 100$ (observations), there is a need for $(2 \cdot 100 - 1)5^{100} \approx 1.6 \cdot 10^{72}$ multiplications and $5^{100} - 1 \approx 8.0 \cdot 10^{69}$ additions! Clearly a more efficient procedure is required to solve this problem. An excellent tool which cuts the computational requirements to linear, relative to T, is the well known forward algorithm.

4.4.1 The Forward Algorithm

Consider a forward variable $\alpha_t(i)$, defined as:

$$\alpha_t(i) = P(\mathbf{o}_1 \mathbf{o}_2 \ldots \mathbf{o}_t, q_t = i|\lambda) \tag{4.32}$$

Where t represents time and i is the state. This gives that $\alpha_t(i)$ will be the probability of the partial observation sequence, $\mathbf{o}_1\mathbf{o}_2 \ldots \mathbf{o}_t$, (until time t) when being in state i at time t. The forward variable can be calculated inductively, see Fig. 4.4.

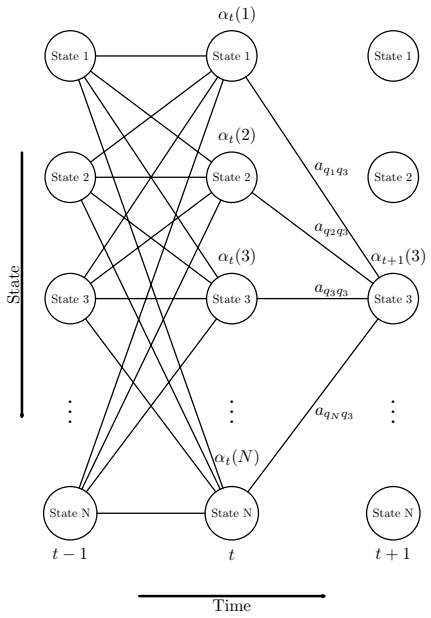

Figure 4.4: Forward Procedure - Induction Step

$\alpha_{t+1}(i)$ is found by summing the forward variable for all N states at time t multiplied with their corresponding state transition probability, a_{ij}, and by the emission probability $b_j(\mathbf{o}_t + 1)$. This can be done with the following procedure:

1. **Initialization**

$$\begin{aligned} \text{Set } t &= 1; && (4.33) \\ \alpha_1(i) &= \pi_i b_i(\mathbf{o}_1), \quad 1 \leq i \leq N && (4.34) \end{aligned}$$

2. **Induction**

$$\alpha_{t+1}(j) = b_j(\mathbf{o}_{t+1}) \sum_{i=1}^{N} \alpha_t(i) a_{ij}, \quad 1 \leq j \leq N \tag{4.35}$$

3. **Update time**

   Set $t = t + 1$;
   Return to step 2 if $t < T$;
   Otherwise, terminate the algorithm (goto step 4).

4. **Termination**

$$P(\mathbf{O}|\lambda) = \sum_{i=1}^{N} \alpha_T(i) \tag{4.36}$$

If the forward algorithm is used there is a need for $N(N+1)(T-1)+N$ multiplications and $N(N-1)(T-1)$ additions. Again for $N = 5$ (states), $T = 100$ (observations), this yields $5(5+1)(100-1)+5 = 2975$ multiplications and $5(5-1)(100-1) = 1980$ additions. This is quite an improvement compared to the direct method (10^{72} multiplications and 10^{69} additions).

4.4.2 The Backward Algorithm

The recursion described in the forward algorithm, can also be done in the reverse time. By defining the backward variable $\beta_t(i)$ as:

$$\beta_t(i) = P\left(\mathbf{o}_{t+1}\mathbf{o}_{t+2}\ldots\mathbf{o}_T|q_t = i, \lambda\right) \tag{4.37}$$

That is, the probability of the partial observation sequence from $t+1$ to the end, given state i at time t and the model λ. Notice that the definition for the forward variable is a joint probability whereas the backward probability is a conditional probability. In a similar manner (according to the forward algorithm), can the backward be calculated inductively, see Fig. 4.5.

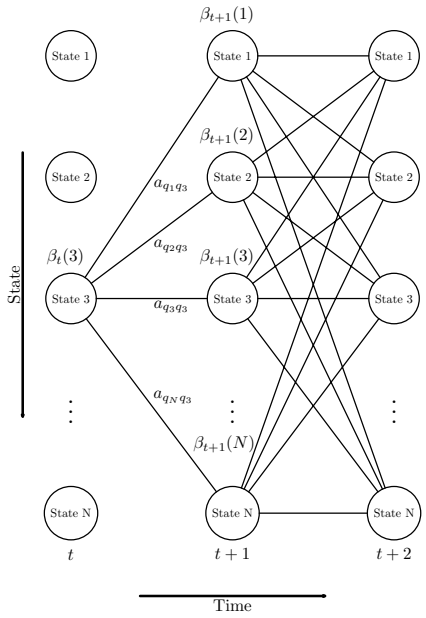

Figure 4.5: Backward Procedure - Induction Step

The backward algorithm includes the following steps:

1. **Initialization**

$$\begin{aligned} \text{Set } t &= T-1; \\ \beta_T(i) &= 1, \quad 1 \leq i \leq N \end{aligned} \tag{4.38}$$

2. **Induction**

$$\beta_t(i) = \sum_{j=1}^{N} \beta_{t+1}(i) a_{ij} b_j(\mathbf{o}_{t+1}), \quad 1 \leq i \leq N \tag{4.39}$$

3. **Update time**

Set $t = t - 1$;
Return to step 2 if $t > 0$;
Otherwise, terminate the algorithm.

Note that the initialization step 1 *arbitrarily* defines $\beta_T(i)$ to be 1 for all i.

4.4.3 Scaling the Forward and Backward Variables

The calculation of $\alpha_t(i)$ and $\beta_t(i)$ involves multiplication with probabilities. All these probabilities have a value less than 1 (generally significantly less than 1), and as t starts to grow large, each term of $\alpha_t(i)$ or $\beta_t(i)$ starts to head exponentially to zero. For sufficiently large t (e.g., 100 or more) the dynamic range of $\alpha_t(i)$ and $\beta_t(i)$ computation will exceed the precision range of essentially any machine (even in double precision) [1]. The basic scaling procedure multiplies $\alpha_t(i)$ by a scaling coefficient that is dependent only of the time t and independent of the state i. The scaling factor for the forward variable is denoted c_t (scaling is done every time t for all states i - $1 \leq i \leq N$) . This factor will also be used for scaling the backward variable, $\beta_t(i)$. Scaling $\alpha_t(i)$ and $\beta_t(i)$ with the same scale factor will show useful in problem 3 (parameter estimation).

Consider the computation of the forward variable, $\alpha_t(i)$. In the scaled variant of the forward algorithm some extra notations will be used. $\alpha_t(i)$ denote the unscaled forward variable, $\hat{\alpha}_t(i)$ denote the scaled and iterated variant of $\alpha_t(i)$, $\widehat{\hat{\alpha}}_t(i)$ denote the local version of $\alpha_t(i)$ before scaling and c_t will represent the scaling coefficient at each time. Here follows the scaled forward algorithm:

1. **Initialization**

$$
\begin{aligned}
\text{Set } t &= 2; \\
\alpha_1(i) &= \pi_i b_i(\mathbf{o}_1), \quad 1 \leq i \leq N && (4.40) \\
\widehat{\hat{\alpha}}_1(i) &= \alpha_1(i), \quad 1 \leq i \leq N && (4.41) \\
c_1 &= \frac{1}{\sum_{i=1}^{N} \alpha_1(i)} && (4.42) \\
\hat{\alpha}_1(i) &= c_1 \alpha_1(i) && (4.43)
\end{aligned}
$$

2. **Induction**

$$
\begin{aligned}
\widehat{\hat{\alpha}}_t(i) &= b_i(\mathbf{o}_t) \sum_{j=1}^{N} \hat{\alpha}_{t-1}(j) a_{ji}, \quad 1 \leq i \leq N && (4.44) \\
c_t &= \frac{1}{\sum_{i=1}^{N} \widehat{\hat{\alpha}}_t(i)} && (4.45) \\
\hat{\alpha}_t(i) &= c_t \widehat{\hat{\alpha}}_t(i), \quad 1 \leq i \leq N && (4.46)
\end{aligned}
$$

3. **Update time**

Set $t = t + 1$;
Return to step 2 if $t \leq T$;
Otherwise, terminate the algorithm (goto step 4).

4. **Termination**

$$
\log P(\mathbf{O}|\lambda) = -\sum_{t=1}^{T} \log c_t \tag{4.47}
$$

The main difference between the scaled and the none scaled forward algorithm lies in steps two and four. In step two can (4.46) be rewritten if (4.44) and (4.45) are used:

$$\begin{aligned}
\hat{\alpha}_t(i) &= c_t\hat{\widehat{\alpha}}_t(i) \\
&= \frac{1}{\sum_{k=1}^{N}\left(b_k(\mathbf{o}_t)\sum_{j=1}^{N}\hat{\alpha}_{t-1}(j)a_{jk}\right)}\cdot\left(b_i(\mathbf{o}_t)\sum_{j=1}^{N}\hat{\alpha}_{t-1}(j)a_{ji}\right) \\
&= \frac{b_i(\mathbf{o}_t)\sum_{j=1}^{N}\hat{\alpha}_{t-1}(j)a_{ji}}{\sum_{k=1}^{N}b_k(\mathbf{o}_t)\sum_{j=1}^{N}\hat{\alpha}_{t-1}(j)a_{jk}}, \quad 1 \le i \le N
\end{aligned} \tag{4.48}$$

By induction, the scaled forward variable can be found in terms of the none scaled as:

$$\hat{\alpha}_{t-1}(j) = \left(\prod_{\tau=1}^{t-1} c_\tau\right)\alpha_{t-1}(j), \quad 1 \le j \le N \tag{4.49}$$

The ordinary induction step can be found as (same as (4.35) but with one time unit shift):

$$\alpha_t(i) = b_i(\mathbf{o}_t)\sum_{j=1}^{N}\alpha_{t-1}(j)a_{ji}, \quad 1 \le i \le N \tag{4.50}$$

With (4.49) and (4.50) it is now possible to rewrite (4.48) as:

$$
\begin{aligned}
\hat{\alpha}_t(i) &= \frac{b_i(\mathbf{o}_t)\sum_{j=1}^{N}\hat{\alpha}_{t-1}(j)a_{ji}}{\sum_{k=1}^{N}b_k(\mathbf{o}_t)\sum_{j=1}^{N}\hat{\alpha}_{t-1}(j)a_{jk}} \\
&= \frac{b_i(\mathbf{o}_t)\sum_{j=1}^{N}\left(\prod_{\tau=1}^{t-1}c_\tau\right)\alpha_{t-1}(j)a_{ji}}{\sum_{k=1}^{N}b_k(\mathbf{o}_t)\sum_{j=1}^{N}\left(\prod_{\tau=1}^{t-1}c_\tau\right)\alpha_{t-1}(j)a_{jk}} \\
&= \frac{\left(\prod_{\tau=1}^{t-1}c_\tau\right)\left(b_i(\mathbf{o}_t)\sum_{j=1}^{N}\alpha_{t-1}(j)a_{ji}\right)}{\left(\prod_{\tau=1}^{t-1}c_\tau\right)\sum_{k=1}^{N}\left(b_k(\mathbf{o}_t)\sum_{j=1}^{N}\alpha_{t-1}(j)a_{jk}\right)} \\
&= \frac{\alpha_t(i)}{\sum_{k=1}^{N}\alpha_t(k)}, \quad 1 \le i \le N
\end{aligned}
\tag{4.51}
$$

As (4.51) shows, each $\alpha_t(i)$ is scaled by the sum over all states of $\alpha_t(i)$ when the scaled forward algorithm is applied.

The termination (step 4) of the scaled forward algorithm, evaluation of $P(\mathbf{O}|\lambda)$, must be done in a different way. This because the sum of $\hat{\alpha}_T(i)$ can not be used, because $\hat{\alpha}_T(i)$ is scaled already. However the following properties can be used:

$$\prod_{\tau=1}^{T}c_\tau\sum_{i=1}^{N}\alpha_T(i) = 1 \tag{4.52}$$

$$\prod_{\tau=1}^{T}c_\tau \cdot P(\mathbf{O}|\lambda) = 1 \tag{4.53}$$

$$P(\mathbf{O}|\lambda) = \frac{1}{\prod_{\tau=1}^{T}c_\tau} \tag{4.54}$$

As (4.54) shows can $P(\mathbf{O}|\lambda)$ be found, but the problem is that if (4.54) is used the result will still be very small (and probable out of the dynamic range for a computer). If the logarithm is taken on both sides the following equation can be used:

$$\log P(\mathbf{O}|\lambda) = \frac{1}{\prod_{\tau=1}^{T} c_\tau} = -\sum_{t=1}^{T} \log c_t \tag{4.55}$$

This is exactly what is done in the termination step of the scaled forward algorithm. The logarithm of $P(\mathbf{O}|\lambda)$ is often just as useful as $P(\mathbf{O}|\lambda)$, because in most cases, this measure is used as comparison with other probabilities (for other models).

The scaled backward algorithm can be found more easily, since it will use the same scale factor as the forward algorithm. The notations used is similar to the forward variables notations, $\beta_t(i)$ denote the unscaled backward variable, $\hat{\beta}_t(i)$ denote the scaled and iterated variant of $\beta_t(i)$, $\hat{\hat{\beta}}_t(i)$ denote the local version of $\beta_t(i)$ before scaling and c_t will represent the scaling coefficient at each time. Here follows the scaled backward algorithm:

1. **Initialization**

$$\begin{aligned} \text{Set } t &= T-1; \\ \beta_T(i) &= 1, \quad 1 \le i \le N \qquad (4.56) \\ \hat{\beta}_T(i) &= c_T \beta_T(i), \quad 1 \le i \le N \qquad (4.57) \end{aligned}$$

2. **Induction**

$$\begin{aligned} \hat{\hat{\beta}}_t(i) &= \sum_{j=1}^{N} \hat{\beta}_{t+1}(j) a_{ij} b_j(\mathbf{o}_{t+1}), \quad 1 \le i \le N \qquad (4.58) \\ \hat{\beta}_t(i) &= c_t \hat{\hat{\beta}}_t(i), \quad 1 \le i \le N \qquad (4.59) \end{aligned}$$

3. **Update time**

   Set $t = t - 1$;
   Return to step 2 if $t > 0$;
   Otherwise, terminate the algorithm.

4.5 Solution to Problem 2 - "Optimal" State Sequence

The problem is to find the optimal sequence of states to a given observation sequence and model. Unlike problem one, for which an exact solution can be

found, there are several possible ways of solving this problem. The difficulty lies with the definition of the optimal state sequence, that is, there are several possible optimality criteria [1]. One optimal criteria is to choose the states, q_t, that are *individually* most likely at each time t. To find this state sequence a the following probability variable is needed:

$$\gamma_t(i) = P\left(q_t = i|\mathbf{O}, \lambda\right) \tag{4.60}$$

That is, the probability of being in state i at time t given the observation sequence, $\mathbf{O}$, and the model λ. Other ways to look at $\gamma_t(i)$ can be:

$$\begin{aligned} \gamma_t(i) &= P\left(q_t = i|\mathbf{O}, \lambda\right) \\ &= \frac{P\left(\mathbf{O}, q_t = i|\lambda\right)}{P\left(\mathbf{O}|\lambda\right)} \\ &= \frac{P\left(\mathbf{O}, q_t = i|\lambda\right)}{\sum_{i=1}^{N} P\left(\mathbf{O}, q_t = i|\lambda\right)} \end{aligned} \tag{4.61}$$

And since $\alpha_t(i) = P\left(\mathbf{o}_1\mathbf{o}_2 \ldots \mathbf{o}_t, q_t = i|\lambda\right)$ and $\beta_t(i) = P\left(\mathbf{o}_{t+1}\mathbf{o}_{t+2} \ldots \mathbf{o}_T|q_t = i, \lambda\right)$, can $P\left(\mathbf{O}, q_t = i|\lambda\right)$ be found as a joint probability:

$$P\left(\mathbf{O}, q_t = i|\lambda\right) = P\left(\mathbf{o}_1\mathbf{o}_2 \ldots \mathbf{o}_t, q_t = i|\lambda\right) \cdot P\left(\mathbf{o}_{t+1}\mathbf{o}_{t+2} \ldots \mathbf{o}_T|q_t = i, \lambda\right) \tag{4.62}$$

With (4.62) it is now possible to rewrite (4.61) as:

$$\gamma_t(i) = \frac{\alpha_t(i)\beta_t(i)}{\sum_{i=1}^{N} \alpha_t(i)\beta_t(i)} \tag{4.63}$$

When $\gamma_t(i)$ is calculated according to (4.63), the most likely state at time t, q_t^*, will be found by:

$$q_t^* = \arg\max_{1 \leq i \leq N} \left[\gamma_t(i)\right], \quad 1 \leq t \leq T \tag{4.64}$$

Even if (4.64) maximizes the expected number of correct states, there could be some problems with the resulting state sequence. This because the state transition probabilities have not been taken into account. For example what happens when some state transitions have zero probability ($a_{ij} = 0$)? This means that the found optimal path may not be valid. Obviously a method generating a path that is guaranteed to be valid would be preferably. Fortunately such a method exist, based on dynamic programming, namely *the Viterbi algorithm*. Even though $\gamma_t(i)$ could not be used for this purpose, it will be useful in problem 3 (parameter estimation).

4.5.1 The Viterbi Algorithm

This algorithm is similar to the forward algorithm. The main difference is that the forward algorithm uses summing over previous states, whereas the Viterbi algorithm uses maximization. The aim for the Viterbi algorithm is to find the single best state sequence, $\mathbf{q} = (q_1, q_2, \ldots, q_T)$, for the given observation sequence $\mathbf{O} = (\mathbf{o}_1, \mathbf{o}_2, \ldots, \mathbf{o}_T)$ and a model λ. Consider the following quantity:

$$\delta_t(i) = \max_{q_1, q_2, \ldots, q_{t-1}} P\left(q_1 q_2 \ldots q_{t-1}, q_t = i, \mathbf{o}_1 \mathbf{o}_2 \ldots \mathbf{o}_t | \lambda\right) \tag{4.65}$$

That is the probability of observing $\mathbf{o}_1 \mathbf{o}_2 \ldots \mathbf{o}_t$ using the best path that ends in state i at time t, given the model λ. By using induction can $\delta_{t+1}(i)$ be found as:

$$\delta_{t+1}(i) = b_j(\mathbf{o}_{t+1}) \max_{1 \le i \le N} [\delta_t(i) a_{ij}] \tag{4.66}$$

To actually retrieve the state sequence, it is necessary to keep track of the argument that maximizes (4.66), for each t and j . This is done by saving the argument in an array $\psi_t(j)$. Here follows the complete Viterbi algorithm:

1. **Initialization**

$$\begin{aligned} \text{Set } t &= 2; \\ \delta_1(i) &= \pi_i b_i(\mathbf{o}_1), \quad 1 \le i \le N && (4.67) \\ \psi_1(i) &= 0, \quad 1 \le i \le N && (4.68) \end{aligned}$$

2. **Induction**

$$\begin{aligned} \delta_t(j) &= b_j(\mathbf{o}_t) \max_{1 \le i \le N} \delta_{t-1}(i) a_{ij}, \quad 1 \le j \le N && (4.69) \\ \psi_t(j) &= \arg \max_{1 \le i \le N} \left[\delta_{t-1}(i) a_{ij}\right], \quad 1 \le j \le N && (4.70) \end{aligned}$$

3. **Update time**

   Set $t = t + 1$;
   Return to step 2 if $t \le T$;
   Otherwise, terminate the algorithm (goto step 4).

4. **Termination**

$$\begin{aligned} P^* &= \max_{1 \le i \le N} [\delta_T(i)] && (4.71) \\ q_T^* &= \arg \max_{1 \le i \le N} [\delta_T(i)] && (4.72) \end{aligned}$$

5. **Path (state sequence) backtracking**

   (a) **Initialization**

   Set $t = T - 1$

   (b) **Backtracking**

$$q_t^* = \psi_{t+1}\left(q_{t+1}^*\right) \tag{4.73}$$

   (c) **Update time**

   Set $t = t - 1$;
   Return to step (b) if $t \geq 1$;
   Otherwise, terminate the algorithm.

The same problem as for the forward and backward algorithm occurs here. That is the algorithm involves multiplication with probabilities and the precision range will be exceeded. This is why an alternative Viterbi algorithm is needed.

4.5.2 The Alternative Viterbi Algorithm

As mentioned the original Viterbi algorithm involves multiplications with probabilities. One way to avoid this is to take the logarithm of the model parameters, giving that the multiplications become additions. Obviously will this logarithm become a problem when some model parameters has zeros present. This is often the case for $\mathbf{A}$ and $\boldsymbol{\pi}$ and can be avoided by adding a small number to the matrixes. Here follows the alternative Viterbi algorithm:

1. **Preprocessing**

$$\begin{aligned} \widetilde{\pi}_i &= \log(\pi_i), \quad 1 \leq i \leq N && (4.74)\\ \widetilde{a}_{ij} &= \log(a_{ij}), \quad 1 \leq i, j \leq N && (4.75) \end{aligned}$$

2. **Initialization**

$$\begin{aligned} \text{Set } t &= 2; \\ \widetilde{b}_i(\mathbf{o}_1) &= \log(b_i(\mathbf{o}_1)), \quad 1 \leq i \leq N && (4.76)\\ \widetilde{\delta}_1(i) &= \log(\delta_1(i)) \\ &= \log(\pi_i b_i(\mathbf{o}_1)) \\ &= \widetilde{\pi}_i + \widetilde{b}_i(\mathbf{o}_1), \quad 1 \leq i \leq N && (4.77)\\ \psi_1(i) &= 0, \quad 1 \leq i \leq N && (4.78) \end{aligned}$$

3. **Induction**

$$\begin{aligned}
\widetilde{b}_j(\mathbf{o}_t) &= \log\left(b_j(\mathbf{o}_t)\right), \quad 1 \leq j \leq N && (4.79)\\
\widetilde{\delta}_t(j) &= \log\left(\delta_t(j)\right)\\
&= \log\left(b_j(\mathbf{o}_t) \max_{1\leq i\leq N} \delta_{t-1}(i)a_{ij}\right)\\
&= \widetilde{b}_j(\mathbf{o}_t) + \max_{1\leq i\leq N}\left[\widetilde{\delta}_{t-1}(i) + \widetilde{a}_{ij}\right], \quad 1 \leq j \leq N && (4.80)\\
\psi_t(j) &= \arg\max_{1\leq i\leq N}\left[\widetilde{\delta}_{t-1}(i) + \widetilde{a}_{ij}\right], \quad 1 \leq j \leq N && (4.81)
\end{aligned}$$

4. **Update time**

   Set $t = t + 1$;
   Return to step 3 if $t \leq T$;
   Otherwise, terminate the algorithm (goto step 5).

5. **Termination**

$$\begin{aligned}
\widetilde{P}^* &= \max_{1\leq i\leq N}\left[\widetilde{\delta}_T(i)\right] && (4.82)\\
q_T^* &= \arg\max_{1\leq i\leq N}\left[\widetilde{\delta}_T(i)\right] && (4.83)
\end{aligned}$$

6. **Path (state sequence) backtracking**

   (a) **Initialization**

   Set $t = T - 1$

   (b) **Backtracking**

$$q_t^* = \psi_{t+1}\left(q_{t+1}^*\right) \qquad (4.84)$$

   (c) **Update time**

   Set $t = t - 1$;
   Return to step (b) if $t \geq 1$;
   Otherwise, terminate the algorithm.

To get a better insight of how the Viterbi (and the alternative Viterbi) works, consider a model with $N = 3$ states and an observation of length $T = 8$. In the initialization ($t = 1$) is $\delta_1(1)$, $\delta_1(2)$ and $\delta_1(3)$ found. Lets assume that $\delta_1(2)$ is the maximum. Next time ($t = 2$) three variables will be used namely $\delta_2(1)$, $\delta_2(2)$ and $\delta_2(3)$. Lets assume that $\delta_2(1)$ is now the maximum. In the same manner will the following variables $\delta_3(3)$, $\delta_4(2)$, $\delta_5(2)$, $\delta_6(1)$, $\delta_7(3)$ and $\delta_8(3)$ be the maximum at their time, see Fig. 4.6.
As Fig. 4.6 shows it easy to see that the Viterbi algorithm is working with the lattice structure.

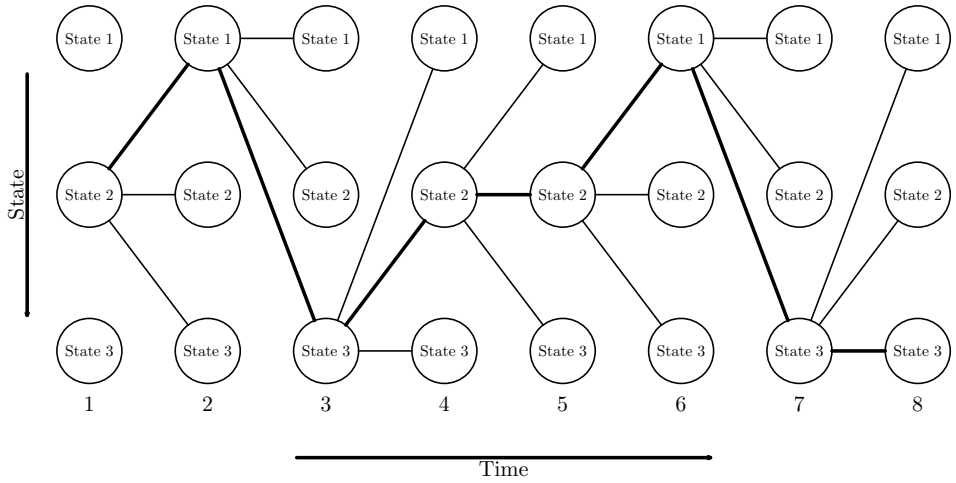

Figure 4.6: Example of Viterbi search

4.6 Solution to Problem 3 - Parameter Estimation

The third problem is concerned with the estimation of the model parameters, $\lambda = (\mathbf{A}, \mathbf{B}, \boldsymbol{\pi})$. The problem can be formulated as:

$$\lambda^* = \arg\max_{\lambda} \left[P\left(\mathbf{O}|\lambda\right) \right] \tag{4.85}$$

Given an observation $\mathbf{O}$, find the model λ^* from all possible λ that maximizes $P\left(\mathbf{O}|\lambda\right)$. This problem is the most difficult of the three problems. This because there is no known way to analytically find the model parameters that maximizes the probability of the observation sequence in a closed form. However can the model parameters be chosen to locally maximize the likelihood $P\left(\mathbf{O}|\lambda\right)$. Some common used methods for solving this problem is Baum-Welch method (also known as expectation-maximation method) or gradient technics. Both of these methods uses iterations to improve the likelihood $P\left(\mathbf{O}|\lambda\right)$, however there are some advantages with the Baum-Welch method compared to the gradient technics:

- Baum-Welch is numerically stable with the likelihood non-decreasing with every iteration.
- Baum-Welch converges to a local optima.
- Baum-Welch has linear convergence.

This is why the Baum-Welch is used in this thesis. This section will derive the reestimation equations used in the Baum-Welch method.

The model λ, has three terms to describe namely the state transition probability distribution $\mathbf{A}$, the initial state distribution $\boldsymbol{\pi}$ and the observation symbol probability distribution $\mathbf{B}$. Since continuous observation densities are used, will $\mathbf{B}$ be represented by[2] c_{jk},$\boldsymbol{\mu}_{jk}$ and $\boldsymbol{\Sigma}_{jk}$. To describe the procedure for reestimation, the following probability will prove useful:

$$\xi_t(i,j) = P\left(q_t = i, q_{t+1} = j|\mathbf{O}, \lambda\right) = \frac{P\left(q_t = i, q_{t+1} = j, \mathbf{O}|\lambda\right)}{P\left(\mathbf{O}|\lambda\right)} \tag{4.86}$$

That is the probability of being in state i at time t, and state j at time $t+1$, given the model λ and the observation sequence $\mathbf{O}$. The paths that satisfy the conditions required by (4.86) are illustrated in Fig. 4.7.

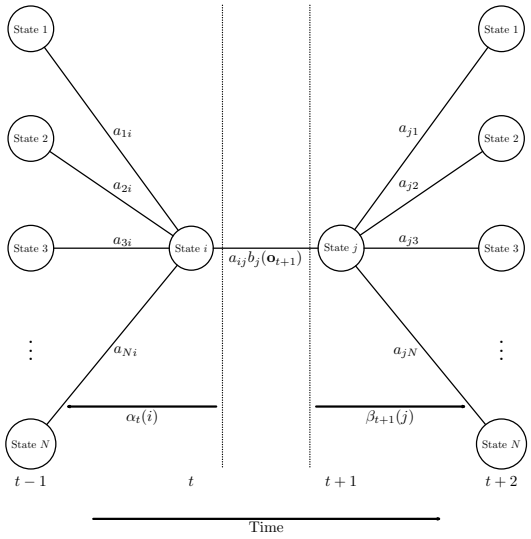

Figure 4.7: Computation of $\xi_t(i,j)$

By using (4.2), (4.11), (4.32), (4.37), (4.62) and (4.86) can $\xi_t(i,j)$ be found as:

$$\begin{aligned} \xi_t(i,j) &= \frac{\alpha_t(i)a_{ij}b_j(\mathbf{o}_{t+1})\beta_{t+1}(j)}{P\left(\mathbf{O}|\lambda\right)} \\ &= \frac{\alpha_t(i)a_{ij}b_j(\mathbf{o}_{t+1})\beta_{t+1}(j)}{\sum_{i=1}^{N}\sum_{j=1}^{N}\alpha_t(i)a_{ij}b_j(\mathbf{o}_{t+1})\beta_{t+1}(j)} \end{aligned} \tag{4.87}$$

As mentioned in problem 2 is $\gamma_t(i)$ the probability of being in state i at time t, given the entire observation sequence $\mathbf{O}$ and the model λ. Hence the relation between $\gamma_t(i)$ and $\xi_t(i,j)$ can be found by using (4.60) and (4.86):

[2]See equations 4.13-4.16

$$\gamma_t(i) = P\left(q_t = i|\mathbf{O}, \lambda\right) = \sum_{j=1}^{N} P\left(q_t = i, q_{t+1} = j|\mathbf{O}, \lambda\right) = \sum_{j=1}^{N} \xi_t(i,j) \tag{4.88}$$

If the sum over time t is applied to $\gamma_t(i)$, one will get a quantity that can be interpreted as the expected (over time) number of times that state i is visited, or equivalently, the expected number of transitions made from state i (if the time slot $t = T$ is excluded) [1]. If the same summation is done over $\xi_t(i,j)$, one will get the expected number of transitions from state i to state j. The term $\gamma_1(i)$ will also prove to be useful. Conclusion:

$$\gamma_1(i) = \text{probability of starting in state } i \tag{4.89}$$

$$\sum_{t=1}^{T-1} \gamma_t(i) = \text{expected number of transitions from state } i \text{ in } \mathbf{O} \tag{4.90}$$

$$\sum_{t=1}^{T-1} \xi_t(i,j) = \text{expected number of transitions from state } i \text{ to state } j \text{ in } \mathbf{O} \tag{4.91}$$

Given the above definitions it is possible to derive the reestimation formulas for $\boldsymbol{\pi}$ and $\mathbf{A}$:

$$\begin{aligned} \pi_i &= \text{expected frequency (number of times) in state } i \text{ at time } t = 1 \\ &= \gamma_1(i) \\ &= \frac{\alpha_1(i)\beta_1(i)}{\sum_{i=1}^{N} \alpha_1(i)\beta_1(i)} \\ &= \frac{\alpha_1(i)\beta_1(i)}{\sum_{i=1}^{N} \alpha_T(i)} \end{aligned} \tag{4.92}$$

And:

$$\begin{aligned} a_{ij} &= \frac{\text{expected number of transitions from state } i \text{ to state } j}{\text{expected number of transitions from state } i} \\ &= \frac{\sum_{t=1}^{T-1} \xi_t(i,j)}{\sum_{t=1}^{T-1} \gamma_t(i)} \\ &= \frac{\sum_{t=1}^{T-1} \alpha_t(i) a_{ij} b_j(\mathbf{o}_{t+1}) \beta_{t+1}(j)}{\sum_{t=1}^{T-1} \alpha_t(i)\beta_t(i)} \end{aligned} \tag{4.93}$$

In the derivation of (4.92) and (4.93), the following probabilities are useful to get a better understanding:

$$\begin{aligned} P(\mathbf{O}|,\lambda) &= \sum_{i=1}^{N} \alpha_t(i)\beta_t(i) = \sum_{i=1}^{N} \alpha_T(i) \\ P(\mathbf{O}, q_t = i|,\lambda) &= \alpha_t(i)\beta_t(i) \\ P(\mathbf{O}, q_t = i, q_{t+1} = j|,\lambda) &= \alpha_t(i) a_{ij} b_j(\mathbf{o}_{t+1}) \beta_{t+1}(j) \end{aligned} \tag{4.94}$$

However is (4.87) and (4.63), and thereby (4.92) and (4.93), found by using the unscaled forward and backward variables. With scaled variables will $\xi_t(i,j)$ be found as:

$$
\begin{aligned}
\xi_t(i,j) &= \frac{\hat{\alpha}_t(i)a_{ij}b_j(\mathbf{o}_{t+1})\hat{\beta}_{t+1}(j)}{\sum_{i=1}^{N}\sum_{j=1}^{N}\hat{\alpha}_t(i)a_{ij}b_j(\mathbf{o}_{t+1})\hat{\beta}_{t+1}(j)} \\
&= \frac{\left(\prod_{\tau=1}^{t}c_\tau\right)\alpha_t(i)a_{ij}b_j(\mathbf{o}_{t+1})\left(\prod_{\tau=t+1}^{T}c_\tau\right)\beta_{t+1}(j)}{\sum_{i=1}^{N}\sum_{j=1}^{N}\left(\prod_{\tau=1}^{t}c_\tau\right)\alpha_t(i)a_{ij}b_j(\mathbf{o}_{t+1})\left(\prod_{\tau=t+1}^{T}c_\tau\right)\beta_{t+1}(j)} \\
&= \frac{\left(\prod_{\tau=1}^{t}c_\tau\right)\left(\prod_{\tau=t+1}^{T}c_\tau\right)\alpha_t(i)a_{ij}b_j(\mathbf{o}_{t+1})\beta_{t+1}(j)}{\left(\prod_{\tau=1}^{t}c_\tau\right)\left(\prod_{\tau=t+1}^{T}c_\tau\right)\sum_{i=1}^{N}\sum_{j=1}^{N}\alpha_t(i)a_{ij}b_j(\mathbf{o}_{t+1})\beta_{t+1}(j)} \\
&= \frac{\alpha_t(i)a_{ij}b_j(\mathbf{o}_{t+1})\beta_{t+1}(j)}{\sum_{i=1}^{N}\sum_{j=1}^{N}\alpha_t(i)a_{ij}b_j(\mathbf{o}_{t+1})\beta_{t+1}(j)}
\end{aligned}
\tag{4.95}
$$

And the scaled $\gamma_t(i)$ will be:

$$
\begin{aligned}
\gamma_t(i) &= \frac{\hat{\alpha}_t(i)\hat{\beta}_t(i)}{\sum_{i=1}^{N}\hat{\alpha}_t(i)\hat{\beta}_t(i)} \\
&= \frac{\left(\prod_{\tau=1}^{t}c_\tau\right)\alpha_t(i)\left(\prod_{\tau=t}^{T}c_\tau\right)\beta_t(i)}{\sum_{i=1}^{N}\left(\prod_{\tau=1}^{t}c_\tau\right)\alpha_t(i)\left(\prod_{\tau=t}^{T}c_\tau\right)\beta_t(i)} \\
&= \frac{\left(\prod_{\tau=1}^{t}c_\tau\right)\left(\prod_{\tau=t}^{T}c_\tau\right)\alpha_t(i)\beta_t(i)}{\left(\prod_{\tau=1}^{t}c_\tau\right)\left(\prod_{\tau=t}^{T}c_\tau\right)\sum_{i=1}^{N}\alpha_t(i)\beta_t(i)} \\
&= \frac{\alpha_t(i)\beta_t(i)}{\sum_{i=1}^{N}\alpha_t(i)\beta_t(i)}
\end{aligned}
\tag{4.96}
$$

As (4.95) and (4.96) shows, $\xi_t(i,j)$ and $\gamma_t(i)$ are the same if scaled or not scaled

forward and backward variables are used. Herein lies the elegance with using a scale factor that is dependent of just time, and not dependent of the state. A closer look at (4.95) shows that $\xi_t(i,j)$ can be found easier when the scaled forward and backward variables are used:

$$\begin{aligned}
\xi_t(i,j) &= \frac{\hat{\alpha}_t(i)a_{ij}b_j(\mathbf{o}_{t+1})\hat{\beta}_{t+1}(j)}{\sum_{i=1}^{N}\sum_{j=1}^{N}\hat{\alpha}_t(i)a_{ij}b_j(\mathbf{o}_{t+1})\hat{\beta}_{t+1}(j)} \\
&= \frac{\hat{\alpha}_t(i)a_{ij}b_j(\mathbf{o}_{t+1})\hat{\beta}_{t+1}(j)}{\sum_{i=1}^{N}\hat{\alpha}_t(i)\left(\sum_{j=1}^{N}a_{ij}b_j(\mathbf{o}_{t+1})\hat{\beta}_{t+1}(j)\right)} \\
&= \frac{\hat{\alpha}_t(i)a_{ij}b_j(\mathbf{o}_{t+1})\hat{\beta}_{t+1}(j)}{\sum_{i=1}^{N}\hat{\alpha}_t(i)\hat{\beta}_t(j)} \\
&= \frac{\hat{\alpha}_t(i)a_{ij}b_j(\mathbf{o}_{t+1})\hat{\beta}_{t+1}(j)}{\frac{1}{\sum_{i=1}^{N}\alpha_t(i)\beta_t(j)}\sum_{i=1}^{N}\alpha_t(i)\beta_t(j)} \\
&= \hat{\alpha}_t(i)a_{ij}b_j(\mathbf{o}_{t+1})\hat{\beta}_{t+1}(j)
\end{aligned} \tag{4.97}$$

Since $\boldsymbol{\pi}$ and $\mathbf{A}$ uses $\xi_t(i,j)$ and $\gamma_t(i)$ for calculation, will these probabilities also be indepent of which forward or backward variables are used (scaled or unscaled). Thereby can (4.92) and (4.93) be found, without numerical problems, as:

$$\pi_i = \frac{\hat{\alpha}_1(i)\hat{\beta}_1(i)}{\sum_{i=1}^{N}\hat{\alpha}_T(i)} \tag{4.98}$$

And:

$$a_{ij} = \frac{\sum_{t=1}^{T-1}\hat{\alpha}_t(i)a_{ij}b_j(\mathbf{o}_{t+1})\hat{\beta}_{t+1}(j)}{\sum_{t=1}^{T-1}\hat{\alpha}_t(i)\hat{\beta}_t(i)} \tag{4.99}$$

The reestimation of c_{jk},$\boldsymbol{\mu}_{jk}$ and $\boldsymbol{\Sigma}_{jk}$ is a bit more complicated. However if the model has only one state j and one mixture, it would be an easy averaging task[3]:

[3]as transpose is $'$ used , this because no confusion with observation time T should be made

$$c_j = 1 \tag{4.100}$$

$$\boldsymbol{\mu}_j = \frac{1}{T}\sum_{t=1}^{T}\mathbf{o}_t \tag{4.101}$$

$$\boldsymbol{\Sigma}_j = \frac{1}{T}\sum_{t=1}^{T}\left(\mathbf{o}_t - \boldsymbol{\mu}_j\right)\left(\mathbf{o}_t - \boldsymbol{\mu}_j\right)' \tag{4.102}$$

In practice, of course, there are multiple states and multiple mixtures and there are no direct assignment of the observation vectors to individual states because the underlying state sequence is unknown. Since the full likelihood of each observation sequence is based on the summation of all possible state sequences, each observation vector $\mathbf{o}_t$ contributes to the computation of the likelihood for each state j. In other words instead of assigning each observation vector to a specific state, each observations i assigned to every state and is weighted with the probability of the model being in that state accounted for that specific mixture when the vector was observed. This probability, for state j and mixture k (there are M mixtures), is found by[4]:

$$\gamma_t(j,k) = \frac{\alpha_t(j)\beta_t(j)}{\sum_{i=1}^{N}\alpha_t(j)\beta_t(j)} \cdot \frac{c_{jk}\mathcal{N}\left(\mathbf{o}_t, \boldsymbol{\mu}_{jk}, \boldsymbol{\Sigma}_{jk}\right)}{\sum_{k=1}^{M} c_{jk}\mathcal{N}\left(\mathbf{o}_t, \boldsymbol{\mu}_{jk}, \boldsymbol{\Sigma}_{jk}\right)} \tag{4.103}$$

The reestimation formula for c_{jk} is the ratio between the expected number of times the system is in state j using the kth mixture component, and the expected number of times the system is in state j. That is:

$$c_{jk} = \frac{\sum_{t=1}^{T}\gamma_t(j,k)}{\sum_{t=1}^{T}\sum_{k=1}^{M}\gamma_t(j,k)} \tag{4.104}$$

To find $\boldsymbol{\mu}_{jk}$ and $\boldsymbol{\Sigma}_{jk}$ one can weight the simple averages in (4.101) and (4.102) by the probability of being in state j and using mixture k when observing $\mathbf{o}_t$:

$$\boldsymbol{\mu}_{jk} = \frac{\sum_{t=1}^{T}\gamma_t(j,k)\mathbf{o}_t}{\sum_{t=1}^{T}\gamma_t(j,k)} \tag{4.105}$$

[4] $\gamma_t(j,k)$ is a generalization of $\gamma_t(j)$

$$\boldsymbol{\Sigma}_{jk} = \frac{\sum_{t=1}^{T} \gamma_t(j,k) \left(\mathbf{o}_t - \boldsymbol{\mu}_j\right) \left(\mathbf{o}_t - \boldsymbol{\mu}_j\right)'}{\sum_{t=1}^{T} \gamma_t(j,k)} \tag{4.106}$$

The reestimation formulas described in this section, are performed based on one training sample. This will of course not be sufficient to get reliable estimates for a training sample, especially when left-right models are used. To get reliable estimates it is convenient to use multiple observation sequences.

4.6.1 Multiple Observation Sequences

In the Baum-Welch method, many parameters are to be estimated. The more parameters that has to be estimated, the more training examples are needed to get robust and reliable estimates. If only one observation sequence is used to train the model then would the model perform good recognition on this particular sample, but might give low recognition rate when testing other utterances of the same word. That is the model will be overtrained to this specific word, this can also give numerical problems to the reestimation procedure.

The modification of the reestimation procedure is straightforward and is as follows. Lets denote a set of R observation sequences as:

$$\mathbf{O} = \left[\mathbf{O}^{(1)}, \mathbf{O}^{(2)} \ldots, \mathbf{O}^{(R)}\right] \tag{4.107}$$

Where $\mathbf{O}^{(r)} = \left(\mathbf{o}_1^{(r)}, \mathbf{o}_2^{(r)}, \ldots, \mathbf{o}_{T_r}^{(r)}\right)$ is the rth observation sequence and T_r is the rth observation sequence length. The new function $P\left(\mathbf{O}|\lambda\right)$ to maximize ,according to (4.85), will now be:

$$P\left(\mathbf{O}|\lambda\right) = \prod_{r=1}^{R} P\left(\mathbf{O}^{(r)}|\lambda\right) \tag{4.108}$$

The method for achieving this maximation problem is similar to the procedure with only one observation of sequences. The variables $\alpha_t(j)$, $\beta_t(j)$, $\gamma_t(j,k)$ and $\xi_t(i,j)$ are found for every training sample. Then the reestimation is performed on the accumulated values. In this way are the new parameters a kind of averaging of the optima given by the individual training sequences. This procedure is then repeated until a optima is found, or some maximum limit of iterations is reached (typically 40). The reestimation formulas will now be:

$$\pi_i = \frac{\sum_{r=1}^{R} \hat{\alpha}_1^{(r)}(i)\hat{\beta}_1^{(r)}(i)}{\sum_{r=1}^{R}\sum_{i=1}^{N} \hat{\alpha}_{T_r}^{(r)}(i)} \tag{4.109}$$

$$a_{ij} = \frac{\sum_{r=1}^{R}\sum_{t=1}^{T_r-1} \hat{\alpha}_t^{(r)}(i)a_{ij}b_j(\mathbf{o}_{t+1}^{(r)})\hat{\beta}_{t+1}^{(r)}(j)}{\sum_{r=1}^{R}\sum_{t=1}^{T_r-1} \hat{\alpha}_t^{(r)}(i)\hat{\beta}_t^{(r)}(i)} \tag{4.110}$$

$$c_{jk} = \frac{\sum_{r=1}^{R}\sum_{t=1}^{T_r} \gamma_t^{(r)}(j,k)}{\sum_{r=1}^{R}\sum_{t=1}^{T_r}\sum_{k=1}^{M} \gamma_t^{(r)}(j,k)} \tag{4.111}$$

$$\boldsymbol{\mu}_{jk} = \frac{\sum_{r=1}^{R}\sum_{t=1}^{T_r} \gamma_t^{(r)}(j,k)\mathbf{o}_t^{(r)}}{\sum_{r=1}^{R}\sum_{t=1}^{T_r} \gamma_t^{(r)}(j,k)} \tag{4.112}$$

$$\boldsymbol{\Sigma}_{jk} = \frac{\sum_{r=1}^{R}\sum_{t=1}^{T_r} \gamma_t^{(r)}(j,k)\left(\mathbf{o}_t^{(r)} - \boldsymbol{\mu}_j\right)\left(\mathbf{o}_t^{(r)} - \boldsymbol{\mu}_j\right)'}{\sum_{r=1}^{R}\sum_{t=1}^{T_r} \gamma_t^{(r)}(j,k)} \tag{4.113}$$

These are the final formulas used for the reestimation. During training is the parameter $\widetilde{P}^*$, see (4.82), is found for every training sample in every iteration (using the alternative Viterbi algorithm). This probabilities are then added to get a total probability. When this total probability not increases anymore the optimum is reached.

4.6.2 Initial Estimates of HMM Parameters

Before the reestimation formulas can be applied for training, it is important to get good initial parameters so that the reestimation leads to the global maximum or as close as possible to it. An adequate choice for $\boldsymbol{\pi}$ and $\mathbf{A}$ is the uniform distribution. But since left-right models are used, $\boldsymbol{\pi}$ will have probability one for the first state and zero for the other states. For example will the left-right model in Fig. 4.3b have the following initial $\boldsymbol{\pi}$ and $\mathbf{A}$:

$$\boldsymbol{\pi} = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix}_{4\times 1} \tag{4.114}$$

$$\mathbf{A} = \begin{bmatrix} 0.5 & 0.5 & 0 & 0 \\ 0 & 0.5 & 0.5 & 0 \\ 0 & 0 & 0.5 & 0.5 \\ 0 & 0 & 0 & 1 \end{bmatrix}_{4\times 4} \tag{4.115}$$

The parameters for the emission distribution needs good initial estimations, to get a rapid and proper convergence. This is done by using uniform segmentation into the states of the model, for every training sample. After segmentation, all observations from the state j is collected from all training samples. Then a clustering algorithm is used to get the initial parameters for state j and this procedure is done for every state. The clustering algorithm used in this thesis is the well known *k-means* algorithm. Before the clustering proceeds one has to choose the number of clusters, K. In this task is the number of clusters equal to the number of mixtures, that is, $K = M$.

The K-Means Algorithm

1. **Initialization**

   Choose K vectors from the training vectors, here denoted x, at random. These vectors will be the *centroids* μ_k, which is to be found correctly.

2. **Recursion**

   For each vector in the training set, let every vector belong to a cluster k. This is done by choosing the cluster closest to the vector:

   $$k^* = \arg\min_k \left[d\left(x, \mu_k\right) \right] \tag{4.116}$$

   Where $d\left(x, \mu_k\right)$ is a distance measure, here is the Euclidian distance measure used:

   $$d\left(x, \mu_k\right) = \sqrt{\left(x - \mu_k\right)^T \left(x - \mu_k\right)} \tag{4.117}$$

3. **Test**

Recomputed the centroids, μ_k, by taking a mean of the vectors that belong to this centroid. This is done for every μ_k. If no vectors belongs to some μ_k for some value k - create new μ_k by choosing a random vector from x. If there has been no change of the centroids from the previous step goto step 4, otherwise go back to step 2.

4. **Termination**

From this clustering (done for one state j), the following parameters are found:

$$\begin{aligned} c_{jk} &= \text{number of vectors classified in cluster } k \text{ of state } j \\ &\quad \text{divided by the number of vectors in state } j \\ \boldsymbol{\mu}_{jk} &= \text{sample mean of vectors classified in cluster } k \text{ of state } j \\ \boldsymbol{\Sigma}_{jk} &= \text{sample covariance matrix of the vectors classified in} \\ &\quad \text{cluster } k \text{ of state } j \end{aligned}$$

4.6.3 Numerical Issues

One problem with training HMM parameters with the reestimation formulas, is that the number of training observation may not be sufficient. There is always an inadequate number of occurrences of low probability events to give food estimates of the model parameters. This has the effect that the mixture weights and the diagonal values in the covariance matrix can be zero, which is undesired. To avoid this a numerical floor ϵ is set:

$$c_{jk} = \begin{cases} c_{jk}, & \text{if } c_{jk} \geq \epsilon \\ \epsilon & \text{otherwise} \end{cases} \tag{4.118}$$

And the same for the diagonal values in the covariance matrix:

$$\boldsymbol{\Sigma}_{jk}(d,d) = \begin{cases} \boldsymbol{\Sigma}_{jk}(d,d), & \text{if } \boldsymbol{\Sigma}_{jk}(d,d) \geq \epsilon \\ \epsilon, & \text{otherwise} \end{cases}, \quad 1 \leq d \leq D \tag{4.119}$$

The numerical floor value is typical in the range of $10^{-10} \leq \epsilon \leq 10^{-3}$.

An other numerical problem occurs when $\gamma_t(j,k)$ is calculated by (4.103). The problem lies in finding:

$$\frac{c_{jk}\mathcal{N}\left(\mathbf{o}_t, \boldsymbol{\mu}_{jk}, \boldsymbol{\Sigma}_{jk}\right)}{\sum_{k=1}^{M} c_{jk}\mathcal{N}\left(\mathbf{o}_t, \boldsymbol{\mu}_{jk}, \boldsymbol{\Sigma}_{jk}\right)} \tag{4.120}$$

The problem is that $\mathcal{N}\left(\mathbf{o}_t, \boldsymbol{\mu}_{jk}, \boldsymbol{\Sigma}_{jk}\right)$ can actually become zero for every mixture, hence a division by zero occurs. This typically occurs when the dimension of the observations is large. The reason why $\mathcal{N}\left(\mathbf{o}_t, \boldsymbol{\mu}_{jk}, \boldsymbol{\Sigma}_{jk}\right)$ becomes zero, can be seen in the definition:

$$\mathcal{N}\left(\mathbf{o}_t, \boldsymbol{\mu}_{jk}, \boldsymbol{\Sigma}_{jk}\right) = \frac{1}{(2\pi)^{D/2}|\boldsymbol{\Sigma}_{jk}|^{1/2}} e^{-\frac{1}{2}\left(\mathbf{o}_t - \boldsymbol{\mu}_{jk}\right)^T \boldsymbol{\Sigma}_{jk}^{-1}\left(\mathbf{o}_t - \boldsymbol{\mu}_{jk}\right)} \tag{4.121}$$

Looking on the exponent it is easy to see that if $-\frac{1}{2}\left(\mathbf{o}_t - \boldsymbol{\mu}_{jk}\right)^T \boldsymbol{\Sigma}_{jk}^{-1}\left(\mathbf{o}_t - \boldsymbol{\mu}_{jk}\right)$ is a large negative number, it will give that the whole expression becomes zero. The solution to this problem will be that a scale factor e^b will be multiplied with (4.120) in both numerator and denominator. That is:

$$\frac{c_{jk}\mathcal{N}\left(\mathbf{o}_t, \boldsymbol{\mu}_{jk}, \boldsymbol{\Sigma}_{jk}\right) \cdot e^b}{\sum_{k=1}^{M} c_{jk}\mathcal{N}\left(\mathbf{o}_t, \boldsymbol{\mu}_{jk}, \boldsymbol{\Sigma}_{jk}\right) \cdot e^b} \tag{4.122}$$

The effect of multiplying with this exponential factor can be seen in the definition of the normal distribution:

$$\begin{aligned} \mathcal{N}\left(\mathbf{o}_t, \boldsymbol{\mu}_{jk}, \boldsymbol{\Sigma}_{jk}\right) \cdot e^b &= \frac{1}{(2\pi)^{D/2}|\boldsymbol{\Sigma}_{jk}|^{1/2}} e^{-\frac{1}{2}\left(\mathbf{o}_t - \boldsymbol{\mu}_{jk}\right)^T \boldsymbol{\Sigma}_{jk}^{-1}\left(\mathbf{o}_t - \boldsymbol{\mu}_{jk}\right)} \cdot e^b \\ &= \frac{1}{(2\pi)^{D/2}|\boldsymbol{\Sigma}_{jk}|^{1/2}} e^{-\frac{1}{2}\left(\mathbf{o}_t - \boldsymbol{\mu}_{jk}\right)^T \boldsymbol{\Sigma}_{jk}^{-1}\left(\mathbf{o}_t - \boldsymbol{\mu}_{jk}\right) + b} \end{aligned} \tag{4.123}$$

Hence the large negative exponential term can be adjusted to be a smaller value. If b is chosen as the minimum absolute exponential value from all mixtures, as:

$$b = \min_k \left| -\frac{1}{2}\left(\mathbf{o}_t - \boldsymbol{\mu}_{jk}\right)^T \boldsymbol{\Sigma}_{jk}^{-1}\left(\mathbf{o}_t - \boldsymbol{\mu}_{jk}\right) \right| \tag{4.124}$$

This gives that the denominator in (4.120) is always nonzero. Hence the division by zero will not occur, and (4.120) will still get a correct value.

Chapter 5

Speech Quality Assessment

This chapter will discuss different methods to measure the quality of speech. One type of quality assignment is the classical signal-to-noise ratio (SNR). The classical SNR measure weights all time domain errors in the speech waveform, but because of that the speech energy generally is time varying, this measure is a poor estimator of speech quality for a broad range of speech distorsions [2]. To enhance the quality measurement there exist other methods of which the following are to be discussed in this chapter: *The segmental SNR* ($\text{SNR}_{\text{seg}-\text{mean}}$,$\text{SNR}_{\text{seg}-\text{median}}$) and *the Itakura Measure*. To get an introduction to the basic knowledge about quality measure, the classical SNR will now be discussed.

5.1 The Classical SNR

The classical SNR ($\text{SNR}_{\text{classical}}$) quality measure, discussed in this section, has one benefit in its mathematical simplicity. The classical SNR represents an average error over time and frequency for a processed signal.

Before the calculation of the classical SNR can be performed, some dependent values need to be calculated.

Let $s(n)$ denote a noise-free signal at time n and $\hat{s}(n)$ the corresponding distorted(noisy) signal. These signals are assumed to be energy signals.

In a general case the energy of a *discrete* time signal $x(n)$ is defined as:

$$E_x = \sum_{n=-\infty}^{\infty} |x(n)|^2 \tag{5.1}$$

The signal $x(n)$ is called an *energy signal* if:

$$0 < E_x < \infty \tag{5.2}$$

With this definition in mind, the time error signal can be written as:

$$\varepsilon(n) = s(n) - \hat{s}(n) \tag{5.3}$$

The error energy is then

$$E_\varepsilon = \sum_{n=-\infty}^{\infty} \varepsilon^2(n) \tag{5.4}$$

The energy in a speech signal $s(n)$ is:

$$E_s = \sum_{n=-\infty}^{\infty} s^2(n) \tag{5.5}$$

Combining (5.4) and (5.5) the classical SNR is formulated as:

$$\mathrm{SNR}_{\mathrm{classical}} = 10 \log_{10} \frac{E_s}{E_\varepsilon} = 10 \log_{10} \frac{\sum_n s^2(n)}{\sum_n \left[s(n) - \hat{s}(n)\right]^2} \tag{5.6}$$

As mentioned earlier this SNR measure is not well suited as a qualitative measure of speech quality. If it is assumed that the noise distortion is broadband then the SNR should vary on a frame-by-frame basis. A reason for this is that if the noisy speech signal $\hat{s}(n)$ contain parts where it is no speech, just noise, then the classical SNR is very poor because it measures over the whole signal. The result from this gives that the classical SNR does not show the correct SNR between signal and noise, if silent parts are present.

One approach to get around this problem is to identify the silent parts. To identify the silent parts in the original speech waveform detect the non-silent parts (speech parts) by using a voice activation detection[1]. Having detected the non-silent parts, the silent parts have automatically been identified. To improve the signal-noise-ratio, the silent parts are excluded, calculating the $\mathrm{SNR}_{\mathrm{classical}}$ only for the non-silent parts of the speech waveform.

If voice activation detection is not possible to perform other quality measures can be useful. These methods take both silent and non-silent parts into consideration, calculating the SNR measure over short frames (about 20-30 ms) and the results are averaged. The method performing this task will be discussed in the following section.

[1]Please recall Ch. 3 Sec. 3.1.2

5.2 The Segmental SNR

The segmental SNR ($\text{SNR}_{\text{seg-mean}}$) is formulated as:

$$\text{SNR}_{\text{seg-mean}} = \frac{1}{M}\sum_{j=0}^{M-1} 10\log_{10}\left[\frac{\sum_{n=m_j-N+1}^{m_j} s^2(n)}{\sum_{n=m_j-N+1}^{m_j}\left[s(n)-\hat{s}(n)\right]^2}\right] \tag{5.7}$$

where m_0,m_1,...,m_{M-1} are the end times for the M frames, each of which is length N. The segments are produced by windowing the speech, with window length typically chosen to be 15 to 20 ms[2][9].

Problems can, in some cases, arise with the $\text{SNR}_{\text{seg-mean}}$ measure if there are long intervals of silence in the utterance. In segments in which the original speech is nearly zero, any amount of noise will give rise to a large negative signal-to-noise ratio for that segment, which could appreciably bias the overall measure of the $\text{SNR}_{\text{seg-mean}}$. Further calculating the $\text{SNR}_{\text{seg-mean}}$ as a mean value of all SNR values for each frame, gives a poor estimate of the reality, because it take all values in consideration. A better approach is to calculate the segmental SNR on a *median* basis, which is not as sensitive, to a large deviation between SNR value for each frame, as the mean value approach. Rewriting 5.7 gives:

$$\text{SNR}_{\text{seg-median}} = \text{median}\left\{10\log_{10}\left[\frac{\sum_{n=m_j-N+1}^{m_j} s^2(n)}{\sum_{n=m_j-N+1}^{m_j}\left[s(n)-\hat{s}(n)\right]^2}\right]\right\} \tag{5.8}$$

where the mathematical function median(x), for a sequence $x(n)$ (that has to be sorted), gives two results depending on whether $x(n)$ is an odd or even sequence. For an odd sequence median(x) is equal to the middle value of $x(n)$ and for an even sequence median(x) is equal to a mean of the two middle values of $x(n)$.

The $\text{SNR}_{\text{seg-median}}$ can be calculated either with respect to the whole sequence or with respect to only to the non-silent parts of the speech waveform. Minor problems can arise when calculating the Segmental SNR on a median basis, if the speech waveform consists of mostly silent parts that can be distorted by for example additive white gaussian noise. Calculating the $\text{SNR}_{\text{seg-median}}$ for these parts give rise to a large negative SNR value in those frames consisting of nearly just noise.

[2]Block length $K = 320$ and $F_s = 16$ kHz gives 20 ms frames

5.3 Comparison between SNR measures

To get a better understanding of the theory presented earlier it is useful to make some practical experiments on a real speech signal.

In the comparison a noise-free speech signal $s(n)$ are used and distorted by additive white gaussian noise $d(n)$ resulting in the signal $\hat{s}_k(n) = s(n) + d(n) \cdot \alpha \cdot k$ with time index n. Here α is a constant that, together with $1 \leq k \leq K$, linearly decade the amount of noise $d(n)$ added to the speech signal $s(n)$, see Fig. 5.1.

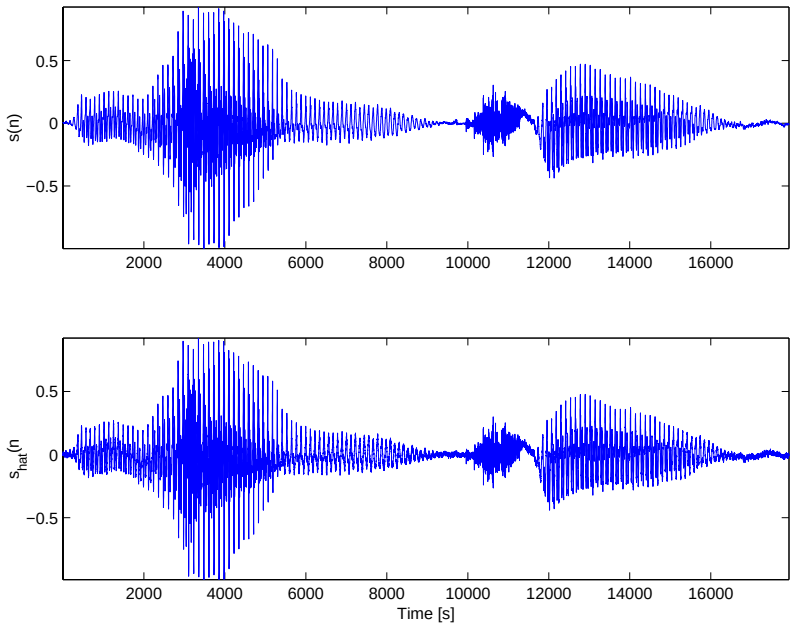

Figure 5.1: $s(n)$ and $\hat{s}(n)$

This comparison will try to show the differences between the SNR methods ($\text{SNR}_{\text{classical}}$,$\text{SNR}_{\text{seg-mean}}$ and $\text{SNR}_{\text{seg-median}}$)[3] and which one to choose in relevance to the tasks in this thesis. For each value of k a new SNR value are calculated for each one of the SNR methods, see Fig. 5.2.

Constant	Value
K	100
α	0.0015

Table 5.1: Choosen values in the SNR measures

Based on the SNR calculations, made in this chapter, the method finally chosen to be used further on is the classical SNR with silence removal. In this way the

[3]Please recall Secs. 5.1,5.2

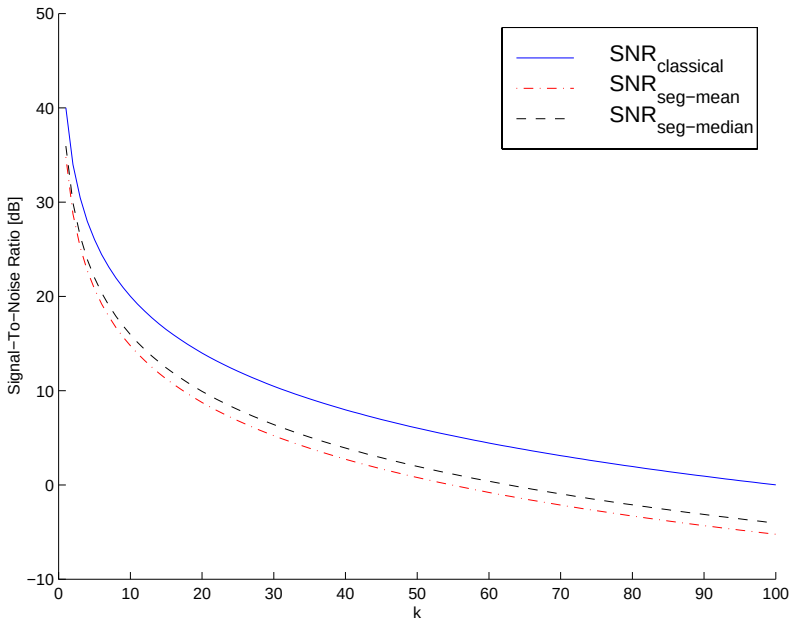

Figure 5.2: Signal-To-Noise Ratio [dB]

SNR will be more relevant to use. This because in a speech recognizer only the non-silent parts are of interest.

5.4 The Itakura Measure

Measures based on SNR, which obtain a distortion measure based on sample-by-sample differences in the original and processed time waveforms, do not provide a meaningful measure of performance when the two waveforms differ in their phase spectra. Distance measures that are sensitive to variations in the speech spectrum are therefore needed. One of the more successful is *the Itakura Measure* which measures the distance between two Linear Prediction (LP) vectors, $\hat{\mathbf{a}}(m)$ and $\hat{\mathbf{b}}(m')$. Since these parameters bear a close relationship to the short-term spectra of the speech frames from which they were drawn, *the Itakura Measure* seek to learn how similar the corresponding spectra are.

Since the LP parameters in the vectors, $\hat{\mathbf{a}}(m)$ and $\hat{\mathbf{b}}(m')$, are highly correlated an estimated correlation matrix is appropriate. The matrix, created based on the clean speech signal $s(n)$, is formulated as:

$$\tilde{\mathbf{R}}_s(m) \stackrel{\text{def}}{=} \begin{bmatrix} r_s(0;m) & \mathbf{r}_s^T(m) \\ \mathbf{r}_s(m) & \mathbf{R}_s(m) \end{bmatrix} \tag{5.9}$$

$\tilde{\mathbf{R}}_s(m)$ signifies the $(M+1)\times(M+1)$ *augmented* correlation matrix where M is

the order of the LP solution. $\mathbf{R}_s(m)$ is the (short-term) autocorrelation matrix obtained by Toeplitz operation on $\mathbf{r}_s(m)$. The LP vectors are found by using, for example, *the Levinson-Durbin Recursion (L-D Recursion)* [3].

The starting point for the L-D recursion can be formulated as:

$$\begin{array}{rcl} \tilde{\mathbf{R}}_s(m) & \times & [\ 1 \quad -\hat{a}(1;m) \quad -\hat{a}(2;m) \quad \cdots \quad -\hat{a}(M;m)\ ]^T \\ & = & [\ \xi(m) \quad 0 \quad 0 \quad \cdots \quad 0\ ]^T \end{array} \tag{5.10}$$

where $\xi(m)$ denotes *the Mean Squared Error (MSE)*[4] associated with the prediction of the frame $f(n;m)$ over all time using parameters $\hat{\mathbf{a}}(m)$ [2]. By defining $\boldsymbol{\alpha}(m)$ as:

$$\begin{array}{rcl} \boldsymbol{\alpha}(m) & \stackrel{\text{def}}{=} & [\ 1 \quad -\hat{a}(1;m) \quad -\hat{a}(2;m) \quad \cdots \quad -\hat{a}(M;m)\ ]^T \\ & = & [\ 1 \quad -\hat{\mathbf{a}}^T(m)\ ]^T \end{array} \tag{5.11}$$

the scalar $\xi_{\hat{\mathbf{a}}}(m)$ can be computed as:

$$\xi_{\hat{\mathbf{a}}}(m) = \boldsymbol{\alpha}^T(m)\tilde{\mathbf{R}}_s(m)\boldsymbol{\alpha}(m) \tag{5.12}$$

Using coefficients $\hat{b}(1;m'),\ldots,\hat{b}(M;m')$ [The elements of LP vector $\hat{\mathbf{b}}(m')$] to predict the frame $f(n;m)$. The vector $\hat{\mathbf{b}}(m')$ can be derived from any frame in any signal. There will be a certain MSE associated with this prediction, defined as:

$$\xi_{\hat{\mathbf{b}}}(m) = \boldsymbol{\beta}^T(m')\tilde{\mathbf{R}}_s(m)\boldsymbol{\beta}(m') \tag{5.13}$$

with

$$\boldsymbol{\beta}(m') \stackrel{\text{def}}{=} [\ 1 \quad -\hat{\mathbf{b}}^T(m)\ ]^T \tag{5.14}$$

Comparing these two scalars ($\xi_{\hat{\mathbf{a}}}(m)$ and $\xi_{\hat{\mathbf{b}}}(m)$) with respect to prediction error it is known that:

$$\xi_{\hat{\mathbf{a}}}(m) \leq \xi_{\hat{\mathbf{b}}}(m) \tag{5.15}$$

because $\xi_{\hat{\mathbf{a}}}(m)$ is the best possible prediction error in the sense of minimizing the average squared prediction error [2].

Finally a measure of how "far" LP vector $\hat{\mathbf{b}}(m')$ is from LP vector $\hat{\mathbf{a}}(m)$ is performed by calculating the *the Itakura Distance* which is defined as [2]:

$$d_1[\hat{\mathbf{a}}(m),\hat{\mathbf{b}}(m')] \stackrel{\text{def}}{=} \log\frac{\xi_{\hat{\mathbf{b}}}(m)}{\xi_{\hat{\mathbf{a}}}(m)} = \log\frac{\boldsymbol{\beta}^T(m')\tilde{\mathbf{R}}_s(m)\boldsymbol{\beta}(m')}{\boldsymbol{\alpha}^T(m')\tilde{\mathbf{R}}_s(m)\boldsymbol{\alpha}(m)} \tag{5.16}$$

[4]Please recall Eq. (5.4)

This measure will always be positive because of the condition in Eq. (5.15). This measure is not a metric because it does not have the required property of symmetry as:

$$d_1[\hat{\mathbf{a}}(m), \hat{\mathbf{b}}(m')] \neq d_1[\hat{\mathbf{b}}(m'), \hat{\mathbf{a}}(m)] \tag{5.17}$$

If a symmetric measure is desired, a combination can be formed such as:

$$\tilde{d_1}[\hat{\mathbf{a}}(m), \hat{\mathbf{b}}(m')] = \frac{1}{2}[d_1[\hat{\mathbf{a}}(m), \hat{\mathbf{b}}(m')] + d_1[\hat{\mathbf{b}}(m'), \hat{\mathbf{a}}(m)]] \tag{5.18}$$

In addition to symmetry, this measure also has the property that if the processed spectrum is identical to the original, the resulting distance is zero.

In the Ikatura measure, reusing $s(n)$ and $\hat{s}(n)$ from Sec. 5.3, a calculation of the distance between $s(n)$ and $\hat{s}(n)$ are performed resulting in the Itakura distance, see Fig. 5.3.

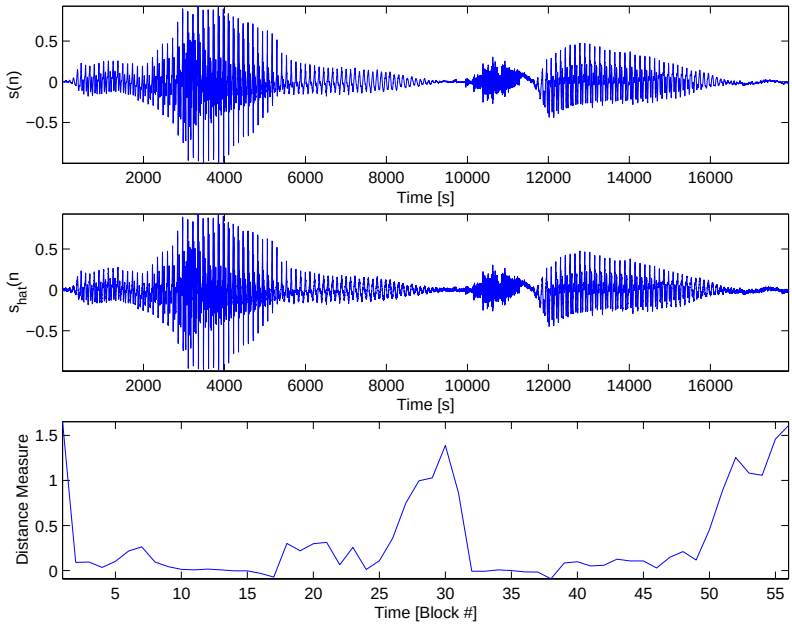

Figure 5.3: Itakura Measure, $s(n)$ different to $\hat{s}(n)$

From this test one can see that the Itakura Distance is low (near zero) if the difference between spectra of $s(n)$ and $\hat{s}(n)$ are small and in contradiction to this, give rise to a large distance value if the difference between $s(n)$ and $\hat{s}(n)$ are large. Hence the Itakura measure is used to get a better view of where noise is present.

Chapter 6

Practical Experimental Results

To test the theory presented earlier and get a practical experience of the speech recognition recognizer, described in this thesis, the recognizer was implemented in Matlab. The implementation was based on equations found in Ch. 3 and Ch. 4. Using this implementation the words zero to nine, spoken by one female and one male speaker, was used in the training phase. To test the recognizer in a real environment some noise recordings in a car has been done.

6.1 Measurements in car

In order to verify the performance of the speech recognizer in noisy environment, noise measurements were performed in two situations in a car. Noise were recorded partly when the car was not moving and partly when it was moving. Both recordings were made with the engine running.

The recordings were made placing a loudspeaker in the front passenger seat, simulating a speaking passenger present, see Fig. 6.1a, this to get the actual signal to noise ratio in the car. In the measurements two microphones were used in different positions. One microphone was placed in the front window, right in front of the simulated passenger, see Fig. 6.1b. The second microphone was positioned to right, in the middle over the passenger door, see Fig. 6.1c.

With these recordings the speech recognizer can be evaluated in different signal to noise ratio (SNR) environments with the noise from the car.

6.2 Performance in noisy environment

Before any evaluation can be done, the speech recognizer has to be trained. To get a reliable training, for the female and the male speaker, 20 samples of each word were recorded, in an environment free from external disturbances. The training

Figure 6.1: Overview of measurement equipment

results in ten models (one for each word) for each speaker. These models represents a statistical model (hidden Markov model, HMM) of each word. The HMM for each word has 15 states and 3 mixtures[1], the structure of the model is the left-right model with $\Delta = 2$, see Fig. 4.3c.

To evaluate the performance of the speech recognizer the trained models were used, recognizing the utterances zero to nine, in different SNR environments. In the evaluation, three different kinds of additive noise were used, *white gaussian noise* (N1), *noise in a car with engine running - car not moving* (N2) and *noise in a car with engine running - car moving* (N3). The spectrogram for these noises can be found in Fig. 6.2.

[1]Please recall Ch. 4 Sec. 4.2.3

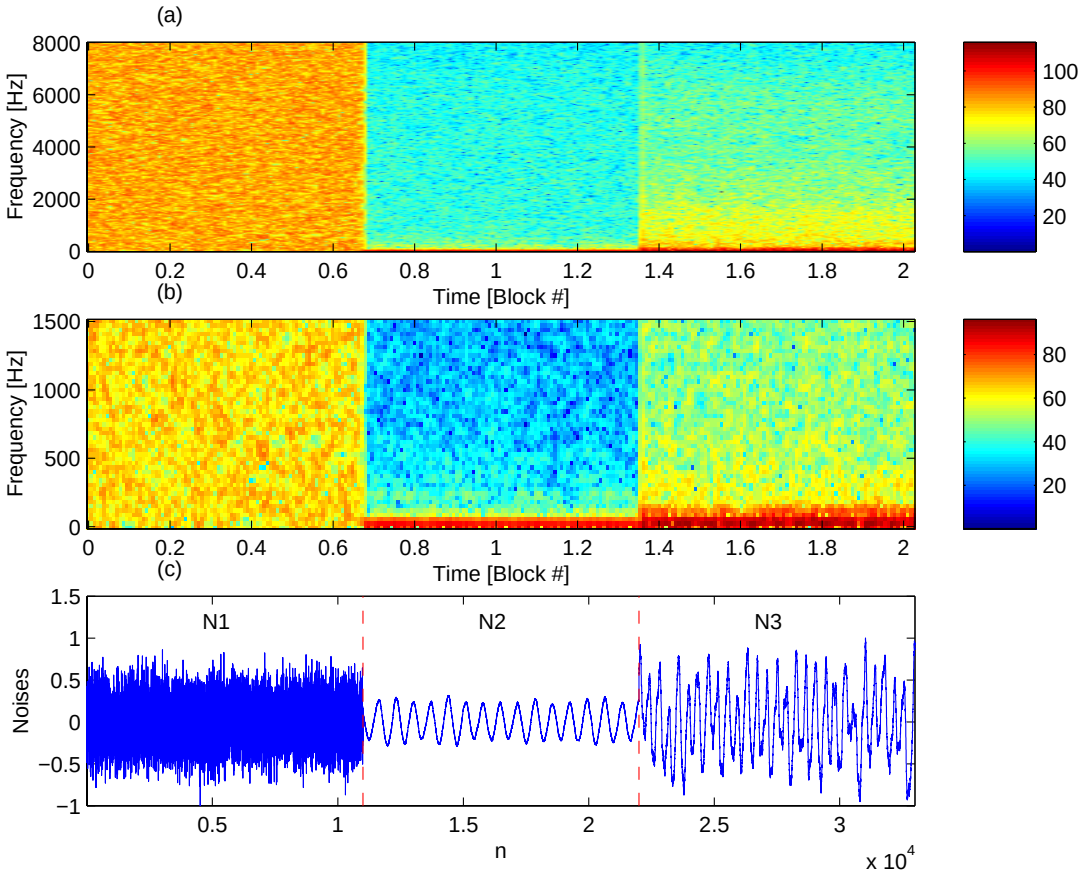

Figure 6.2: Properties of noises N1, N2 and N3

The spectrum for N1, shows that all frequencies will be influenced by this noise. N2 on the other hand has its influence mainly in the lower frequencies and this is also the case for N3. A closer look at the differences between N2 and N3, shows that N3 has more influence in the higher frequencies than N2, especially in the frequency range 100-1500 Hz.

The word recognition rate (WRR) in relationship to the SNR for the female speaker with N1, N2 and N3 can be found in Fig. 6.3, and the result for the male speaker can be found in Fig. 6.4. The actual SNR levels in the car environment are found in Tab. 6.1.

Mic position	N2	N3
Front	0.85295	-10.4454
Right	6.1878	-8.1028

Table 6.1: SNR [dB] levels in car

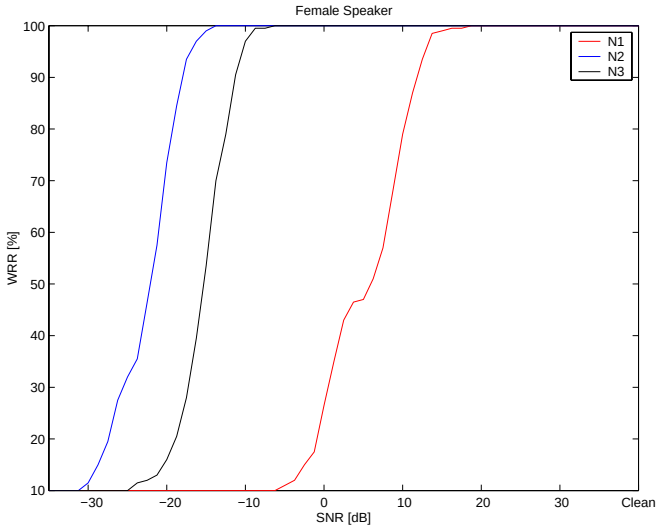

Figure 6.3: Recognition Rates - Female speaker

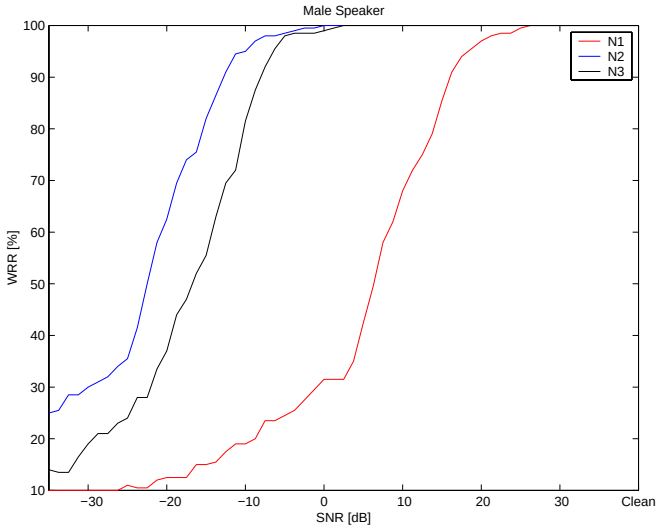

Figure 6.4: Recognition Rates - Male speaker

Studying Fig. 6.3 and Tab. 6.1 the actual recognition rates can be found in Tab. 6.2.

	Female		Male	
Mic position	N2	N3	N2	N3
Front	100	94.7	100	78.1
Right	100	99.5	100	89.8

Table 6.2: Word Recognition Rates [%] with actual SNR

This is fairly good results, considering that the noise in the car is rather high. Notice that placing the microphone to the right[2] will give better performance, probably because its placement is closer to the speaker. Adding some extra noise source (for example the fan in the car is turned on) the recognition rate will decrease. It would be desired to have a higher SNR to get further away from the critical area, approximately 0 dB and below. This might be done by applying some noise reduction algorithms.

[2]Please recall Fig. 6.1b

Chapter 7

Summary and Conclusions

The theory of hidden Markov model have been studied thoroughly. Together with the signal processing of speech signals, a speech recognizer has been implemented in Matlab. The speech recognizer simulation was found to perform good results (75-100%) in a car with the engine running. The comparison between, according to the recognition rate, white gaussian noise and the noise recorded in the car, shows that the recognizer is more robust to the noise in the car[1]. The reason to this is that the noise in the car mainly affects the low frequencies (0-2000 Hz), whilst the white gaussian noise affects all frequencies.

7.1 Further Work

Further improvements and expansions may be achieved by using one or more of the following suggestions:

- The speech recognizer is implemented in Matlab and because of that it runs slow. Implementing the speech recognizer in C or assembler will be desired to get a faster execution time.
- In a noisy environment, like in a car, noise reduction algorithms are preferred to enhance signal to noise ratio. Algorithms useful can be based on adaptive noise reduction, spectral subtraction or beam forming.
- Record a larger evaluation database, for different speakers and different environments, to get more test cases.
- Try different setting in the speech recognizer, for example change the model structure, the number of states or mixtures. More or less measures of the speech can be added to the feature vectors, that is experiment with the feature vector dimension and its content.

[1]Please recall Fig. 6.3

Appendix A

Phonemes

A.1 Continuant

The continuant sounds are *vowels, fricatives, affricates and nasals.*

A.1.1 Vowels

Vowels are phonated and are normally among the phonemes with highest amplitude. Vowels can vary widely in duration (typically 40-400 ms)[2] and are spectrally well defined. Vowels are produced by exciting an fixed vocal tract shape with quasi-periodic pulses of air caused by the vibration of the vocal cords. Variations in the cross-sectional area along the vocal tract determines the formants[1] of the vowels.

Vowels are differentiated by the position of the tongue-hump. The tongue-hump position divides the vowels into three groups: *front, middle, back.*

A.1.2 Consonants

All forms of excitation discussed in Sec. 2.2.1 may be involved by consonants. Factors that affect these types of sounds (consonants) are whether the tone of the vocal cords are present or not, precise dynamic movement of the vocal tract articulators and how the talker articulates. Consonants that are classified as continuant might not require a motion of the vocal tract.

Fricatives

Fricatives are produced by exciting the vocal tract with steady airstream that becomes turbulent at some point of constriction. Depending on the form of

[1] Please recall Section 2.2.3

excitation the fricatives are divided into two groups, unvoiced and voiced. Those with simple unvoiced excitation are usually called *unvoiced fricatives* while those of mixed excitation are called *voiced fricatives*.

Affricates

Affricates are formed by transitions from a stop[2] to a fricative. The unvoiced affricate is formed when an unvoiced stop, followed by a transition to the unvoiced fricative are produced. The voiced affricate is formed by producing a voiced stop followed by a vocal tract transition to the voiced fricative.

Nasals

Nasal consonants are voiced sounds produced by the waveform exciting an open nasal cavity and closed oral cavity. Their waveform resemble vowels, but are normally weaker in energy due to limited ability of the nasal cavity to radiate sound. When forming a nasal the front of the vocal tract are completely closed either with the lips or the tongue. The velum is opened wide to allow sound propagation through the nasal cavity. Nasal formants have a bandwidth that are normally wider than those for vowels. When phonemes precede or follow a nasal sound, it gives that those phonemes become *nasalized*. The nasalization produces phonemes with broader bandwidths and are less peaked than those without nasal coupling. This is caused by damping of the formant resonance by the loss of energy through the opening into the nasal cavity[2].

A.2 Non-Continuant

The non-continuant sounds are *diphthongs, liquids, glides and stops*.

A.2.1 Diphthongs

A diphthong involve a movement from one vowel toward another. When a sound is produced and one wish to determine if a specific part of this sound is a vowel or diphthong it depends on how the sound are produced. If the vocal tract does not maintain a constant shape, or if the sound cannot be sustained without articulatory movement, and both vocal targets are vowels, then the sound is a diphthong.

A.2.2 Semivowels

Semivowels are classified as either *liquids* or *glides*. Liquids have spectral characteristics similar to vowels, but are normally weaker than most vowels due to

[2]Please recall Sec. A.2.3

their more constricted vocal tract. A glide consists of one target position, with associated formant transitions toward and away from the target. Glides can be viewed as transient sounds as they maintain the target position for much less time than vowels.

A.2.3 Stops

Sounds in which the airstream enters the oral cavity and is stopped for a brief period are called *stops*. Stops are transient, non-continuant sounds that are produced by building up pressure behind a total constriction somewhere along the vocal tract, and suddenly release this pressure. This sudden explosion and aspiration of air characterizes the stop consonants.

Bibliography

[1] Rabiner Lawrence, Juang Bing-Hwang. *Fundamentals of Speech Recognition*, Prentice Hall , New Jersey, 1993, ISBN 0-13-015157-2

[2] Deller John R., Jr., Hansen John J.L., Proakis John G. , *Discrete-Time Processing of Speech Signals*, IEEE Press, ISBN 0-7803-5386-2

[3] Hayes H. Monson, *Statistical Digital Signal Processing and Modeling*, John Wiley & Sons Inc. , Toronto, 1996, ISBN 0-471-59431-8

[4] Proakis John G., Manolakis Dimitris G., *Digital Signal Processing, principles, algorithms, and applications, Third Edition*, Prentice Hall , New Jersey, 1996, ISBN 0-13-394338-9

[5] Jelinek Frederick, *Statistical Methods for Speech Recognition*, MIT Press, 1998, ISBN 0-262-10066-5

[6] Baum L.E., Eagon J. A., *An inequality with applications to statistical estimation for probabilistic functions of Markov processe and to model for ecology*, Bulletin of American Mathematical Society, 73:360-363, 1967

[7] Baum L.E., Petrie T., Soules G et al., *A maximization technique in the statistical analysis of probabilistic functions of Markov chains*, Annals of mathematical Statistics 41:164-171, 1970

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

[9] Quackenbush Schuler R., Barnwell III Thomas P., Clements Mark A., *Objective Measures of Speech Quality*, Prentice Hall, New Jersey, 1988, ISBN 0-13-629056-6

[10] Young Steve, *A Review of Large-vocabulary Continuous-speech Recognition*, IEEE SP Magazine, 13:45-57, 1996, ISSN 1053-5888

[11] Mammone Richard J., Zhang Xiaoyu, Ramachandran Ravi P., *Robust Speaker Recognition*, IEEE SP Magazine, 13:58-71, 1996, ISSN 1053-5888

[12] Davis K. H., Biddulph R. and Balashek S., *Automatic Recognition of Spoken Digits*, J. Acoust. Soc. Am., 24 (6):637-642, 1952

[13] Forgie J. W. and Forgie C. D., *Results Obtained From a Vowel Recognition Computer Program*, J. Acoust. Soc. Am., 31 (11):1480-1489, 1959

[14] Velichko V. M. and Zagoruyko N. G., *Automatic Recognition of 200 Words*, Int. J. Man-Machine Studies, 2:223, 1970

[15] Sakoe H. and Chiba S., *Dynamic Programming Algorithm Optimization for Spoken Word Recognition*, IEEE Trans. Acoustics, Speech, Signal Proc., ASSP-26 (1):43-49, 1978

[16] Itakura F., *Minimum Prediction Residual Applied to Speech Recogntition*, IEEE Trans. Acoustics, Speech, Signal Proc., ASSP-23 (1):67-72, 1975

[17] Rabiner L. R., Levinson S. E., Rosenberg A. E. and Wilpon J. G., *Speaker Independent Recognition of Isolated Words using Clustering Techniques*, IEEE Trans. Acoustics, Speech, Signal Proc., ASSP-27:336-349, 1979

Master Thesis MEE-01-27

Printed by Kaserntryckeriet AB, Karlskrona 2002