An Introduction to Speech Recognition

B. Plannerer

March 28, 2005

This document ...

... was written after I presented a tutorial on speech recognition on the graduates workshop "SMG", November 2000, Venice, held by Prof. Dr. H. G. Tillmann, Institut für Phonetik und Sprachliche Kommunikation, University of Munich, Germany. Initially, I intended to give a short overview over the principles of speech recognition and the industrial applications of speech recognition technology to an audience consisting both of engineers and non–engineers. As I worked on the slides, I tried to explain most of the principles and algorithms of speech recognition by using pictures and examples that can also be understood by non–technicians. For the engineers in the audience and to prevent me from making inappropriate simplifications, I also added the formal framework. Thus, I always started to explain the algorithms by using pictures and examples to give an idea of what the algorithm does and then went into some of the details of the formulae. Although I was afraid that this might be the perfect approach to boring both the engineers and the non–engineers of my audience, it turned out that they both could live with this mixed presentation. Being asked to convert my slides into this written introduction to speech recognition, I took the chance to add some more details and formal framework which can be skipped by the non–engineer readers.

Version 1.1

In version 1.1 of this document some error corrections were made. In some chapters examples and pictures were added. Of course, errors (old and new ones)will still be contained and additional refinements might be necessary. If you have any suggestions how to improve this document, feel free to send an email to
plannerer@ieee.org

Copyright

Who should read this?

This introduction to speech recognition is intended to provide students and graduates with the necessary background to understand the principles of speech recognition technology. It will give an overview over the very basics of pattern matching and stochastic modelling and it will show how the dynamic programming algorithm is used for recognition of isolated words as well as for the recognition of continuous speech. While it is impossible to present every detail of all these topics, it is intended to provide enough information to the interested reader to select among the numerous scientific publications on the subject. Those who are not interested in further studies should at least be provided with an idea of how automated speech recognition works and what today's systems can do — and can't do.

Chapter 1

The Speech Signal

1.1 Production of Speech

While you are producing speech sounds, the air flow from your lungs first passes the glottis and then your throat and mouth. Depending on which speech sound you articulate, the speech signal can be excited in three possible ways:

- **voiced excitation** The glottis is closed. The air pressure forces the glottis to open and close periodically thus generating a periodic pulse train (triangle–shaped). This *"fundamental frequency"* usually lies in the range from 80Hz to 350Hz.
- **unvoiced excitation** The glottis is open and the air passes a narrow passage in the throat or mouth. This results in a turbulence which generates a noise signal. The spectral shape of the noise is determined by the location of the narrowness.
- **transient excitation** A closure in the throat or mouth will raise the air pressure. By suddenly opening the closure the air pressure drops down immediately. ("plosive burst")

With some speech sounds these three kinds of excitation occur in combination. The spectral shape of the speech signal is determined by the shape of the vocal tract (the pipe formed by your throat, tongue, teeth and lips). By changing the shape of the pipe (and in addition opening and closing the air flow through your nose) you change the spectral shape of the speech signal, thus articulating different speech sounds.

1.2 Technical Characteristics of the Speech Signal

An engineer looking at (or listening to) a speech signal might characterize it as follows:

- The bandwidth of the signal is 4 kHz

- The signal is periodic with a fundamental frequency between 80 Hz and 350 Hz
- There are peaks in the spectral distribution of energy at

$$(2n-1) * 500\ Hz\ ;\ n = 1, 2, 3, \ldots \tag{1.1}$$

- The envelope of the power spectrum of the signal shows a decrease with increasing frequency (-6dB per octave)

This is a very rough and technical description of the speech signal. But where do those characteristics come from?

1.2.1 Bandwidth

The bandwidth of the speech signal is much higher than the 4 kHz stated above. In fact, for the fricatives, there is still a significant amount of energy in the spectrum for high and even ultrasonic frequencies. However, as we all know from using the (analog) phone, it seems that within a bandwidth of 4 kHz the speech signal contains all the information necessary to understand a human voice.

1.2.2 Fundamental Frequency

As described earlier, using voiced excitation for the speech sound will result in a pulse train, the so-called fundamental frequency. Voiced excitation is used when articulating vowels and some of the consonants. For fricatives (e.g., /f/ as in **f**ish or /s/, as in me**ss**), unvoiced excitation (noise) is used. In these cases, usually no fundamental frequency can be detected. On the other hand, the zero crossing rate of the signal is very high. Plosives (like /p/ as in **p**ut), which use transient excitation, you can best detect in the speech signal by looking for the short silence necessary to build up the air pressure before the plosive bursts out.

1.2.3 Peaks in the Spectrum

After passing the glottis, the vocal tract gives a characteristic spectral shape to the speech signal. If one simplifies the vocal tract to a straight pipe (the length is about 17cm), one can see that the pipe shows resonance at the frequencies as given by (1.1). These frequencies are called formant frequencies. Depending on the shape of the vocal tract (the diameter of the pipe changes along the pipe), the frequency of the formants (especially of the 1st and 2nd formant) change and therefore characterize the vowel being articulated.

1.2.4 The Envelope of the Power Spectrum Decreases with Increasing Frequency

The pulse sequence from the glottis has a power spectrum decreasing towards higher frequencies by -12dB per octave. The emission characteristics of the lips show a high-pass characteristic with +6dB per octave. Thus, this results in an overall decrease of -6dB per octave.

1.3 A Very Simple Model of Speech Production

As we have seen, the production of speech can be separated into two parts: Producing the excitation signal and forming the spectral shape. Thus, we can draw a simplified model of speech production [ST95, Rus88]:

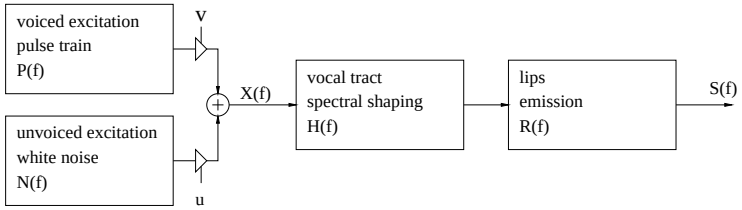

Figure 1.1: A simple model of speech production

This model works as follows: Voiced excitation is modelled by a pulse generator which generates a pulse train (of triangle–shaped pulses) with its spectrum given by $P(f)$. The unvoiced excitation is modelled by a white noise generator with spectrum $N(f)$. To mix voiced and unvoiced excitation, one can adjust the signal amplitude of the impulse generator (v) and the noise generator (u). The output of both generators is then added and fed into the box modelling the vocal tract and performing the spectral shaping with the transmission function $H(f)$. The emission characteristics of the lips is modelled by $R(f)$. Hence, the spectrum $S(f)$ of the speech signal is given as:

$$S(f) = (v \cdot P(f) + u \cdot N(f)) \cdot H(f) \cdot R(f) = X(f) \cdot H(f) \cdot R(f) \qquad (1.2)$$

To influence the speech sound, we have the following parameters in our speech production model:

- the mixture between voiced and unvoiced excitation (determined by v and u)
- the fundamental frequency (determined by $P(f)$)
- the spectral shaping (determined by $H(f)$)
- the signal amplitude (depending on v and u)

These are the technical parameters describing a speech signal. To perform speech recognition, the parameters given above have to be computed from the time signal (this is called speech signal analysis or *"acoustic preprocessing"*) and then forwarded to the speech recognizer. For the speech recognizer, the most valuable information is contained in the way the spectral shape of the speech signal changes in time. To reflect these dynamic changes, the spectral shape is determined in short intervals of time, e.g., every 10 ms. By directly computing the spectrum of the speech signal, the fundamental frequency would be implicitly contained in the measured spectrum (resulting in unwanted "ripples" in the spectrum). Figure 1.2 shows the time signal of the vowel /a:/ and fig. 1.3 shows the logarithmic power spectrum of the vowel computed via FFT.

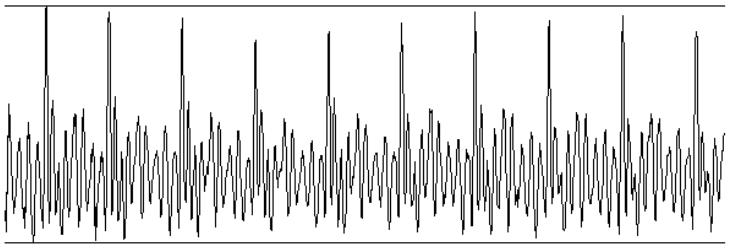

Figure 1.2: Time signal of the vowel /a:/ (f_s = 11kHz, length = 100ms). The high peaks in the time signal are caused by the pulse train $P(f)$ generated by voiced excitation.

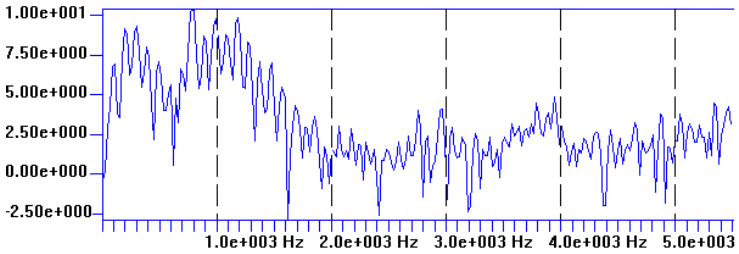

Figure 1.3: Log power spectrum of the vowel /a:/ (f_s = 11kHz, $N = 512$). The ripples in the spectrum are caused by $P(f)$.

One could also measure the spectral shape by means of an analog filter bank using several bandpass filters as is depicted below. After rectifying and smoothing the filter outputs, the output voltage would represent the energy contained in the frequency band defined by the corresponding bandpass filter (cf. [Rus88]).

1.4 Speech Parameters Used by Speech Recognition Systems

As shown above, the direct computation of the power spectrum from the speech signal results in a spectrum containing "ripples" caused by the excitation spectrum $X(f)$. Depending on the implementation of the acoustic preprocessing however, special transformations are used to separate the excitation spectrum $X(f)$ from the spectral shaping of the vocal tract $H(f)$. Thus, a smooth spectral shape (without the ripples), which represents $H(f)$ can be estimated from the speech signal. Most speech recognition systems use the so–called *mel frequency cepstral coefficients (MFCC)* and its first (and sometimes second) derivative in time to better reflect dynamic changes.

1.4.1 Computation of the Short Term Spectra

As we recall, it is necessary to compute the speech parameters in short time intervals to reflect the dynamic change of the speech signal. Typically, the spec-

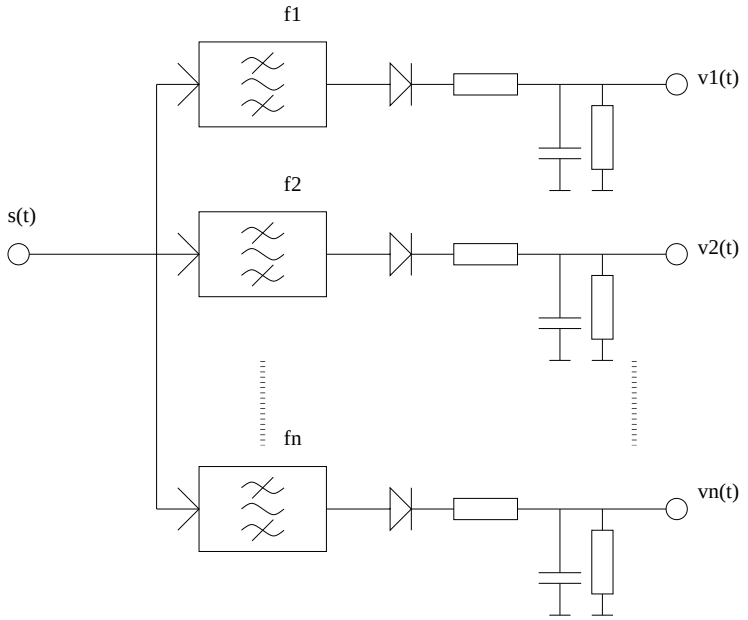

Figure 1.4: A filter bank for spectral analysis

tral parameters of speech are estimated in time intervals of 10ms. First, we have to sample and digitize the speech signal. Depending on the implementation, a sampling frequency f_s between 8kHz and 16kHz and usually a 16bit quantization of the signal amplitude is used. After digitizing the analog speech signal, we get a series of speech samples $s(k \cdot \Delta t)$ where $\Delta t = 1/f_s$ or, for easier notation, simply $s(k)$. Now a preemphasis filter is used to eliminate the -6dB per octave decay of the spectral energy:

$$\hat{s}(k) = s(k) - 0.97 \cdot s(k-1) \tag{1.3}$$

Then, a short piece of signal is cut out of the whole speech signal. This is done by multiplying the speech samples $\hat{s}(k)$ with a windowing function $w(k)$ to cut out a short segment of the speech signal, $v_m(k)$ starting with sample number $k = m$ and ending with sample number $k = m + N - 1$. The length N of the segment (its duration) is usually chosen to lie between 16ms to 25 ms, while the time window is shifted in time intervals of about 10ms to compute the next set of speech parameters. Thus, overlapping segments are used for speech analysis. Many window functions can be used, the most common one is the so–called Hamming-Window:

$$w(k) = \begin{cases} 0.54 - 0.46\cos\left(\frac{2\pi k}{N-1}\right) & : \text{ if } k = 0, 1, \ldots N-1 \\ 0 & : \text{ else} \end{cases} \tag{1.4}$$

where N is the length of the time window in samples. By multiplying our speech signal with the time window, we get a short speech segment $v_m(k)$:

$$v_m(k) = \begin{cases} \hat{s}(k) \cdot w(k-m) & : \text{ if } k = m, m+1, \ldots m+N-1 \\ 0 & : \text{ else} \end{cases} \tag{1.5}$$

As already mentioned, N denotes the length of the speech segment given in samples (the window length is typically between 16ms and 25ms) while m is the start time of the segment. The start time m is incremented in intervals of (usually) 10ms, so that the speech segments are overlapping each other. All the following operations refer to this speech segment $v_m(k)$, $k = m \ldots m + N - 1$. To simplify the notation, we shift the signal in time by m samples to the left, so that our time index runs from $0 \ldots N - 1$ again. From the windowed signal, we want to compute its discrete power spectrum $|V(n)|^2$. First of all, the complex spectrum $V(n)$ is computed. The complex spectrum $V(n)$ has the following properties:

- The spectrum $V(n)$ is defined within the range from $n = -\infty$ to $n = +\infty$.
- $V(n)$ is periodic with period N, i.e.,

$$V(n \pm i \cdot N) = V(n) \; ; \; i = 1, 2, \ldots \tag{1.6}$$

- Since $v_m(k)$ is real–valued, the absolute values of the coefficients are also symmetric:

$$|V(-n)| = |V(n)| \tag{1.7}$$

To compute the spectrum, we compute the discrete Fourier transform (DFT, which gives us the discrete, complex–valued short term spectrum $\hat{V}(n)$ of the speech signal (for a good introduction to the DFT and FFT, see [Bri87], and for both FFT and Hartley Transform theory and its applications see [Bra00]):

$$\hat{V}(n) = \sum_{k=0}^{N-1} v(k) \cdot e^{-j2\pi kn/N} \; ; \; n = 0, 1, \ldots N - 1 \tag{1.8}$$

The DFT gives us N discrete complex values for the spectrum $\hat{V}(n)$ at the frequencies $n \cdot \Delta f$ where

$$\Delta f = \frac{1}{N \Delta t} \tag{1.9}$$

Remember that the complex spectrum $V(n)$ is defined for $n = -\infty$ to $n = +\infty$, but is periodic with period N (1.6). Thus, the N different values of $\hat{V}(n)$ are sufficient to represent $V(n)$. One should keep in mind that we have to interpret the values of $\hat{V}(n)$ ranging from $n = N/2$ to $n = N - 1$ as the values for the negative frequencies of the spectrum $V(n \cdot \Delta f)$, i.e for $-N/2 \cdot \Delta f$ to $-1 \cdot \Delta f$. One could think that with a frequency range from $-N/2 \cdot \Delta f$ to $+N/2 \cdot \Delta f$ we should have $N + 1$ different values for $V(n)$, but since $V(n)$ is periodic with period N (1.6), we know that

$$V(-N/2 \cdot \Delta f) = V(N/2 \cdot \Delta f)$$

. So nothing is wrong, and the N different values we get from $\hat{V}(n)$ are sufficient to describe the spectrum $V(n)$. For further processing, we are only interested in the power spectrum of the signal. So we can compute the squares of the absolute values, $|V(n)|^2$.
Due to the periodicity (1.6) and symmetry (1.7) of $V(n)$, only the values $|V(0)|^2 \ldots |V(N/2)|^2$ are used for further processing, giving a total number of $N/2 + 1$ values.
It should be noted that $|V(0)|$ contains only the DC-offset of the signal and therefore provides no useful information for our speech recognition task.

1.4.2 Mel Spectral Coefficients

As was shown in perception experiments, the human ear does not show a linear frequency resolution but builds several groups of frequencies and integrates the spectral energies within a given group. Furthermore, the mid-frequency and bandwidth of these groups are non–linearly distributed. The non–linear warping of the frequency axis can be modeled by the so–called *mel-scale*. The frequency groups are assumed to be linearly distributed along the mel-scale. The so–called mel–frequency f_{mel} can be computed from the frequency f as follows:

$$f_{mel}\,(f) = 2595 \cdot \log\left(1 + \frac{f}{700Hz}\right) \tag{1.10}$$

Figure 1.5 shows a plot of the mel scale (1.10).

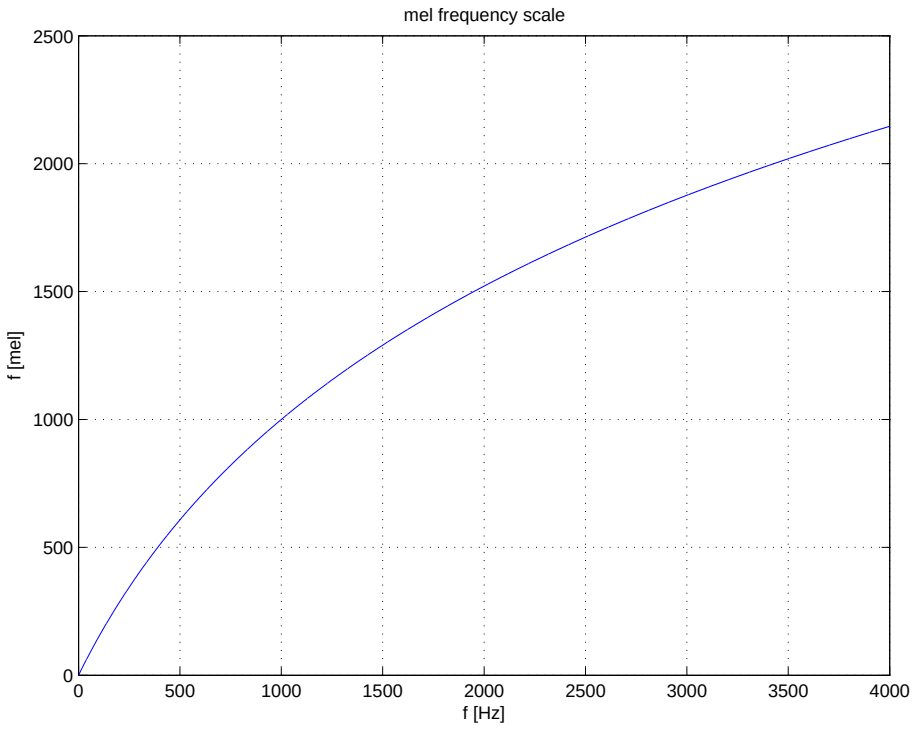

Figure 1.5: The mel frequency scale

The human ear has high frequency resolution in low–frequency parts of the spectrum and low frequency resolution in the high–frequency parts of the spectrum. The coefficients of the power spectrum $|V(n)|^2$ are now transformed to reflect the frequency resolution of the human ear.
A common way to do this is to use $\mathcal{K}$ triangle–shaped windows in the spectral domain to build a weighted sum over those power spectrum coefficients $|V(n)|^2$ which lie within the window. We denote the windowing coefficients as

$$\eta_{kn}\ ;\ k = 0, 1, \ldots \mathcal{K} - 1\ ;\ n = 0, 1, \ldots N/2$$

This gives us a new set of coefficients,

$$G(k)\ ;\ k = 0, 1, \ldots, \mathcal{K} - 1$$

the so–called *mel spectral coefficients*, e.g., [ST95].

$$G(k) = \sum_{n=0}^{N/2} \eta_{kn} \cdot |V(n)|^2\ ;\ k = 0, 1, \ldots \mathcal{K} - 1 \tag{1.11}$$

Caused by the symmetry of the original spectrum (1.7), the mel power spectrum is also symmetric in k:

$$G(k) = G(-k) \tag{1.12}$$

Therefore, it is sufficient to consider only the positive range of k, $k = 0, 1, \ldots \mathcal{K} - 1$.

1.4.3 Example: Mel Spectral Coefficients

This example will show how to compute the window coefficients η_{kn} for a given set of parameters. We start with the definition of the preprocessing parameters:

$$\begin{array}{rcc} f_s & = & 8000 \text{ Hz} \\ N & = & 256 \\ \mathcal{K} & = & 22 \end{array}$$

From these values, we can derive some useful parameters. First, we compute the maximum frequency of the sampled signal: $f_g = \frac{f_s}{2} = 4000$ Hz.
According to (1.10), this corresponds to a maximum mel frequency of f_{max} [mel] = 2146.1 mel. The center frequencies of our triangle-shaped windows are equally spaced on the mel scale. Therefore, the frequency distance bewteen every two center frequencies can easily be computed:

$$\Delta_{\text{mel}} = \frac{f_{max}}{\mathcal{K} + 1} = 93.3 \text{ mel}$$

Thus, the center frequencies on the mel scale can be expressed as:

$$f_c(k) = (k + 1) \cdot \Delta_{\text{mel}}\ ;\ k = 0 \cdots \mathcal{K} - 1$$

Given the center frequencies in the mel scale, the center frequencies on the linear frequency scale can easily be computed by solving equantion (1.10).
In figure 1.6 you can see a plot of the resulting center frequencies $f_c(k)$ on the linear frequency scale vs. the mel frequency scale.
Now the values for the center frequencies on the linear frequency scale have to be mapped to the indices of the FFT. The frequency resolution of our FFT with length N is given as $\Delta f = \frac{f_s}{N} = 31.25$Hz.
Dividing $f_c(k)$ on the linear frequency scale by Δf and rounding to the next integer, we get the corresponding indices of the coefficients of the power spectrum $|V(n)|^2$. The rounding to the next integer can be interpreted as a quantization of the linear frequency scale where the frequency resolution is given by Δf. We denote the power spectrum indices corresponding to the center frequencies as $n_c(k)$.

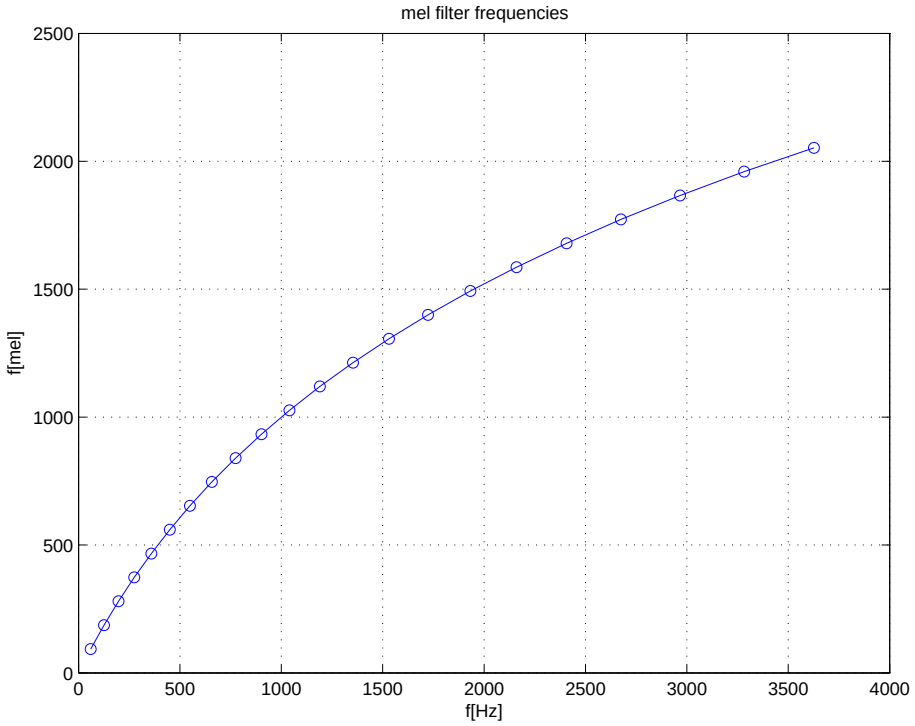

Figure 1.6: Center frequencies of the mel filter. ($f_s = 8\text{kHz}$; $\mathcal{K} = 22$).

In figure 1.7 you can see a plot of the power spectrum coefficient indices $n_c(k)$ versus the mel spectral coefficient index k ; $k = 0 \cdots \mathcal{K} - 1$.
Now we know the center frequencies $f_c(k)$ of our windows and the coresponding coefficients of the power spectrum $n_c(k)$.
All what is left to do is to compute the windowing coefficients η_{kn} ; $k = 0, 1, \ldots \mathcal{K} - 1$; $n = 0, 1, \ldots N/2$. The windowing coefficients η_{kn} can be represented as a windowing matrix $\tilde{W} = [\eta_{kn}]$. The rows of this matrix represent the individual windows k ("'filter channels"'), the columns of the matrix represent the weighting coefficients for each power spectrum coefficient n.
For each window (rows k in the matrix), the weighting coefficients η_{kn} are computed to form a triangle-shaped window with a maximum value of 1.0 at the center frequency $f_c(k)$. The column index of the window's maximum is given by the corresponding index of the power spectrum $n_c(k)$. The left and right boundaries of the triangles of a given window are defined to be the center frequencies of the left and right filter channel, i.e., $f_c(k-1)$ and $f_c(k+1)$. Thus, the corresponding weighting coefficients $\eta_{k,n_c(k-1)}$ and $\eta_{k,n_c(k+1)}$ are defined to be zero. The coefficients within the window are chosen to linearly raise from zero at column index $n_c(k-1)$ to the maximum value 1.0 at $n_c(k)$ and then again decrease towards zero at index $n_c(k+1)$.
The resulting matrix $\tilde{W}$ is shown in Figure 1.8. A black value on the grayscale denotes a coefficient value of 1.0. Note that the bandwidth (number of power spectrum coefficients within a window) increases with the center frequency of

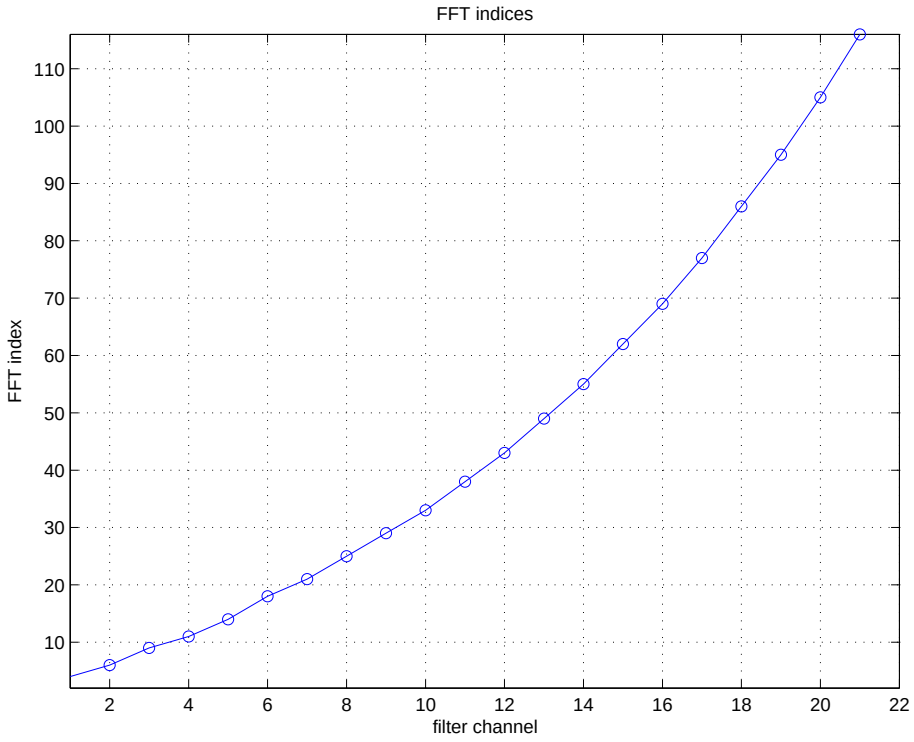

Figure 1.7: Indices of power spectrum vs mel spectral coefficient index.

the window.

1.4.4 Cepstral Transformation

Before we continue, lets remember how the spectrum of the speech signal was described by (1.2). Since the transmission function of the vocal tract $H(f)$ is multiplied with the spectrum of the excitation signal $X(f)$, we had those unwanted "ripples" in the spectrum. For the speech recognition task, a smoothed spectrum is required which should represent $H(f)$ but not $X(f)$. To cope with this problem, *cepstral analysis* is used. If we look at (1.2), we can separate the product of spectral functions into the interesting vocal tract spectrum and the part describing the excitation and emission properties:

$$S(f) = X(f) \cdot H(f) \cdot R(f) = H(f) \cdot U(f) \tag{1.13}$$

We can now transform the product of the spectral functions to a sum by taking the logarithm on both sides of the equation:

$$\begin{aligned} \log(S(f)) &= \log(H(f) \cdot U(f)) \\ &= \log(H(f)) + \log(U(f)) \end{aligned} \tag{1.14}$$

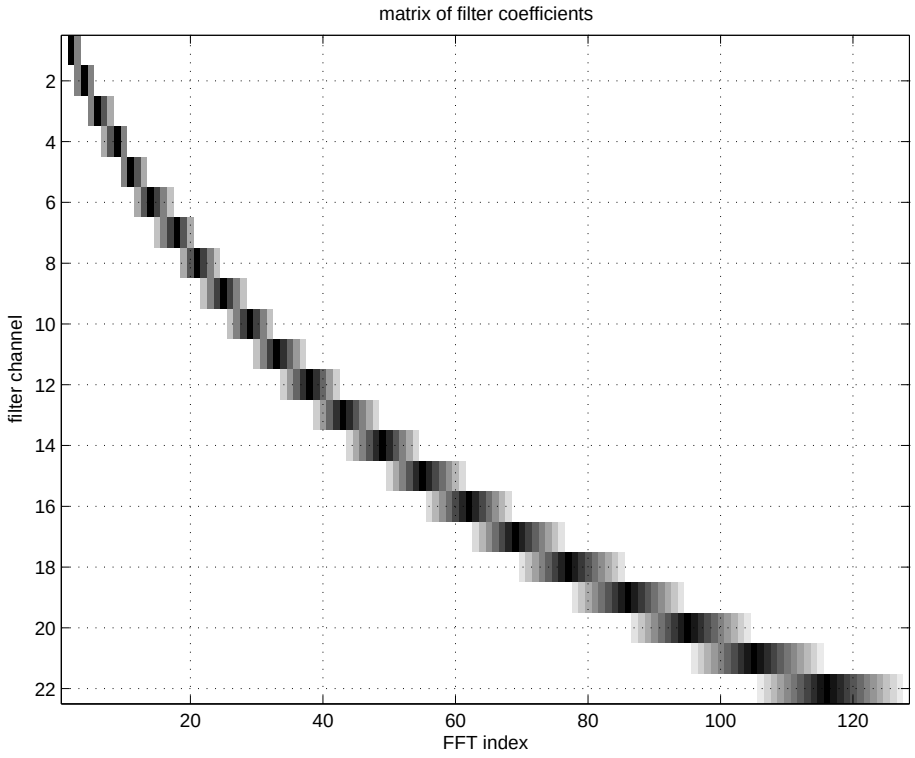

Figure 1.8: Matrix of coefficients η_{kn}.

This holds also for the absolute values of the power spectrum and also for their squares:

$$\begin{aligned} \log\left(|S(f)|^2\right) &= \log\left(|H(f)|^2 \cdot |U(f)|^2\right) \\ &= \log\left(|H(f)|^2\right) + \log\left(|U(f)|^2\right) \end{aligned} \tag{1.15}$$

In figure 1.9 we see an example of the log power spectrum, which contains unwanted ripples caused by the excitation signal $U(f) = X(f) \cdot R(f)$.
In the log–spectral domain we could now subtract the unwanted portion of the signal, if we knew $|U(f)|^2$ exactly. But all we know is that $U(f)$ produces the "ripples", which now are an additive component in the log–spectral domain, and that if we would interpret this log–spectrum as a time signal, the "ripples" would have a "high frequency" compared to the spectral shape of $|H(f)|$. To get rid of the influence of $U(f)$, one would have to get rid of the "high-frequency" parts of the log–spectrum (remember, we are dealing with the spectral coefficients as if they would represent a time signal). This would be a kind of low–pass filtering. The filtering can be done by transforming the log–spectrum back into the time–domain (in the following, $\mathcal{FT}^{-1}$ denotes the inverse Fourier transform):

$$\hat{s}(d) = \mathcal{FT}^{-1}\left\{\log\left(|S(f)|^2\right)\right\} = \mathcal{FT}^{-1}\left\{\log\left(|H(f)|^2\right)\right\} + \mathcal{FT}^{-1}\left\{\log\left(|U(f)|^2\right)\right\} \tag{1.16}$$

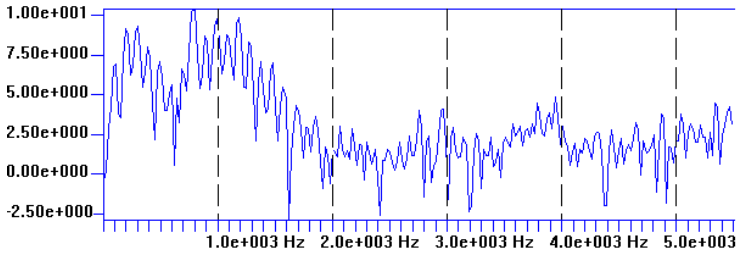

Figure 1.9: Log power spectrum of the vowel /a:/ ($f_s = 11$kHz, $N = 512$). The ripples in the spectrum are caused by $X(f)$.

The inverse Fourier transform brings us back to the time–domain (d is also called the delay or quefrency), giving the so–called *cepstrum* (a reversed "spectrum"). The resulting cepstrum is real–valued, since $|U(f)|^2$ and $|H(f)|^2$ are both real-valued and both are even: $|U(f)|^2 = |U(-f)|^2$ and $|H(f)|^2 = |H(-f)|^2$. Applying the inverse DFT to the log power spectrum coefficients $\log\left(|V(n)|^2\right)$ yields:

$$\hat{s}(d) = \frac{1}{N}\sum_{n=0}^{N-1} \log\left(|V(n)|^2\right) \cdot e^{j2\pi dn/N} \;;\; d = 0, 1, \ldots N-1 \tag{1.17}$$

Figure 1.10 shows the result of the inverse DFT applied on the log power spectrum shown in fig. 1.9.
The peak in the cepstrum reflects the ripples of the log power spectrum: Since the inverse Fourier transformation (1.17) is applied to the log power spectrum, an oscillation in the frequency domain with a Hertz-period of

$$p_f = n \cdot \Delta f = n \cdot \frac{f_s}{N} = \frac{n}{N \cdot \Delta t} \tag{1.18}$$

will show up in the cepstral domain as a peak at time t:

$$t = \frac{1}{p_f} = \frac{N}{n} \cdot \frac{1}{f_s} = \frac{N \cdot \Delta t}{n} \tag{1.19}$$

The cepstral index d of the peak expressed as a function of the period length n is:

$$d = \frac{t}{\Delta t} = \frac{N \cdot \Delta t}{n \cdot \Delta t} = \frac{N}{n} \tag{1.20}$$

The low–pass filtering of our energy spectrum can now be done by setting the higher-valued coefficients of $\hat{s}(d)$ to zero and then transforming back into the frequency domain. The process of filtering in the cepstral domain is also called *liftering*. In figure 1.10, all coefficients right of the vertical line were set to zero and the resulting signal was transformed back into the frequency domain, as shown in fig. 1.11.
One can clearly see that this results in a "smoothed" version of the log power spectrum if we compare figures 1.11 and 1.9.

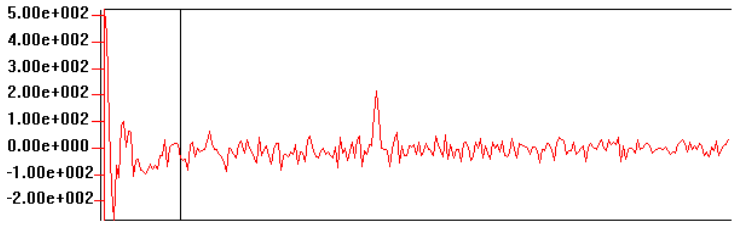

Figure 1.10: Cepstrum of the vowel /a:/ ($f_s = 11$kHz, $N = 512$). The ripples in the spectrum result in a peak in the cepstrum.

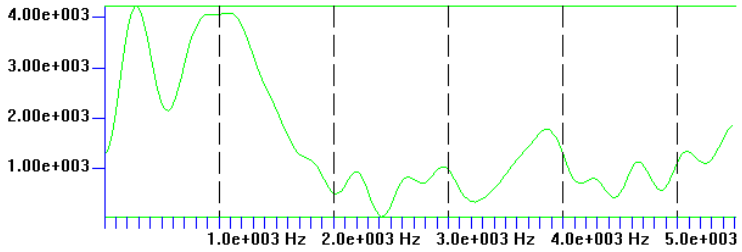

Figure 1.11: Power spectrum of the vowel /a:/ after cepstral smoothing. All but the first 32 cepstral coefficients were set to zero before transforming back into the frequency domain.

It should be noted that due to the symmetry (1.7) of $|V(n)|^2$, is is possible to replace the inverse DFT by the more efficient cosine transform [ST95]:

$$\hat{s}(0) = \sqrt{\frac{2}{N}} \sum_{n=0}^{N/2-1} \log\left(|V(n)|^2\right) \tag{1.21}$$

$$\hat{s}(d) = \sqrt{\frac{4}{N}} \sum_{n=0}^{N/2-1} \log\left(|V(n)|^2\right) \cdot \cos\left(\frac{\pi d\,(2n+1)}{N}\right) \;;\; d = 1, 2, ...N/2 \tag{1.22}$$

1.4.5 Mel Cepstrum

Now that we are familiar with the cepstral transformation and cepstral smoothing, we will compute the mel cepstrum commonly used in speech recognition. As stated above, for speech recognition, the mel spectrum is used to reflect the perception characteristics of the human ear. In analogy to computing the cepstrum, we now take the logarithm of the *mel* power spectrum (1.11) (instead of the power spectrum itself) and transform it into the quefrency domain to compute the so–called *mel cepstrum*. Only the first $\mathcal{Q}$ (less than 14) coefficients of the mel cepstrum are used in typical speech recognition systems. The restriction to the first $\mathcal{Q}$ coefficients reflects the low–pass liftering process as described above.

Since the mel power spectrum is symmetric due to (1.12), the Fourier-Transform can be replaced by a simple cosine transform:

$$c(q) = \sum_{k=0}^{\mathcal{K}-1} \log\left(G(k)\right) \cdot \cos\left(\frac{\pi q(2k+1)}{2\mathcal{K}}\right) \; ; \; q = 0, 1, \ldots, \mathcal{Q}-1 \tag{1.23}$$

While successive coefficients $G(k)$ of the mel power spectrum are correlated, the *Mel Frequency Cepstral Coefficients (MFCC)* resulting from the cosine transform (1.23) are decorrelated. The MFCC are used directly for further processing in the speech recognition system instead of transforming them back to the frequency domain.

1.4.6 Signal Energy

Furthermore, the signal energy is added to the set of parameters. It can simply be computed from the speech samples $s(n)$ within the time window by:

$$e = \sum_{n=0}^{N-1} s^2(n) \tag{1.24}$$

1.4.7 Dynamic Parameters

As stated earlier in eqn (1.5), the MFCC are computed for a speech segment v_m at time index m in short time intervals of typically 10ms. In order to better reflect the dynamic changes of the MFCC $c_m(q)$ (and also of the energy e_m) in time, usually the first and second derivatives in time are also computed, e.g by computing the difference of two coefficients lying τ time indices in the past and in the future of the time index under consideration.
For the first derivative we get:

$$\Delta c_m(q) = c_{m+\tau}(q) - c_{m-\tau}(q) \; ; \; q = 0, 1, ..\mathcal{Q}-1 \tag{1.25}$$

The sevond derivative is computed from the differences of the first derivatives:

$$\Delta\Delta c_m(q) = \Delta c_{m+\tau}(q) - \Delta c_{m-\tau}(q) \; ; \; q = 0, 1, ..\mathcal{Q}-1 \tag{1.26}$$

The time intervall usually lies in the range $2 \leq \tau \leq 4$.
This results in a total number of up to 63 parameters (e.g., [Ney93]) which are computed every 10ms. Of course, the choice of parameters for acoustic preprocessing has a strong impact on the performance of the speech recognition systems. One can spend years of research on the acoustic preprocessing modules of a speech recognition system, and many Ph.D. theses have been written on that subject. For our purposes however, it is sufficient to remember that the information contained in the speech signal can be represented by a set of parameters which has to be measured in short intervals of time to reflect the dynamic change of those parameters.

Chapter 2

Features and Vector Space

Until now, we have seen that the speech signal can be characterized by a set of parameters (features), which will be measured in short intervals of time during a preprocessing step. Before we start to look at the speech recognition task, we will first get familiar with the concept of feature vectors and vector space.

2.1 Feature Vectors and Vector Space

If you have a set of numbers representing certain features of an object you want to describe, it is useful for further processing to construct a vector out of these numbers by assigning each measured value to one component of the vector. As an example, think of an air conditioning system which will measure the temperature and relative humidity in your office. If you measure those parameters every second or so and you put the temperature into the first component and the humidity into the second component of a vector, you will get a series of two–dimensional vectors describing how the air in your office changes in time. Since these so–called feature vectors have two components, we can interpret the vectors as points in a two–dimensional vector space. Thus we can draw a two–dimensional map of our measurements as sketched below. Each point in our map represents the temperature and humidity in our office at a given time. As we know, there are certain values of temperature and humidity which we find more comfortable than other values. In the map the comfortable value–pairs are shown as points labelled "+" and the less comfortable ones are shown as "-". You can see that they form regions of convenience and inconvenience, respectively.

2.2 A Simple Classification Problem

Lets assume we would want to know if a value–pair we measured in our office would be judged as comfortable or as uncomfortable by you. One way to find out is to initially run a test series trying out many value–pairs and labelling each point either "+" or "-" in order to draw a map as the one you saw above. Now if you have measured a new value–pair and you are to judge if it will be convenient or not to a person, you would have to judge if it lies within those regions which are marked in your map as "+" or if it lies in those marked as "-". This is our

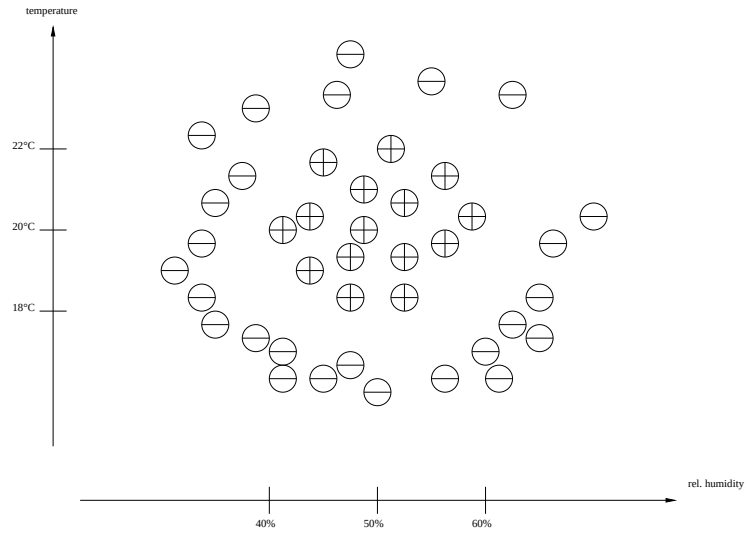

Figure 2.1: A map of feature vectors

first example of a classification task: We have two classes ("comfortable" and "uncomfortable") and a vector in feature space which has to be assigned to one of these classes. — But how do you describe the shape of the regions and how can you decide if a measured vector lies within or without a given region? In the following chapter we will learn how to represent the regions by prototypes and how to measure the distance of a point to a region.

2.3 Classification of Vectors

2.3.1 Prototype Vectors

The problem of how to represent the regions of "comfortable" and "uncomfortable" feature vectors of our classification task can be solved by several approaches. One of the easiest is to select several of the feature vectors we measured in our experiments for each of our classes (in our example we have only two classes) and to declare the selected vectors as "*prototypes*" representing their class. We will later discuss how one can find a good selection of prototypes using the "k–means algorithm". For now, we simply assume that we were able to make a good choice of the prototypes, as shown in figure 2.2.

2.3.2 Nearest Neighbor Classification

The classification of an unknown vector is now accomplished as follows: Measure the distance of the unknown vector to all classes. Then assign the unknown vector to the class with the smallest distance. The distance of the unknown vector to a given class is defined as the smallest distance between the unknown vector and all of the prototypes representing the given class. One could also verbalize the classification task as: Find the nearest prototype to the unknown vector and assign the unknown vector to the class this "nearest neighbor" represents

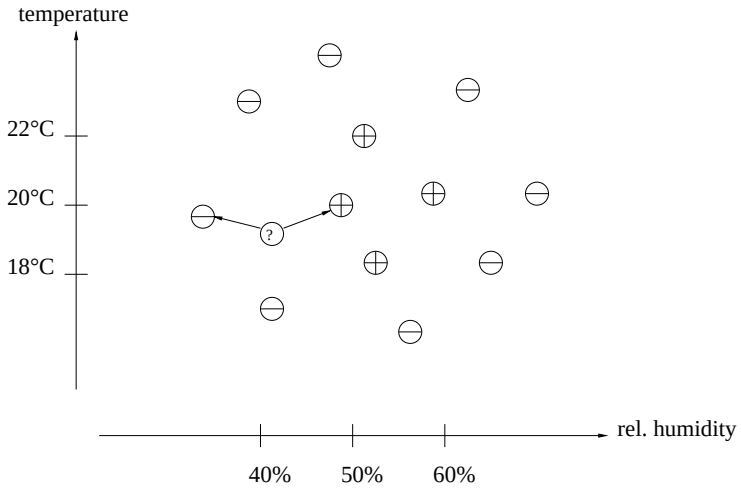

Figure 2.2: Selected prototypes

(hence the name). Fig. 2.2 shows the unknown vector and the two "nearest neighbors" of prototypes of the two classes. The classification task we described can be formalized as follows: Let $\Omega = \left\{\omega_1, \omega_2, \ldots, \omega_{(V-1)}\right\}$ be the set of classes, V being the total number of classes. Each class is represented by its prototype vectors $\vec{p}_{k,\omega_v}$, where $k = 0, 1, \ldots, (K_{\omega_v} - 1)$. Let $\vec{x}$ denote the unclassified vector. Let the distance measure between the vector and a prototype be denoted as $d\left(\vec{x}, \vec{p}_{k,\omega_v}\right)$ (eg., the Euclidean distance. We will discuss several distance measures later) . Then the class distance between $\vec{x}$ and the class ω_v is defined as:

$$d_{\omega_v}\left(\vec{x}\right) = \min_k \left\{d\left(\vec{x}, \vec{p}_{k,\omega_v}\right)\right\}; k = 0, 1, \ldots, (K_{\omega_v} - 1) \tag{2.1}$$

Using this class distance, we can write the classification task as follows:

$$\vec{x} \in \omega_v \Leftrightarrow v = \arg\min_v \left\{d_{\omega_v}\left(\vec{x}\right)\right\} \tag{2.2}$$

Chapter 3

Distance Measures

So far, we have found a way to classify an unknown vector by calculation of its class–distances to predefined classes, which in turn are defined by the distances to their individual prototype vectors. Now we will briefly look at some commonly used distance measures. Depending on the application at hand, each of the distance measures has its pros and cons, and we will discuss their most important properties.

3.1 Euclidean Distance

The Euclidean distance measure is the "standard" distance measure between two vectors in feature space (with dimension DIM) as you know it from school:

$$d^2_{Euclid}\left(\vec{x},\vec{p}\right) = \sum_{i=0}^{DIM-1} \left(x_i - p_i\right)^2 \tag{3.1}$$

To calculate the Euclidean distance measure, you have to compute the sum of the squares of the differences between the individual components of $\vec{x}$ and $\vec{p}$. This can also be written as the following scalar product:

$$d^2_{Euclid}\left(\vec{x},\vec{p}\right) = \left(\vec{x}-\vec{p}\right)' \cdot \left(\vec{x}-\vec{p}\right) \tag{3.2}$$

where $'$ denotes the vector transpose. Note that both equations (3.1) and (3.2) compute the square of the Euclidean distance, d^2 instead of d. The Euclidean distance is probably the most commonly used distance measure in pattern recognition.

3.2 City Block Distance

The computation of the Euclidean distance involves computing the squares of the individual differences thus involving many multiplications. To reduce the computational complexity, one can also use the absolute values of the differences instead of their squares. This is similar to measuring the distance between two points on a street map: You go three blocks to the East, then two blocks to the South (instead of straight trough the buildings as the Euclidean distance would

assume). Then you sum up all the absolute values for all the dimensions of the vector space:

$$d_{City-Block}(\vec{x},\vec{p}) = \sum_{i=0}^{DIM-1} |x_i - p_i| \tag{3.3}$$

3.3 Weighted Euclidean Distance

Both the Euclidean distance and the City Block distance are treating the individual dimensions of the feature space equally, i.e., the distances in each dimension contributes in the same way to the overall distance. But if we remember our example from section 2.1, we see that for real–world applications, the individual dimensions will have different scales also. While in our office the temperature values will have a range of typically between 18 and 22 degrees Celsius, the humidity will have a range from 40 to 60 percent relative humidity. While a small difference in humidity of e.g., 4 percent relative humidity might not even be noticed by a person, a temperature difference of 4 degrees Celsius certainly will. In Figure 3.1 we see a more abstract example involving two classes and two dimensions. The dimension x_1 has a wider range of values than dimension x_2, so all the measured values (or prototypes) are spread wider along the axis denoted as "x_1" as compared to axis "x_2". Obviously, an Euclidean or City Block distance measure would give the wrong result, classifying the unknown vector as "class A" instead of "class B" which would (probably) be the correct result.

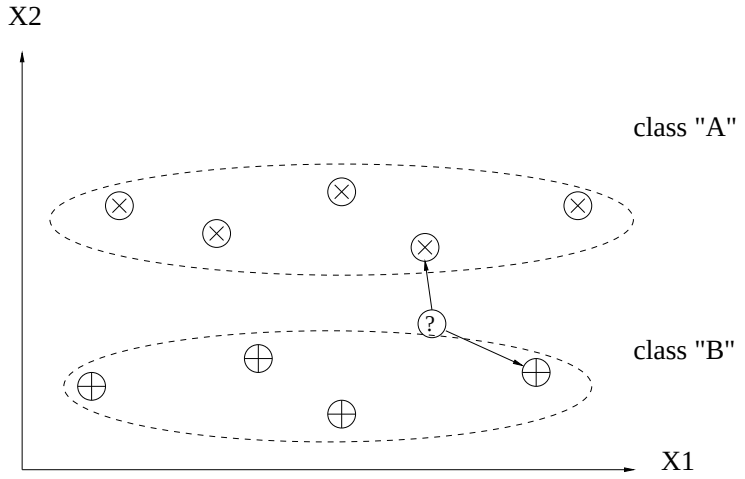

Figure 3.1: Two dimensions with different scales

To cope with this problem, the different scales of the dimensions of our feature vectors have to be compensated when computing the distance. This can be done by multiplying each contributing term with a scaling factor specific for the respective dimension. This leads us to the so–called *"Weighted Euclidean Distance"*:

$$d^2_{Weighted\_Euclid}(\vec{x},\vec{p}) = \sum_{i=0}^{DIM-1} \lambda_i \cdot (x_i - p_i)^2 \tag{3.4}$$

As before, this can be rewritten as:

$$d^2_{Weighted\_Euclid}(\vec{x},\vec{p}) = (\vec{x}-\vec{p})' \cdot \tilde{\Lambda} \cdot (\vec{x}-\vec{p}) \tag{3.5}$$

Where $\tilde{\Lambda}$ denotes the diagonal matrix of all the scaling factors:

$$\tilde{\Lambda} = \text{diag}\begin{bmatrix} \lambda_0 & & & & \\ & \cdots & & & \\ & & \lambda_i & & \\ & & & \cdots & \\ & & & & \lambda_{DIM-1} \end{bmatrix}$$

The scaling factors are usually chosen to compensate the variances of the measured features:

$$\lambda_i = \frac{1}{{\sigma_i}^2}$$

The variance of dimension i is computed from a training set of N vectors $\{\vec{x}_0, \vec{x}_1, \ldots, \vec{x}_{N-1}\}$. Let $x_{i,n}$ denote the i-th element of vector $\vec{x}_n$, then the variances ${\sigma_i}^2$ can be estimated from the training set as follows::

$${\sigma_i}^2 = \frac{1}{N-1} \sum_{n=0}^{N-1} (x_{i,n} - m_i)^2 \tag{3.6}$$

where m_i is the mean value of the training set for dimension i:

$$m_i = \frac{1}{N} \sum_{n=0}^{N-1} x_{i,n} \tag{3.7}$$

3.4 Mahalanobis Distance

So far, we can deal with different scales of our features using the weighted Euclidean distance measure. This works very well if there is no correlation between the individual features as it would be the case if the features we selected for our vector space were statistically independent from each other. What if they are not? Figure 3.2 shows an example in which the features x_1 and x_2 are correlated.

Obviously, for both classes A and B, a high value of x_1 correlates with a high value for x_2 (with respect to the mean vector (center) of the class), which is indicated by the orientation of the two ellipses. In this case, we would want the distance measure to regard both the correlation and scale properties of the features. Here a simple scale transformation as in (3.4) will not be sufficient. Instead, the correlations between the individual components of the feature vector will have to be regarded when computing the distance between two vectors. This leads us to a new distance measure, the so–called *Mahalanobis Distance*:

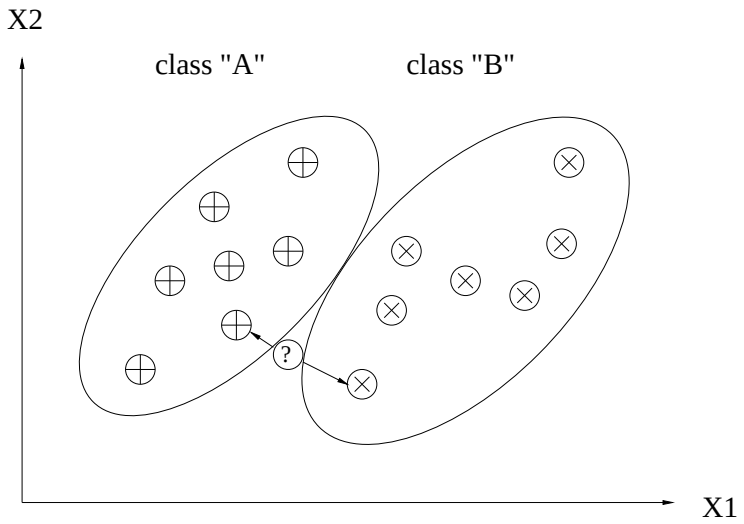

Figure 3.2: Correlated features

$$d_{Mahalanobis}\left(\vec{x},\vec{p},\tilde{C}\right) = (\vec{x}-\vec{p})' \cdot \tilde{C}^{-1} \cdot (\vec{x}-\vec{p}) \tag{3.8}$$

Where $\tilde{C}^{-1}$ denotes the inverse of the covariance matrix $\tilde{C}$:

$$\tilde{C} = \begin{bmatrix} \sigma_{0,0} & \dots & \sigma_{0,j} & \dots & \sigma_{0,DIM-1} \\ \vdots & & \vdots & & \vdots \\ \sigma_{i,0} & \dots & \sigma_{i,j} & \dots & \sigma_{i,DIM-1} \\ \vdots & & \vdots & & \vdots \\ \sigma_{DIM-1,0} & \dots & \sigma_{DIM-1,j} & \dots & \sigma_{DIM-1,DIM-1} \end{bmatrix}$$

The individual components $\sigma_{i,j}$ represent the covariances between the components i and j. The covariance matrix can be estimated from a training set of N vectors $\{\vec{x}_0, \vec{x}_1, \dots, \vec{x}_{N-1}\}$ as follows:

$$\tilde{C} = \frac{1}{N-1} \sum_{n=0}^{N-1} (\vec{x}_n - \vec{m}) \cdot (\vec{x}_n - \vec{m})' \tag{3.9}$$

where $\vec{m}$ is the mean vector of the training set:

$$\vec{m} = \frac{1}{N} \sum_{n=0}^{N-1} \vec{x}_n \tag{3.10}$$

looking at a single element of the matrix $\tilde{C}$, say, $\sigma_{i,j}$, this can also be written as:

$$\sigma_{i,j} = \frac{1}{N-1} \sum_{n=0}^{N-1} (x_{i,n} - m_i) \cdot (x_{j,n} - m_j) \tag{3.11}$$

From (3.6) and (3.11) it is obvious that $\sigma_{i,i}$, the values of the main diagonal of $\tilde{C}$, are identical to the variances ${\sigma_i}^2$ of the dimension i. Two properties of $\tilde{C}$ can be seen immediately form (3.11): First, $\tilde{C}$ is symmetric and second, the values $\sigma_{i,i}$ of its main diagonal are either positive or zero. A zero variance would mean that for this specific dimension of the vector we chose a feature which does not change its value at all).

3.4.1 Properties of the Covariance Matrix

How can we interpret the elements of the covariance matrix? For the $\sigma_{i,i}$, we already know that they represent the variances of the individual features. A big variance for one dimension reflects the fact that the measured values of this feature tend to be spread over a wide range of values with respect to the mean value, while a variance close to zero would indicate that the values measured are all nearly identical and therefore very close to the mean value. For the covariances $\sigma_{i,j}$, $i \neq j$, we will expect a positive value of $\sigma_{i,j}$ if a big value for feature i can be frequently observed together with a big value for feature j (again, with respect to the mean value) and observations with small values for feature i also tend to show small values for feature j . In this case, for a two–dimensional vector space example, the clusters look like the two clusters of the classes A and B in Fig. 3.2, i.e., they are oriented from bottom–left to top–right. On the other hand, if a cluster is oriented from top–left to bottom–right, high values for feature i often are observed with low values for feature j and the value for the covariance $\sigma_{i,j}$ would be negative (due to the symmetry of $\tilde{C}$, you can swap the indices i and j in the description above).

3.4.2 Clusters With Individual and Shared Covariance Matrices

In our Fig. 3.2 the two classes A and B each consisted of one cluster of vectors having its own center (mean vector, that is) and its own covariance matrix. But in general, each class of a classification task may consist of several clusters which belong to that class, if the "shape" of the class in our feature space is more complex. As an example, think of the shape of the "uncomfortable" class of Fig. 2.1. In these cases, each of those clusters will have its own mean vector (the "prototype vector" representing the cluster) and its own covariance matrix. However, in some applications, it is sufficient (or necessary, due to a lack of training data) to estimate one covariance matrix out of all training vectors of one class, regardless to which cluster of the class the vectors are assigned. The resulting covariance matrix is then class specific but is shared among all clusters of the class while the mean vectors are estimated individually for the clusters. To go even further, it is also possible to estimate a single "global" covariance matrix which is shared among all classes (and their clusters). This covariance matrix is computed out of the complete trainig set of all vectors regardless of the cluster and class assignments.

3.4.3 Using the Mahalanobis Distance for the Classification Task

With the Mahalanobis distance, we have found a distance measure capable of dealing with different scales and correlations between the dimensions of our feature space. In section 7.3, we will learn that it plays a significant role when dealing with the gaussian probability density function.
To use the Mahalanobis distance in our nearest neighbor classification task, we would not only have to find the appropriate prototype vectors for the classes (that is, finding the mean vectors of the clusters), but for each cluster we also have to compute the covariance matrix from the set of vectors which belong to the cluster. We will later learn how the "k–means algorithm" helps us not only in finding prototypes from a given training set, but also in finding the vectors that belong to the clusters represented by these prototypes. Once the prototype vectors and the corresponding (either shared or individual) covariance matrices are computed, we can insert the Mahalanobis distance according to (3.8) into the equation for the class distance (2.1) and get:

$$d_{\omega_v}(\vec{x}) = \min_k \left\{ d_{Mahalanobis}\left(\vec{x}, \vec{p}_{k,\omega_v}, \tilde{C}_{k,\omega_v}\right) \right\}; k = 0, 1, \ldots, (K_{\omega_v} - 1) \quad (3.12)$$

Then we can perform the Nearest Neighbor classification as in (2.2).

Chapter 4

Dynamic Time Warping

In the last chapter, we were dealing with the task of classifying single vectors to a given set of classes which were represented by prototype vectors computed from a set of training vectors. Several distance measures were presented, some of them using additional sets of parameters (e.g., the covariance matrices) which also had to be computed from a training vectors.
How does this relate to speech recognition?
As we saw in chapter 1, our speech signal is represented by a series of feature vectors which are computed every 10ms. A whole word will comprise dozens of those vectors, and we know that the number of vectors (the duration) of a word will depend on how fast a person is speaking. Therefore, our classification task is different from what we have learned before. In speech recognition, we have to classify not only single vectors, but *sequences* of vectors. Lets assume we would want to recognize a few command words or digits. For an utterance of a word w which is T_X vectors long, we will get a sequence of vectors $\tilde{X} = \{\vec{x}_0, \vec{x}_1, \ldots, \vec{x}_{T_X-1}\}$ from the acoustic preprocessing stage. What we need here is a way to compute a "distance" between this unknown sequence of vectors $\tilde{X}$ and known sequences of vectors $\tilde{W}_k = \left\{\vec{w_k}_0, \vec{w_k}_1, \ldots, \vec{w_k}_{T_{W_k}}\right\}$ which are prototypes for the words we want to recognize. Let our vocabulary (here: the set of classes Ω) contain V different words $w_0, w_1, \ldots w_{V-1}$. In analogy to the Nearest Neighbor classification task from chapter 2, we will allow a word w_v (here: class $\omega_v \in \Omega$) to be represented by a set of prototypes $\tilde{W}_{k,\omega_v}, k = 0, 1, \ldots, (K_{\omega_v} - 1)$ to reflect all the variations possible due to different pronunciation or even different speakers.

4.1 Definition of the Classification Task

If we have a suitable distance measure $D(\tilde{X}, \tilde{W}_{k,\omega_v})$ between $\tilde{X}$ and a prototype $\tilde{W}_{k,\omega_v}$, we can define the class distance $D_{\omega_v}(\tilde{X})$ in analogy to (2.2) as follows:

$$D_{\omega_v}(\tilde{X}) = \min_k \left\{ D\left(\tilde{X}, \tilde{W}_{k,\omega_v}\right) \right\}; k = 0, 1, \ldots, (K_{\omega_v} - 1) \tag{4.1}$$

where ω_v is the class of utterances whose orthographic representation is given by the word w_v
Then we can write our classification task as:

$$\tilde{X} \in \omega_v \Leftrightarrow v = \arg\min_v \left\{ D_{\omega_v} \left(\tilde{X} \right) \right\} \tag{4.2}$$

This definition implies that the vocabulary of our speech recognizer is limited to the set of classes $\Omega = \left\{ \omega_0, \omega_1, \ldots, \omega_{(V-1)} \right\}$. In other words, we can only recognize V words, and our classifier will match any speech utterance to one of those V words in the vocabulary, even if the utterance was intended to mean completely different. The only criterium the classifier applies for its choice is the acoustic similarity between the utterance $\tilde{X}$ and the known prototypes $\tilde{W}_{k,\omega_v}$ as defined by the distance measure $D(\tilde{X}, \tilde{W}_{k,\omega_v})$.

4.2 Distance Between Two Sequences of Vectors

As we saw before, classification of a spoken utterance would be easy if we had a good distance measure $D(\tilde{X}, \tilde{W})$ at hand (in the following, we will skip the additional indices for ease of notation). What should the properties of this distance measure be ? The distance measure we need must:

- Measure the distance between two sequences of vectors of different length (T_X and T_W, that is)
- While computing the distance, find an optimal assignment between the individual feature vectors of $\tilde{X}$ and $\tilde{W}$
- Compute a total distance out of the sum of distances between individual pairs of feature vectors of $\tilde{X}$ and $\tilde{W}$

4.2.1 Comparing Sequences With Different Lengths

The main problem is to find the optimal assignment between the individual vectors of $\tilde{X}$ and $\tilde{W}$. In Fig. 4.1 we can see two sequences $\tilde{X}$ and $\tilde{W}$ which consist of six and eight vectors, respectively. The sequence $\tilde{W}$ was rotated by 90 degrees, so the time index for this sequence runs from the bottom of the sequence to its top. The two sequences span a grid of possible assignments between the vectors. Each path through this grid (as the path shown in the figure) represents one possible assignment of the vector pairs. For example, the first vector of $\tilde{X}$ is assigned the first vector of $\tilde{W}$, the second vector of $\tilde{X}$ is assigned to the second vector of $\tilde{W}$, and so on.
Fig. 4.1 shows as an example the following path P given by the sequence of time index pairs of the vector sequences (or the grid point indices, respectively):

$$P = \left\{ g_1, g_2, \ldots, g_{L_p} \right\} = \{(0,0), (1,1), (2,2), (3,2), (4,2), (5,3), (6,4), (7,4)\} \tag{4.3}$$

The length L_P of path P is determined by the maximum of the number of vectors contained in $\tilde{X}$ and $\tilde{W}$. The assignment between the time indices of $\tilde{W}$ and $\tilde{X}$ as given by P can be interpreted as *"time warping"* between the time axes of $\tilde{W}$ and $\tilde{X}$. In our example, the vectors $\vec{x}_2$, $\vec{x}_3$ and $\vec{x}_4$ were all assigned to $\vec{w}_2$, thus warping the duration of $\vec{w}_2$ so that it lasts three time indices instead of one. By this kind of time warping, the different lengths of the vector sequences can be compensated.

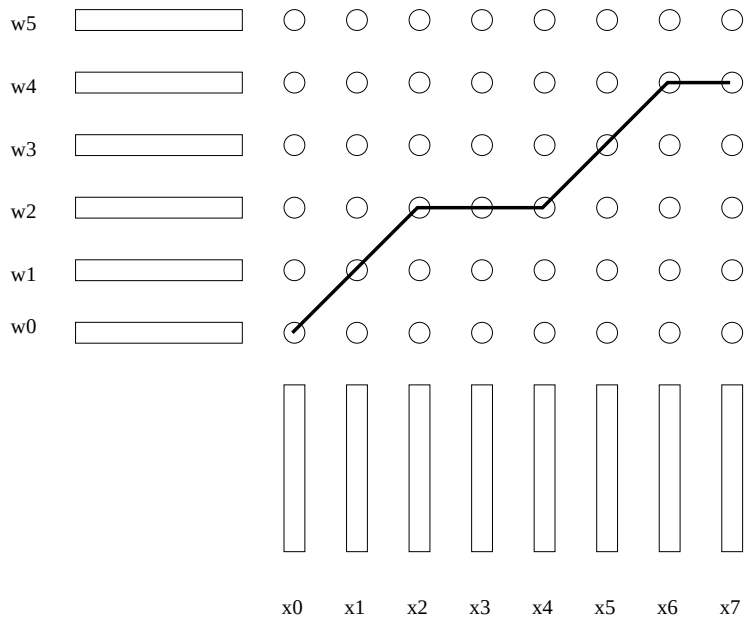

Figure 4.1: Possible assignment between the vector pairs of $\tilde{X}$ and $\tilde{W}$

For the given path P, the distance measure between the vector sequences can now be computed as the sum of the distances between the individual vectors. Let l denote the sequence index of the grid points. Let $d(g_l)$ denote the vector distance $d(\vec{x}_i, \vec{w}_j)$ for the time indices i and j defined by the grid point $g_l = (i, j)$ (the distance $d(\vec{x}_i, \vec{w}_j)$ could be the Euclidean distance from section 3.1). Then the overall distance can be computed as:

$$D\left(\tilde{X}, \tilde{W}, P, \right) = \sum_{l=1}^{L_P} d(g_l) \tag{4.4}$$

4.2.2 Finding the Optimal Path

As we stated above, once we have the path, computing the distance D is a simple task. How do we find it ?
The criterium of optimality we want to use in searching the optimal path P_{opt} should be to minimize $D\left(\tilde{X}, \tilde{W}, P\right)$:

$$P_{opt} = \arg\min_{P} \left\{ D\left(\tilde{X}, \tilde{W}, P\right) \right\} \tag{4.5}$$

Fortunately, it is not necessary to compute *all* possible paths P and corresponding distances $D\left(\tilde{X}, \tilde{W}, P\right)$ to find the optimum.
Out of the huge number of theoretically possible paths, only a fraction is reasonable for our purposes. We know that both sequences of vectors represent feature vectors measured in short time intervals. Therefore, we might want to restrict the time warping to reasonable boundaries: The first vectors of $\tilde{X}$ and

$\tilde{W}$ should be assigned to each other as well as their last vectors. For the time indices in between, we want to avoid any giant leap backward or forward in time, but want to restrict the time warping just to the "reuse" of the preceding vector(s) to locally warp the duration of a short segment of speech signal. With these restrictions, we can draw a diagram of possible "local" path alternatives for one grid point and its possible predecessors (of course, many other local path diagrams are possible):

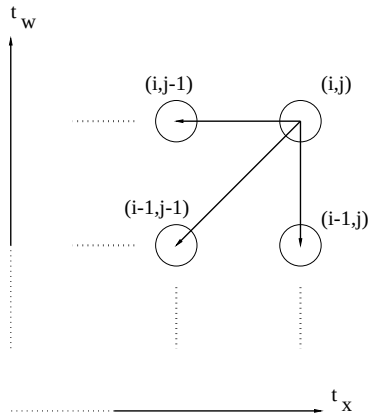

Figure 4.2: Local path alternatives for a grid point

Note that Fig. 4.2 does not show the possible extensions of the path from a given point but the possible *predecessor paths* for a given grid point. We will soon get more familiar with this way of thinking. As we can see, a grid point (i, j) can have the following predecessors:

- $(i-1, j)$: keep the time index j of $\tilde{X}$ while the time index of $\tilde{W}$ is incremented
- $(i-1, j-1)$: both time indices of $\tilde{X}$ and $\tilde{W}$ are incremented
- $(i, j-1)$: keep of the time index i of $\tilde{W}$ while the time index of $\tilde{X}$ is incremented

All possible paths P which we will consider as possible candidates for being the optimal path P_{opt} can be constructed as a concatenation of the local path alternatives as described above. To reach a given grid point (i, j) from $(i-1, j-1)$, the diagonal transition involves only the single vector distance at grid point (i, j) as opposed to using the vertical or horizontal transition, where also the distances for the grid points $(i-1, j)$ or $(i, j-1)$ would have to be added. To compensate this effect, the local distance $d(\vec{w}_i, \vec{x}_j)$ is added twice when using the diagonal transition.

Bellman's Principle

Now that we have defined the local path alternatives, we will use *Bellman's Principle* to search the optimal path P_{opt}.
Applied to our problem, Bellman's Principle states the following:

***If** P_{opt} **is the optimal path through the matrix of grid points beginning at** $(0,0)$ **and ending at** (T_W-1, T_X-1)**, and the grid point** (i,j) **is part of path** P_{opt}**, then the partial path from** $(0,0)$ **to** (i,j) **is also part of** P_{opt}**.*

From that, we can construct a way of iteratively finding our optimal path P_{opt}: According to the local path alternatives diagram we chose, there are only three possible predecessor paths leading to a grid point (i,j): The partial paths from $(0,0)$ to the grid points $(i-1,j)$, $(i-1,j-1)$ and $(i,j-1)$).
Let's assume we would know the optimal paths (and therefore the accumulated distance $\delta(.)$ along that paths) leading from $(0,0)$ to these grid points. All these *path hypotheses* are possible predecessor paths for the optimal path leading from $(0,0)$ to (i,j).
Then we can find the (globally) optimal path from $(0,0)$ to grid point (i,j) by selecting exactly the one path hypothesis among our alternatives which minimizes the accumulated distance $\delta(i,j)$ of the resulting path from $(0,0)$ to (i,j). The optimization we have to perform is as follows:

$$\delta(i,j) = \min \begin{cases} \delta(i,j-1) & + & d(\vec{w}_i, \vec{x}_j) \\ \delta(i-1,j-1) & + & 2 \cdot d(\vec{w}_i, \vec{x}_j) \\ \delta(i-1,j) & + & d(\vec{w}_i, \vec{x}_j) \end{cases} \tag{4.6}$$

By this optimization, it is ensured that we reach the grid point (i,j) via the optimal path beginning from $(0,0)$ and that therefore the accumulated distance $\delta(i,j)$ is the minimum among all possible paths from $(0,0)$ to (i,j).
Since the decision for the best predecessor path hypothesis reduces the number of paths leading to grid point (i,j) to exactly one, it is also said that the possible path hypotheses are *recombined* during the optimization step.
Applying this rule, we have found a way to iteratively compute the optimal path from point $(0,0)$ to point (T_W-1, T_X-1), starting with point $(0,0)$:

- Initialization: For the grid point $(0,0)$, the computation of the optimal path is simple: It contains only grid point $(0,0)$ and the accumulated distance along that path is simply $d(\vec{w}_0, \vec{x}_0)$.
- Iteration: Now we have to expand the computation of the partial paths to all grid points until we reach (T_W-1, T_X-1): According to the local path alternatives, we can now only compute two points from grid point $(0,0)$: The upper point $(1,0)$, and the right point $(0,1)$. Optimization of (4.6) is easy in these cases, since there is only one valid predecessor: The start point (0.0). So we add $\delta(0,0)$ to the vector distances defined by the grid points $(1,0)$ and $(0,1)$ to compute $\delta(1,0)$ and $\delta(0,1)$. Now we look at the points which can be computed from the three points we just finished. For each of these points (i,j), we search the optimal predecessor point out of the set of possible predecessors (Of course, for the leftmost column $(i,0)$ and the bottom row $(0,j)$ the recombination of path hypotheses is always trivial). That way we walk trough the matrix from bottom-left to top-right.
- Termination: $D(\tilde{W}, \tilde{X}) = \delta(T_W-1, T_X-1)$ is the distance between $\tilde{W}$ and $\tilde{X}$.

Fig. 4.3 shows the iteration through the matrix beginning with the start point $(0, 0)$. Filled points are already computed, empty points are not. The dotted arrows indicate the possible path hypotheses over which the optimization (4.6) has to be performed. The solid lines show the resulting partial paths after the decision for one of the path hypotheses during the optimization step. Once we reached the top–right corner of our matrix, the accumulated distance $\delta(T_W - 1, T_X - 1)$ is the distance $D(\tilde{W}, \tilde{X})$ between the vector sequences. If we are also interested in obtaining not only the distance $D(\tilde{W}, \tilde{X})$, but also the optimal path P, we have — in addition to the accumulated distances — also to keep track of all the decisions we make during the optimization steps (4.6). The optimal path is known only after the termination of the algorithm, when we have made the last recombination for the three possible path hypotheses leading to the top–right grid point $(T_W - 1, T_X - 1)$. Once this decision is made, the optimal path can be found by reversely following all the local decisions down to the origin $(0, 0)$. This procedure is called *backtracking*.

4.2.3 Dynamic Programming / Dynamic Time Warping

The procedure we defined to compare two sequences of vectors is also known as *Dynamic Programming* (DP) or as *Dynamic Time Warping* (DTW). We will use it extensively and will add some modifications and refinements during the next chapters. In addition to the verbal description in the section above, we will now define a more formal framework and will add some hints to possible realizations of the algorithm.

The Dynamic Programming Algorithm

In the following formal framework we will iterate through the matrix column by column, starting with the leftmost column and beginning each column at the bottom and continuing to the top.
For ease of notation, we define $d(i, j)$ to be the distance $d(\vec{w}_i, \vec{x}_j)$ between the two vectors $\vec{w}_i$ and $\vec{x}_i$.
Since we are iterating through the matrix from left to right, and the optimization for column j according to (4.6) uses only the accumulated distances from columns j and $j-1$, we will use only two arrays $\delta_j(i)$ and $\delta_{j-1}(i)$ to hold the values for those two columns instead of using a whole matrix for the accumulated distances $\delta(i, j)$:

Let $\delta_j(i)$ be the accumulated distance $\delta(i, j)$ at grid point (i, j) and $\delta_{j-1}(i)$ the accumulated distance $\delta(i, j - 1)$ at grid point $(i, j - 1)$.

It should be mentioned that it possible to use a single array for time indices j and $j - 1$. One can overwrite the old values of the array with the new ones. However, for clarity, the algorithm using two arrays is described here and the formulation for a single–array algorithm is left to the reader.
To keep track of all the selections among the path hypotheses during the optimization, we have to store each path alternative chosen for every grid point. We could for every grid point (i, j) either store the indices k and l of the predecessor point (k, l) or we could only store a code number for one of the three path alternatives (horizontal, diagonal and vertical path) and compute the predecessor point (k, l) out of the code and the current point (i, j). For ease of notation,

we assume in the following that the indices of the predecessor grid point will be stored:

Let $\psi(i,j)$ be the predecessor grid point (k,l) chosen during the optimization step at grid point (i,j).

With these definitions, the dynamic programming (DP)algorithm can be written as follows:

- **Initialization:**
  - Grid point $(0,0)$:
$$\begin{aligned} \psi(0,0) &= (-1,-1) \\ \delta_j(0) &= d(0,0) \end{aligned} \tag{4.7}$$
  - Initialize first column (only vertical path possible):
    for $i = 1$ **to** $T_W - 1$
    {
$$\begin{aligned} \delta_j(i) &= d(i,0) + \delta_j(i-1) \\ \psi(i,0) &= (i-1,0) \end{aligned} \tag{4.8}$$
    }
- **Iteration:**
  compute all columns:
  for $j = 1$ **to** $T_X - 1$
  {
  - swap arrays $\delta_{j-1}(.)$ and $\delta_j(.)$
  - first point ($i = 0$, only horizontal path possible):
$$\begin{aligned} \delta_j(0) &= d(0,j) + \delta_{j-1}(0) \\ \psi(0,j) &= (0,j-1) \end{aligned} \tag{4.9}$$
  - compute column j:
    for $i = 1$ **to** $T_W - 1$
    {
    - optimization step:
$$\delta_j(i) = \min \left\{ \begin{array}{rcl} \delta_{j-1}(i) & + & d(i,j) \\ \delta_{j-1}(i-1) & + & 2 \cdot d(i,j) \\ \delta_j(i-1) & + & d(i,j) \end{array} \right. \tag{4.10}$$
    - tracking of path decisions:
$$\psi(i,j) = \underset{(k,l) \in \left\{ \begin{array}{r} (i,j-1), \\ (i-1,j-1), \\ (i-1,j) \end{array} \right\}}{\arg\min} \left\{ \begin{array}{rcl} \delta_{j-1}(i) & + & d(i,j) \\ \delta_{j-1}(i-1) & + & 2 \cdot d(i,j) \\ \delta_j(i-1) & + & d(i,j) \end{array} \right. \tag{4.11}$$
    }

}

- **Termination:**

$$D(T_W - 1, T_X - 1) = \delta_j(T_W - 1, T_X - 1) \tag{4.12}$$

- **Backtracking:**

  - Initialization:

$$i = T_W - 1, j = T_X - 1 \tag{4.13}$$

  - **while** $\psi(i, j) \neq -1$
    {
    - get predecessor point:

$$i, j = \psi(i, j) \tag{4.14}$$

    }

Additional Constraints

Note that the algorithm above will find the optimum path by considering all path hypotheses through the matrix of grid points. However, there will be paths regarded during computation which are not very likely to be the optimum path due to the extreme time warping they involve. For example, think of the borders of the grid: There is a path going horizontally from $(0, 0)$ until $(T_W - 1, 0)$ and then running vertically to $(T_W - 1, T_X - 1)$. Paths running more or less diagonally trough the matrix will be much more likely candidates for being the optimal path. To avoid unnecessary computational efforts, the DP algorithm is therefore usually restricted to searching only in a certain region around the diagonal path leading from $(0, 0)$ to $(T_W - 1, T_X - 1)$. Of course, this approach takes the (small) risk of missing the globally optimal path and instead finding only the best path within the defined region. Later we will learn far more sophisticated methods to limit the so–called *search space* of the DP algorithm.

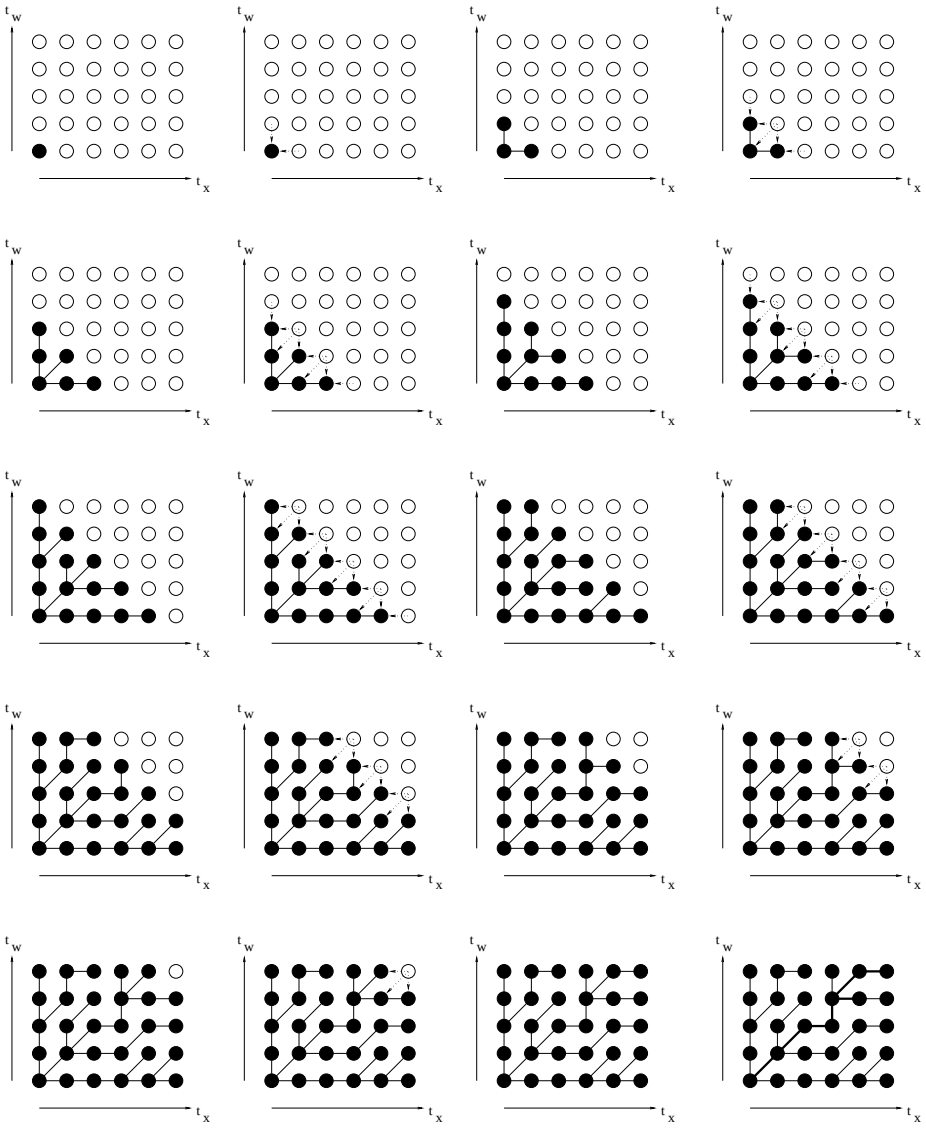

Figure 4.3: Iteration steps finding the optimal path

Chapter 5

Recognition of Isolated Words

In the last chapter we saw how we can compute the distance between two sequences of vectors to perform the classification task defined by (4.2). The classification was performed by computing the distance between an unknown vector sequence to all prototypes of all classes and then assigning the unknown sequence to the class having the smallest class distance.
Now we will reformulate the classification task so that it will depend directly on the optimum path chosen by the dynamic programming algorithm and not by the class distances.
While the description of the DTW classification algorithm in chapter 4 might let us think that one would compute all the distances sequentially and then select the minimum distance, it is more useful in practical applications to compute *all* the distances between the unknown vector sequence and the class prototypes *in parallel*. This is possible since the DTW algorithm needs only the values for time index t and $(t-1)$ and therefore there is no need to wait until the utterance of the unknown vector sequence is completed. Instead, one can start with the recognition process immediately as soon as the utterance begins (we will not deal with the question of how to recognize the start and end of an utterance here).
To do so, we have to reorganize our search space a little bit. First, lets assume the total number of all prototypes over all classes is given by M. If we want to compute the distances to all M prototypes simultaneously, we have to keep track of the accumulated distances between the unknown vector sequence and the prototype sequences individually. Hence, instead of the column (or two columns, depending on the implementation) we used to hold the accumulated distance values for all grid points, we now have to provide M columns during the DTW procedure.
Now we introduce an additional "virtual" grid point together with a specialized local path alternative for this point: The possible predecessors for this point are defined to be the upper–right grid points of the individual grid matrices of the prototypes. In other words, the virtual grid point can only be reached from the end of each prototype word, and among all the possible prototype words, the one with the smallest accumulated distance is chosen. By introducing this virtual

grid point, the classification task itself (selecting the class with the smallest class distance) is integrated into the framework of finding the optimal path.
Now all we have to do is to run the DTW algorithm for each time index j and along all columns of all prototype sequences. At the last time slot $(T_W - 1)$ we perform the optimization step for the virtual grid point, i.e, the predecessor grid point to the virtual grid point is chosen to be the prototype word having the smallest accumulated distance. Note that the search space we have to consider is spanned by the length of the unknown vector sequence on one hand and the sum of the length of all prototype sequences of all classes on the other hand.
Figure 5.1 shows the individual grids for the prototypes (only three are shown here) and the selected optimal path to the virtual grid point. The backtracking procedure can of course be restricted to keeping track of the final optimization step when the best predecessor for the virtual grid point is chosen. The classification task is then performed by assigning the unknown vector sequence to the very class to which the prototype belongs whose word end grid point was chosen.

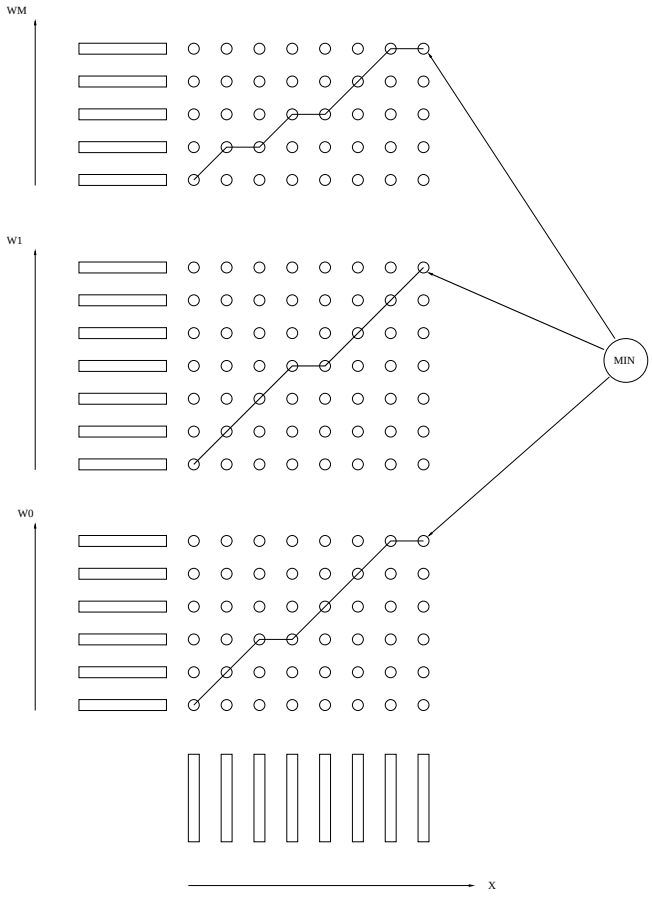

Figure 5.1: Classification task redefined as finding the optimal path among all prototype words

Of course, this is just a different (and quite complicated) definition of how we can perform the DTW classification task we already defined in (4.2). Therefore, only a verbal description was given and we did not bother with a formal description. However, by the reformulation of the DTW classification we learned a few things:

- The DTW algorithm can be used for real–time computation of the distances
- The classification task has been integrated into the search for the optimal path
- Instead of the accumulated distance, now the optimal path itself is important for the classification task

In the next chapter, we will see how the search algorithm described above can be extended to the recognition of sequences of words.

Chapter 6

Recognition of Connected Words

In the last chapter, we learned how to recognize isolated words, e.g., simple command words like "start", "stop", "yes', "no". The classification was performed by searching the optimal path through the search space, which was spanned by the individual prototype vector sequences on one hand and by the unknown vector sequence on the other hand.

What if we want to recognize sequences of words, like a credit card number or telephone number? In this case, the classification task can be divided into two different subtasks: The segmentation of the utterance, i.e., finding the boundaries between the words and the classification of the individual words within their boundaries. As one can imagine, the number of possible combinations of words of a given vocabulary together with the fact that each word may widely vary in its length provides for a huge number of combinations to be considered during the classification. Fortunately, there is a solution for the problem, which is able to find the word boundaries and to classify the words within the boundaries in one single step. Best of all, you already know the algorithm, because it is the Dynamic Programming algorithm again ! In the following, first a verbal description of what modifications are necessary to apply the DP algorithm on the problem of connected word recognition is given. Then a formal description of the algorithm is presented.

6.1 The Search Space for Connected Word Recognition

If we want to apply the DP algorithm to our new task, we will first have to define the search space for our new task. For simplicity, let's assume that each word of our vocabulary is represented by only one prototype, which will make the notation a bit easier for us. An unknown utterance of a given length T_X will contain several words out of our vocabulary, but we do not know which words these are and at what time index they begin or end. Therefore, we will have to match the utterance against all prototypes and we will have to test for

all possible word ends and word beginnings. The optimal path we want to find with our DP algorithm must therefore be allowed to run through a search space which is spanned by the set of all vectors of all prototypes in one dimension and by the vector sequence of the unknown utterance in the other dimension, as is shown in Fig. 6.1.

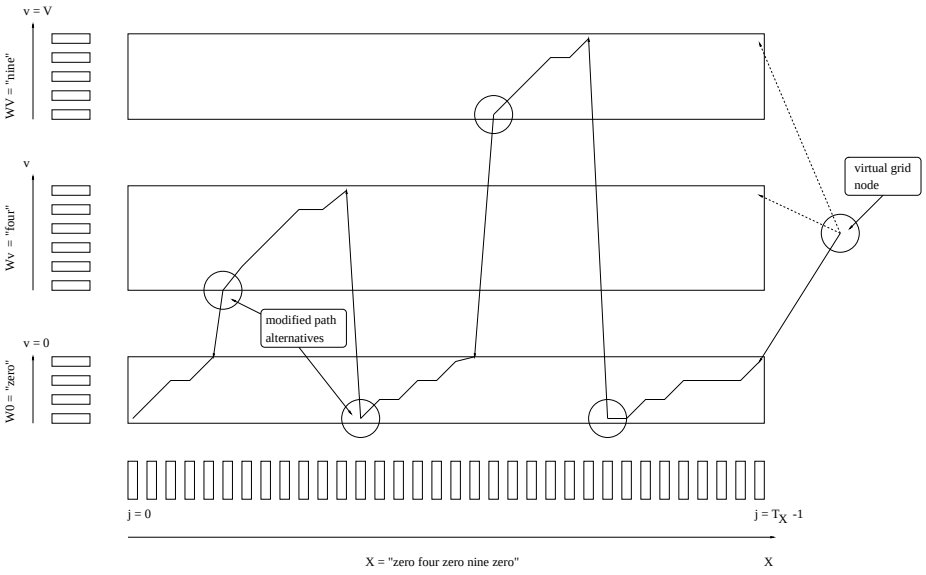

Figure 6.1: DTW algorithm for connected word recognition

6.2 Modification of the Local Path Alternatives

Remember that in our DP algorithm in section 4 we were dealing with the local path alternatives each grid could select as possible predecessors. The alternatives were including grid points one time index in the past (the "horizontal" and "diagonal" transitions) and in the present (the "vertical" transition). This allowed us to implement the DP algorithm in real–time, column per column.
Now we have to modify our definition of local path alternatives for the first vector of each prototype: The first vector of a prototype denotes the beginning of the word. Remember that in the case of isolated word recognition, the grid point corresponding to that vector was initialized in the first column with the distance between the first vector of the prototype and the first vector of the utterance. During the DP, i.e., while we were moving from column to column, this grid point had only one predecessor, it could only be preceded by the same grid point in the preceding column (the "horizontal" transition), indicating that the first vector of the prototype sequence was warped in time to match the next vector of the unknown utterance.
For connected word recognition however, the grid point corresponding to the first vector of a prototype has more path alternatives: It may either select the same grid point in the preceding column (this is the "horizontal" transition, which we already know), or it may select every *last grid point (word end, that*

is) of all prototypes (including the prototype itself) of the preceding column as a possible predecessor for between–word path recombination. In this case, the prototype word under consideration is assumed to start at the current time index and the word whose word end was chosen is assumed to end at the time index before.
Fig. 6.2 shows the local path alternatives for the first grid point of the column associated with a prototype.

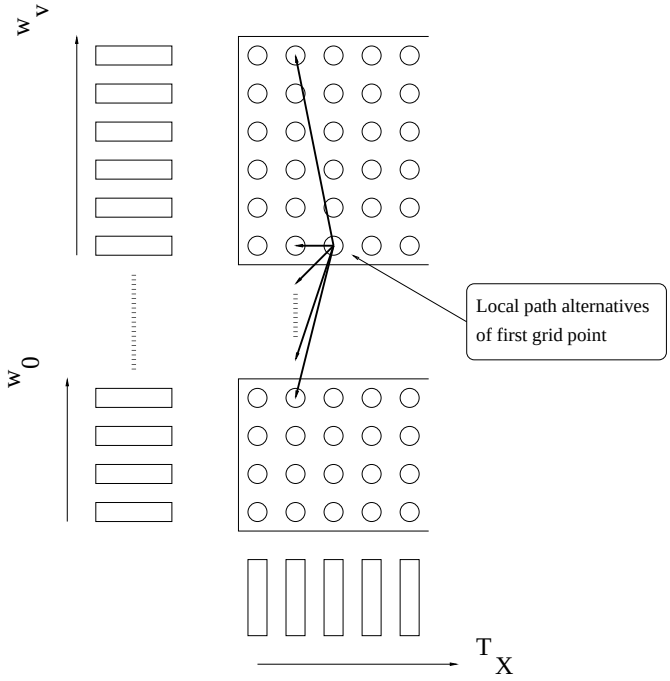

Figure 6.2: Local path alternatives of the first grid point of a column

By allowing these additional path alternatives, we allow the optimal path to run through several words in our search space, where a word boundary can be fond at those times at which the path jumps from the last grid point of a prototype column to the first grid point of a prototype column.
As in chapter 5, an additional "virtual grid point" is used to represent the selection of the best word end at the end of the utterance.
If we now use the DP algorithm again to find the optimal path through our search space, it will automatically match all words at all boundaries. If we then

use the backtracking to determine the optimal path, this path will represent the sequence of prototype words which best matches the given utterance !

6.3 Backtracking

To allow proper backtracking, we do not need to keep track of all the path alternatives selected within the prototype sequences, but we only need to keep track of the predecessor grid points selected during the between–word path recombinations. So for backtracking, we need backtracking memory to hold for each time index j the following values:

- For each time index j the best word end $w(j)$ at time j. Any word starting at time $(j+1)$ will select this best word end at time t as its predecessor.
- For each time index j the start time $t_s(j)$ of the word end $w(j)$, i.e., the time index at which the path ending at time index j in word $w(j)$ has entered the word.

Using these values, we can go backwards in time to find all the words and word boundaries defined by the optimal path.
Note that there is only one best word end for each time index j which all words starting at time index $(j+1)$ will select as a predecessor word. Therefore, it is also sufficient to keep track of the start time $t_s(j)$ for this best word end. The best word end at the end of the utterance (time index $(T_X - 1)$) is then used as the starting point for the backtracking procedure.
Figure 6.1 shows the search space, the virtual grid point, the modified path alternatives at the first grid points of the prototypes and a possible path through the search space, resulting in a corresponding sequence of words.

6.4 Formal Description

We now will deduce a formal description of the connected word recognition algorithm.

6.4.1 Dimensions of the Search Space

Let V be the size of our vocabulary. For simplicity, we will assume that each word w_v of our vocabulary is represented by a single prototype vector sequence $\tilde{W}_v$ only. The use of multiple prototypes per word can easily be implemented by mapping the sequence of prototypes found by the DP algorithm to the appropriate sequence of words.
Let T_X be the length of the unknown utterance $\tilde{X}$ and let T_v, $v = 0, 1, \ldots, (V-1)$ be the length of the prototype sequence of prototype $\tilde{W}_v$.
Using these definitions, our search space is spanned by the length of the utterance T_X in one dimension and the sum of the length of the prototype vector sequences in the other dimension. Instead of referring to a single time index for all the vectors of all the prototypes, we will use the prototype index v and the time index i within a given prototype separately:
Let v be the index of the prototype sequence, $v = 0, 1, \ldots (V-1)$. This will divide our search space into V separate "partitions" P_v, each representing the

grid points corresponding to a prototype $\tilde{W}_v$ of a word w_v. These partitions are visualized in Fig. 6.1. Let i be the time index within a given partition P_v, with $i = 0, 1, \ldots, (T_v - 1)$. Let j be the time index of the unknown utterance with $j = 0, 1, \ldots, (T_X - 1)$.
Given these definitions, a grid point in our search space can be identified by three indices, (i, v, j). We denote the local distance between a vector $\vec{x}_j$ of the utterance $\tilde{X}$ and a prototype vector $\vec{w}_{v,i}$ of prototype $\tilde{W}_v$ by $d(\vec{w}_{v,i}, \vec{x}_j)$ and the accumulated distance for grid point (i, v, j) we denote by $\delta(i, v, j)$.

6.4.2 Local Path Recombination

Within–Word Transition

For the grid points within a partition P_v, we use the same local path recombination as in (4.6):

$$\delta(i,v,j) = \min \left\{ \begin{array}{rcl} \delta(i,v,j-1) & + & d(\vec{w}_{v,i}, \vec{x}_j) \\ \delta(i-1,v,j-1) & + & 2 \cdot d(\vec{w}_{v,i}, \vec{x}_j) \\ \delta(i-1,v,j) & + & d(\vec{w}_{v,i}, \vec{x}_j) \end{array} \right. \tag{6.1}$$

This equation holds for all grid points *within* the partition except for the first grid point, i.e., for all grid points (i, v, j) with partition time index $i = 1, \ldots, (T_v - 1)$, partition index $v = 0, 1, \ldots (V-1)$ and utterance time index $j = 1, 2, \ldots, (T_X - 1)$. Please note that – as in section 4 – , there is no predecessor for the first time index $j = 0$ of the utterance, so we have to limit the path alternatives for the grid points $(i, v, 0)$ to their only predecessor grid point $(i - 1, v, 0)$, the "vertical" transition.

Between–Word Transitions

In addition, we have to define the local path alternatives for the between–word transitions, which are possible at the *first* grid point of the column of each partition, i.e., all grid points $(0, v, j)$. In this case, it is allowed to select either the "horizontal" transition from grid point $(0, v, j - 1)$ to grid point $(0, v, j)$, or to select any word end, i.e., the last grid point of any partition, $(T_v - 1, v, 1)$. From all these alternatives, the best predecessor is selected. To simplify the notation, the optimization over all partitions (word ends) is carried out as a separate step for a given time index $j-1$, resulting in $\gamma(j-1)$, the accumulated distance of the best word end hypothesis at time index $(j - 1)$:

$$\gamma(j-1) = \min_{v=0,1,\ldots,V-1} \{\delta(T_v - 1, v, 1) \tag{6.2}$$

This step recombines all path hypotheses containing a word end for the time index $(j - 1)$ (between–word path recombination). The optimization step for the first grid point $(0, v, j)$ of each partition can now be written as:

$$\delta(0,v,j) = \min \left\{ \begin{array}{rcl} \delta(0,v,j-1) & + & d(\vec{w}_{v,i}, \vec{x}_j) \\ \gamma(j-1) & + & d(\vec{w}_{v,i}, \vec{x}_j) \end{array} \right. \tag{6.3}$$

This optimization step will be processed for all partition indices $v = 0, 1, \ldots (V - 1)$ and utterance time index $j = 1, 2, \ldots, (T_X - 1)$. Again, for time index $j = 0$, there is no predecessor since the utterance starts at $j = 0$. Therefore, we need

to initialize all the grid points $(0, v, 0)$ (i.e., all the words starting at $j = 0$) with the local vector distances $d(\vec{w}_{v,0}, \vec{x}_0)$.

6.4.3 Termination

After we have processed all time indices $j = 0, 1, \ldots, (T_X - 1)$, we have for each partition P_v a hypothesis for a word end at time $j = (T_X - 1)$. These V path hypotheses are now recombined and – as for all other times j – the best accumulated distance is entered into $\gamma(T_X - 1)$. This step performs the final recombination of paths in the virtual grid point of Fig. 6.1, its value representing the accumulated distance for the globally optimal path.

6.4.4 Backtracking

During the backtracking operation, we have to run backwards in time from the end of the utterance to its beginning, and we have to follow all word boundaries along the optimal path. This results in a list of word indices, representing the sequence of words (in reversed order) contained in our vocabulary which best matches the spoken utterance.

Keeping Track of the Word Boundaries

To follow the path from the end of the utterance to its beginning, we need to track for each time index j which best word end was chosen during the optimization (6.2). We will hold this information in the array $w(j)$:

$$w(j) = \underset{v=0,1,\ldots,V-1}{\arg\min} \{\delta((T_v - 1), v, j) \tag{6.4}$$

For the best word end at every time index j, we also need to know the start time of that word. For the backtracking procedure, this enables us to run backwards in time through the array of best word ends:
Beginning with time index $j = (T_X - 1)$ (the last time index of the utterance), we look up the best word $w(j)$ and its start time $\hat{j} = t_s(j)$, which leads us to its predecessor word $w(\hat{j} - 1)$ which had to end at time index $(\hat{j} - 1)$. Using the start time of that word, we find its predecessor word again and so on.
So, for the best word end $w(j)$ at time index j we need to know at what time index the path ending in that word has entered the first grid point of the partition with index $v = w(j)$. We will hold this information in an array $t_s(j)$.
It is important to understand that this information has to be forwarded during the within–word path recombination (6.1) and also during the between–word path recombination (6.3): Whenever we are selecting a path alternative (k, v, l) as a predecessor for a given grid point (i, v, j), we not only have to add its accumulated distance, but we also have to keep track of the point in time when that selected path alternative entered the partition P_v.
Therefore, in addition to the accumulated distance $\delta(i, v, j)$ we need another variable $\xi(i, v, j)$ to hold the start time for each grid point (i, v, j).

Extensions for Backtracking

Now we are able to extend all the recombination steps by the bookkeeping needed for backtracking:

First we start with the last grid points $(T_v - 1, j - 1, v)$ of all partitions v: For these grid points, which represent the word ends, we have to perform the optimization over all possible word ends, i.e., we have to find the best word end at time $j - 1$ and store its its accumulated distance into $\gamma(j - 1)$, its partition index into $w(j - 1)$ and the corresponding word start time into $t_s(j - 1)$. Note that the word end optimization is done for the "old values", i.e., for time index $j - 1$, since the between–word path recombination will use $\gamma(j - 1)$ for finding the best predecessor word at time j.

- Find best word end:

$$v_{opt} = \underset{v=0,1,\ldots,(V-1)}{\arg\min} \{\delta(T_v - 1, v, j - 1)\}. \tag{6.5}$$

- Store accumulated distance of the best word end:

$$\gamma(j - 1) = \delta(T_{v_{opt}} - 1, v_{opt}, j - 1) \tag{6.6}$$

- Store the word index for backtracking:

$$w(j - 1) = v_{opt} \tag{6.7}$$

- Store word start time for backtracking:

$$t_s(j - 1) = \xi(T_{v_{opt}} - 1, v_{opt}, j - 1) \tag{6.8}$$

Now we will extend the within–word optimization step to keep track of the start times within the partitions:

- Within–word path recombination for partition v:

$$\delta(i, v, j) = \min \left\{ \begin{array}{rcl} \delta(i, v, j - 1) & + & d(\vec{w}_{v,i}, \vec{x}_j) \\ \delta(i - 1, v, j - 1) & + & 2 \cdot d(\vec{w}_{v,i}, \vec{x}_j) \\ \delta(i - 1, v, j) & + & d(\vec{w}_{v,i}, \vec{x}_j) \end{array} \right. \tag{6.9}$$

- Within–word backtracking variable $\xi(i, v, j)$ for partition v:
  - Get predecessor grid point (k, v, l):

$$(k, v, l) = \underset{k,v,l \in \left\{ \begin{array}{c} (i, v, j - 1), \\ (i - 1, v, j - 1), \\ (i - 1, v, j) \end{array} \right\}}{\arg\min} \left\{ \begin{array}{rcl} \delta(i, v, j - 1) & + & d(\vec{w}_{v,i}, \vec{x}_j) \\ \delta(i - 1, v, j - 1) & + & 2 \cdot d(\vec{w}_{v,i}, \vec{x}_j) \\ \delta(i - 1, v, j) & + & d(\vec{w}_{v,i}, \vec{x}_j) \end{array} \right. \tag{6.10}$$

  - Forward the start time of the predecessor:

$$\xi(i, v, j) = \xi(k, v, l) \tag{6.11}$$

For the first grid point $(0, v, j)$, there are two possible cases: Either the "horizontal" transition is selected, in which case we simply have to forward the start time $\xi(0, v, j - 1)$, which comes along with that path hypothesis, or the best word end at time $j - 1$ is selected. In that case, we have just (re-)started the word v at time j and we have to enter the start time j into $\xi(i, v, j)$. Now we are able to extend the between-word path recombination in analogy with (6.3):

- Path recombination at the first grid point:

$$\delta(0,v,j) = \min \left\{ \begin{array}{rcl} \delta(0,v,j-1) & + & d(\vec{w}_{v,i},\vec{x}_j) \\ \gamma(j-1) & + & d(\vec{w}_{v,i},\vec{x}_j) \end{array} \right. \tag{6.12}$$

- Forward start time in variable $\xi(0,v,j)$:

$$\xi(0,v,j) = \left\{ \begin{array}{rcl} \xi(0,v,j-1) & , & \text{if } \delta(0,v,j-1) \leq \gamma(j-1) \\ j & , & \text{else} \end{array} \right. \tag{6.13}$$

6.5 Implementation issues

As explained in section 4, the fact that all optimization steps need only the columns for the last time index helps us in saving computer memory. Only an array $\delta_j(i,v)$ is needed to hold the accumulated distances $\delta(i,v,j)$. The start time indices forwarded during recombination will also be held in an array $\xi_j(i,v)$. For backtracking, only the two arrays $w(j)$ and $t_s(j)$ are needed. So we only have to deal with a few arrays (and several "partition–local " variables) instead of explicitly holding in memory the information for all grid points (i,v,j) of the huge matrix spanned by our search space.
Of course, all the additional constraints of the search space as mentioned in section 4.2.3 can be applied here, too.
With the definitions of the equations for path recombination and backtracking given above, it is easy to write down the complete algorithm in analogy to the one given in section 4. This is left as an exercise to the reader.

6.6 Summary

What have we learned in this chapter?
We have extended the dynamic programming algorithm to deal with sequence of words instead of isolated words. This was achieved by reformulating the classification task in such a way that it could be described as finding the optimal path through a matrix of grid points together with the local schemes for within–word and between–word path recombination. Since the recognition process was reduced to finding the optimal path through the search space, we had to store the information needed for backtracking while processing the columns of the search space. We found that we only needed to store the information about the best word end optimization steps during the between–word path recombinations, which we can hold in two additional arrays.
Speech recognition systems based on the DP algorithm using prototype vector sequences to match against the unknown utterance (also known as *dynamic pattern matching*) were widely used some years ago, and were even extended by a finite state syntax (for a detailed description see e.g., [Ney84]).
Of course, there are many details to work out to implement a working speech recognition system based on the DP algorithm as described above. Some of the open issues not dealt with in the sections above are:

- How do we detect the start and the end of the utterance properly?
- How many prototype vector sequences $\tilde{W}_v$ should we use to represent a word of the vocabulary?

- How do we find appropriate prototypes for the words ?
- For a speaker–independent system the number of prototypes will have to be very large to cover all the variabilities involved with speaker–independent systems.
- The more prototypes we use, the larger our search space will be, consuming more memory and more computation time. How can we implement a speaker–independent system dealing with large vocabularies?

As we will see in the next chapters, there are better ways to model the prototype words for a speech recognition system.

Chapter 7

Hidden Markov Models

In the last chapter, we were using the DP algorithm to find the optimum match between an utterance and a sequence of words contained in our vocabulary. The words of the vocabulary were represented by one or more prototype vector sequences. All variability of speech that naturally occurs if a word is spoken at different times by a person or even by different persons had to be reflected by the prototype vector sequences. It should be clear that the number of prototypes we have to store in order to implement a speaker–independent system might become quite large, especially if we are dealing with a large vocabulary size also.
What we are rather looking for to handle this problem is a way to represent the words of our vocabulary in a more generic form than just to store many speech samples for each word. One would like to look for a model of the speech generation process for each word itself instead of storing samples of its output. If, for example, we would have a general stochastic model for the generation of feature vectors corresponding to a given word, then we could calculate how good a given utterance fits to our model. If we calculate a fit–value for each model of our vocabulary, then we can assign the unknown utterance to the model which best fits to the utterance.
This is a very simplistic description of a general classification method, the so–called *statistical classification*. For additional reading on statistical methods for speech recognition, the book [Jel98] is strongly recommended.
In the following, we will see how it can be used for speech recognition.

7.1 Statistical Classification

The classification task for isolated word recognition can be formulated in a statistical framework as follows:
Let's assume we have for each word w_v of our vocabulary a model Λ_v which is defined by a set of parameters λ_v. What kind of parameters these are we will learn later. For now, we assume that we are able to compute for a vector sequence $\tilde{X}$ the probability density that the utterance $\tilde{X}$ was produced by the model Λ_v. We denote this conditional probability density function, which depends on the model Λ_v, as $p(\tilde{X}|\Lambda_v)$. It will later often be called the *emission probability density*. Note that $p(\tilde{X}|\Lambda_v)$ is not a probability, but rather a class

specific *probability density function* (*PDF*), since we are dealing with *continuous* values of each element of the vectors $\vec{x}_j$ of the sequence, instead of discrete values.

7.1.1 Bayes Classification

The task of classifying isolated words can now be verbalized as: Assign the unknown utterance $\tilde{X}$ to that class ω_v, to which the unknown utterance "belongs" to with the highest probability.
What we need here is the so–called *a–posteriori probability* $P(\omega_v|\tilde{X})$. It denotes the probability that the utterance $\tilde{X}$ belongs to the class ω_v and will be computed with respect to all our models Λ for all classes ω.
The classification task at hand can now be defined as:

$$\tilde{X} \in \omega_v \Leftrightarrow v = \arg\max_v \left\{ P(\omega_v|\tilde{X}) \right\} \tag{7.1}$$

Note that in contrary to (4.2), we have to chose the *maximum a–posteriori probability* instead of the minimum class distance. The classification scheme as defined above is called the "Bayes Classification", e.g., [Rus88].
How can we compute the probability $P(\omega_v|\tilde{X})$ measuring the probability that $\tilde{X}$ belongs to class ω_v ? Using Bayes' equation it is computed as follows:

$$P(\omega_v|\tilde{X}) = \frac{p(\tilde{X}, \omega_v)}{p(\tilde{X})} = \frac{p(\tilde{X}|\omega_v) \cdot P(\omega_v)}{p(\tilde{X})} \tag{7.2}$$

If we insert (7.2) into (7.1), the unconditional emission probability density $p(\tilde{X})$ can be eliminated, since we are performing a maximum search and $p(\tilde{X})$ is independent from the class under consideration. Thus, we can rewrite (7.1):

$$\tilde{X} \in \omega_v \Leftrightarrow v = \arg\max_v \left\{ p(\tilde{X}|\omega_v) \cdot P(\omega_v) \right\} \tag{7.3}$$

So we only need the probability density functions $p(\tilde{X}|\omega_v)$ and the a–priori probabilities of the classes $P(\omega_v)$, which denote the unconditional (i.e., not depending on the measured vector sequence $\tilde{X}$) probability that the utterance belongs to class ω_v.
We can now use our word models Λ as approximations for the speech generation process and by using the emission probability densities of our word models, (7.3) can be rewritten as:

$$\tilde{X} \in \Lambda_v \Leftrightarrow v = \arg\max_v \left\{ p(\tilde{X}|\Lambda_v) \cdot P(\Lambda_v) \right\} \tag{7.4}$$

7.1.2 Maximum Likelihood Classification

Equation (7.4) uses the probability density functions $p(\tilde{X}|\Lambda_v)$ – which we assume we are able to compute – and the a–priori probabilities of the models $P(\Lambda_v)$, which could be determined by simply counting the word frequencies within a given training set.
However, the speech recognition task we are currently dealing with is the recognition of isolated words, like simple command words. It it reasonable to assume that these command words will occur with equally distributed probability, e.g.,

for a "yes / no" task, we can assume that the utterance "yes" will occur as often as "no". If we assume equally distributed a–priori probabilities, $P(\Lambda_v)$ becomes a constant value $(1/V)$ and we can rewrite (7.4) as:

$$\tilde{X} \in \Lambda_v \Leftrightarrow v = \arg\max_v \left\{ p(\tilde{X}|\Lambda_v) \right\} \tag{7.5}$$

Now the classification depends solely on the model (or class) specific probability densities $p(\tilde{X}|\Lambda_v)$, which are also called the *likelihood functions*. Therefore, this classification approach is called the *maximum likelihood classification*. In the next sections we will see how to compute the density functions $p(\tilde{X}|\Lambda_v)$.

7.2 Hidden Markov Models

One major breakthrough in speech recognition was the introduction of the statistical classification framework described above for speech recognition purposes. As we have seen, the important part of that framework is the emission probability density $p(\tilde{X}|\Lambda_v)$. These density functions have to be estimated for each word model Λ_v to reflect the generation (or "emission") process of all possible vector sequences $\tilde{X}_v$ belonging to the word class ω_v. As we know, these vector sequences my vary in their length as well as in the individual spectral shapes of the feature vectors within that sequence. Thus, a model is needed which is capable of dealing with both of these variabilities. The *Hidden Markov Models* (short: HMMs) are used successfully in speech recognition for many years.
In the following, we will only discuss those properties of the HMMs which we will need to understand how they are used in algorithms for speech recognition. A very detailed introduction into HMMs can be found in [RJ86, Rab89, Jel98], while early applications are described in [LRS85a, LRS89].

7.2.1 The Parameters of a Hidden Markov Model

The HMM is modelling a stochastic process defined by a set of *states* and *transition probabilities* between those states, where each state describes a stationary stochastic process and the transition from one state to another state describes how the process changes its characteristics in time.
Each state of the HMM can model the generation (or: *emission*) of the observed symbols (or in our case of the feature vectors) using a stationary stochastic emission process. For a given observation (vector) however, we do not know in which state the model has been when emitting that vector. The underlying stochastic process is therefore "hidden" from the observer.
In our application, each state of the HMM will model a certain segment of the vector sequence of the utterance. These segments (which are assumed to be stationary) are described by the stationary emission processes assigned to the states, while the dynamic changes of the vector sequence will be modelled by transitions between the states of the HMM.
Now it is time to introduce a little bit of formal framework:
Let the HMM Λ_v have N^v different states denoted as s_θ^v with $\theta \in \Omega_{N^v}$, where $\Omega_{N^v} = \{0, 1, \ldots, (N^v - 1)\}$ is the set of state indices.

Let $a^v_{i,j}$; $i,j \in \{0,1,\ldots,(N^v-1)\}$ be the transition probability between state s^v_i and s^v_j. The set of all transition probabilities can be compactly described by the matrix $\tilde{A}_{(N^v \times N^v)} = \left[a^v_{i,j}\right]$. Each state s^v_θ has assigned its stationary emission process defined by the emission probability density function $p\left(\vec{x}|s^v_\theta\right)$. In addition, we define the initial state probability u^v_θ ; $\theta \in \{0,1,\ldots,(N^v-1)\}$, which denotes the initial probability of the model being in state s^v_θ when the generation process starts.

7.2.2 HMMs for Word Modelling

A vector sequence $\tilde{X} = \left\{\vec{x}_0, \vec{x}_1, \ldots, \vec{x}_{(T_X-1)}\right\}$ which belongs to a word w_v can now be generated by the HMM Λ_v as follows (to simplify the notation, we skip the word class index v):
At each time $t = 0, 1, ...(T_X - 1)$ the HMM is in a state s_{θ_t} and will generate a vector $\vec{x}_t$ with the emission probability density $p\left(\vec{x}_t|s_{\theta_t}\right)$ (the initial state probability u_θ gives the probability being in state s_θ at time $t = 0$). Then it will make a state transition from state s_{θ_t} to state $s_{\theta_{t+1}}$ with the transition probability $a_{\theta_t,\theta_{t+1}}$ and so on. A given vector sequence $\tilde{X}$ can thus be generated by running through a sequence of state indices $\Theta = \left\{\theta_0, \theta 1, \ldots \theta_{(T_X-1)}\right\}$. However, given only the sequence $\tilde{X}$, one can not determine which sequence Θ was used to generate $\tilde{X}$ i.e., the state sequence is hidden from the observer.
The state transitions of the HMM reflect the sequence of quasi–stationary segments of speech contained in the utterance $\tilde{X}$. Therefore, one usually restricts the transition probability densities of the HMM to a "from left to right" structure, i.e., being in a certain state s_θ, only states with the same or a higher state index θ can be reached with nonzero probability. The initial state probability is usually set to one for the first state and to zero for all the other states, so the generation process always starts in state s_0. Usually the state transition probabilities are restricted to reach only the two next state indices (e.g., [LRS85a]):

$$\begin{array}{lll} a_{\theta,\theta+\Delta} > 0 & , & \text{if } 0 \leq \Delta \leq 2 \\ a_{\theta,\theta+\Delta} = 0 & , & \text{else} \end{array} \tag{7.6}$$

This leads to a left–to–right HMM structure as shown for five states in Fig. 7.1

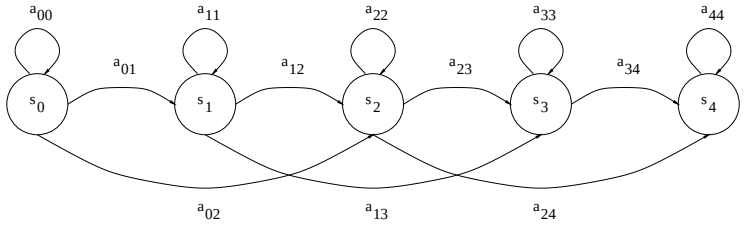

Figure 7.1: Typical left–to–right HMM for word recognition

Note that by choosing nonzero self–transitions probabilities $a_{\theta,\theta}$, vector sequences of infinite length can generated, the shortest length of the sequence is defined by the shortest path through the model, e.g., in Fig. 7.1, the shortest vector sequence must have three vectors.

7.2.3 Emission Probability Density of a Vector Sequence

As we saw in section 7.1.2, we have to compute the emission probability density $p\left(\tilde{X}|\Lambda\right)$ for a given sequence $\tilde{X}$ and model Λ. We remember that the HMM will generate the vector sequence $\tilde{X}$ along a state index sequence $\Theta = \left\{\theta_0, \theta 1, \ldots \theta_{(T_X-1)}\right\}$. Assuming that we know the state sequence Θ which produces $\tilde{X}$, we can easily compute the probability density $p\left(\tilde{X}, \Theta|\Lambda\right)$, which is the probability density of generating $\tilde{X}$ along the state sequence Θ using the model Λ. This is done by simply multiplying all the emission probability densities and transition probabilities along the state sequence Θ:

$$p\left(\tilde{X}, \Theta|\Lambda\right) = u_{\theta_0} \cdot p\left(\vec{x}_0|s_{\theta_0}\right) \cdot \prod_{t=1}^{T_X-1} \left(a_{\theta_{(t-1)},\theta_t} \cdot p\left(\vec{x}_t|s_{\theta_t}\right)\right) \tag{7.7}$$

To compute the desired emission probability density $p\left(\tilde{X}|\Lambda\right)$, we now have to sum up $p\left(\tilde{X}, \Theta|\Lambda\right)$ over all possible state sequences Θ with length T_X. The number of possible state sequences equals the number of T_X – tupels that can be generated from the set of state indices Ω_N. We denote the set of all state sequences of length T_X given the set of state indices Ω_N as $\Omega_{\left(N^{(T_X)}\right)}$.
Then, $p\left(\tilde{X}|\Lambda\right)$ can be computed as:

$$p\left(\tilde{X}|\Lambda\right) = \sum_{\Theta \in \Omega_{\left(N^{(T_X)}\right)}} p\left(\tilde{X}, \Theta|\Lambda\right) \tag{7.8}$$

If we insert (7.7) into (7.8), we get:

$$p\left(\tilde{X}|\Lambda\right) = \sum_{\Theta \in \Omega_{\left(N^{(T_X)}\right)}} \left(u_{\theta_0} \cdot p\left(\vec{x}_0|s_{\theta_0}\right) \cdot \prod_{t=1}^{T_X-1} \left(a_{\theta_{(t-1)},\theta_t} \cdot p\left(\vec{x}_t|s_{\theta_t}\right)\right)\right) \tag{7.9}$$

To compute the sum over all possible state sequences would require lots of computational efforts, but fortunately there exists an efficient algorithm to do so, the so–called *Baum–Welch* algorithm [Rab89].
Instead of using the Baum–Welch algorithm, we choose a different approach: We will not compute the sum over all state sequences $\Theta \in \Omega_{\left(N^{(T_X)}\right)}$, but we will approximate the sum by the single state sequence Θ_{opt} which has the highest contribution to the sum (7.9):

$$p\left(\tilde{X}|\Lambda\right) \approx p\left(\tilde{X}, \Theta_{opt}|\Lambda\right) = u_{\theta_0} \cdot p\left(\vec{x}_0|s_{\theta_0}\right) \cdot \prod_{t=1}^{T_X-1} \left(a_{\theta_{(t-1)},\theta_t} \cdot p\left(\vec{x}_t|s_{\theta_t}\right)\right) \tag{7.10}$$

Where Θ_{opt} is defined as the state sequence which maximizes the emission probability density $p\left(\tilde{X}, \Theta|\Lambda\right)$:

$$\Theta_{opt} = \underset{\Theta \in \Omega_{\left(N^{(T_X)}\right)}}{\arg\max} \; p\left(\tilde{X}, \Theta|\Lambda\right) \tag{7.11}$$

If we evaluate (7.10), we have to deal with product terms only, so that we can easily take the logarithm of the individual probability (density) values and replace the product by the sum of those log–values.
Since the probability density values may vary over several orders of magnitude, taking the logarithm has the additional advantage of reducing the dynamic range of those values. The logarithm of the probability density is often called the *log score value*.
How do we find the optimal state sequence Θ_{opt} ?
Fig. 7.2 shows all the possible state sequences or paths (also called the trellis) for the HMM shown in Fig. 7.1. In this Figure, the HMM is rotated by 90 degrees and on the horizontal axis the vectors of the utterance $\tilde{X}$ are shown. At every time index t, each possible path will go through one of the N states of the HMM. In other words, at every grid point shown in Fig. 7.2, a path recombination takes place. If we are using the log score values for the transition probabilities and the emission probability densities, the evaluation of (7.10) is transformed to adding up all the log score values along a given path, in analogy to adding all the local distances along the path in section 4.

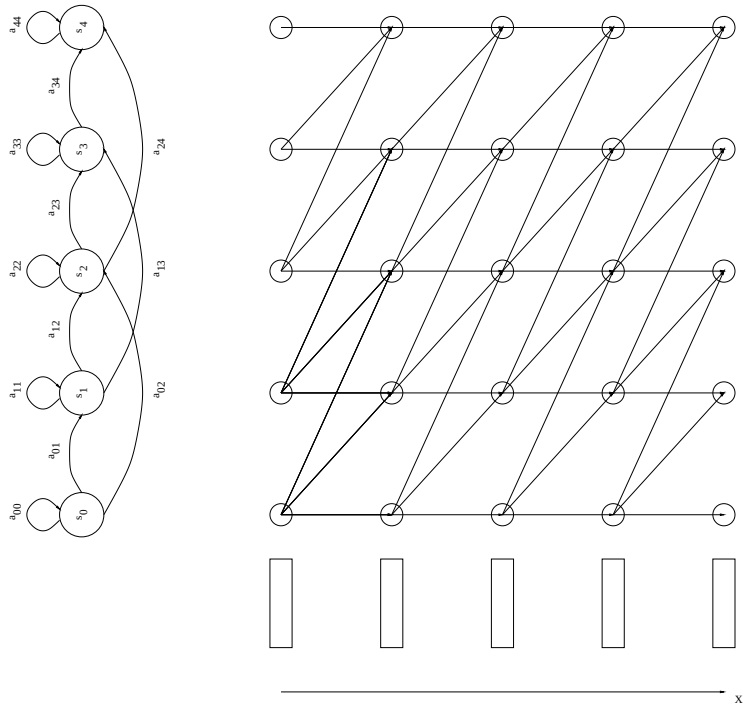

Figure 7.2: Possible state sequences through the HMM

Now it is easy to find the optimum state sequence / the optimum path through the grid: It can be done with the DP algorithm (Also known as *Viterbi algorithm*) we learned in section 4 ! All we have to do is to define the equation for the local recombination.

7.2.4 Computing the Emission Probability Density

To compute $p\left(\tilde{X}, \Theta_{opt}|\Lambda\right)$, we have to find the path or state sequence which yields the *maximum* accumulated sum of log scores along that path. In section 4, we were searching for the *minimum* accumulated distance, instead. If we would use the *negative* log score values, we could even search for the path with the minimum accumulated negative log score. However, it is more intuitive to deal with higher scores for better matches, so we will use log scores and search for the maximum in the following.

Scores and Local Path Recombination

We will start with the definition of the log score values. Since we are dealing with one given HMM Λ_v here, we will leave out the model index v in the following. Let $b(t, s_\theta)$ denote the logarithm of the emission probability density of vector $\vec{x}_t$ being emitted by state s_θ:

$$b(t, s_\theta) = \log\left(p\left(\vec{x}_t|s_\theta\right)\right) \tag{7.12}$$

Let $d(s_i, s_\theta)$ be the log probability for the state transition $s_i \rightarrow s_\theta$:

$$d(s_i, s_\theta) = \log\left(a_{s_i, s_\theta}\right) \tag{7.13}$$

Let $\delta(s_\theta, t)$ be the accumulated log probability density for the optimal path beginning at time index 0 and ending in state s_θ at time index t (If we further assume that $u_0 = 1$, then all paths are starting at time index 0 in state s_0).
Now we will define the local path alternatives for path recombination. Depending on the transition probabilities of the model, a given state may have up to N predecessors. In our left–to–right structure, the states s_θ in the middle of the model have only three possible predecessors (see Fig. 7.3): The state s_θ itself, and its two predecessors with the lower state indices $(\theta - 1)$ and $(\theta - 2)$. However, this is not true for the first and second state of the model. To allow for more complex model structures also, we define the set $\Xi\left(s_\theta\right)$ as the set of all valid predecessor state indices for a given state s_θ.
With these definitions, the local path recombination scheme can be written as:

$$\delta(s_\theta, t) = b(t, s_\theta) + \max_{i \in \Xi(s_\theta)} \{d(s_i, s_\theta) + \delta(s_i, (t-1))\} \tag{7.14}$$

Note that in contrary to (4.6), all predecessor grid points have the time index $(t-1)$ here (there is no vertical transition possible, since each state transition in the HMM implies an increase of the time index t). This local path recombination will now be performed for each time index t and within each time index for all state indices θ.

The Viterbi Algorithm

Now, we can formulate the Viterbi (or DP) algorithm for computing the score $\log\left(p\left(\tilde{X}, \Theta_{opt}|\Lambda\right)\right)$. To enable the backtracking of the state sequence Θ_{opt}, we will use the backtracking variable $\psi(\theta, t)$ in analogy to section 4:

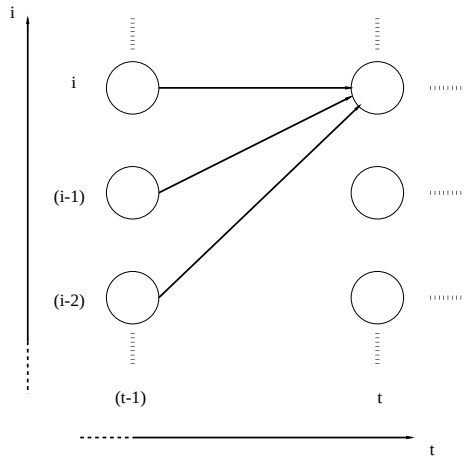

Figure 7.3: Possible predecessor states for a given HMM state

- Initialization ($t = 0$):
  For the left–to–right structure we use, we define $u_0 = 1$. Therefore, all other initial state probabilities are zero, enforcing all paths to begin in state s_0 with the emission of the vector $\vec{x}_0$:

  - Initialize first column:

$$\delta(s_\theta, 0) = \begin{cases} b(0, s_0) & , \quad \theta = 0 \\ -\infty & , \quad \theta = 1, 2, \ldots, (N-1) \end{cases} \tag{7.15}$$

  - Initialize backtracking variables:

$$\psi(\theta, 0) = -1 \; ; \; \theta = 0, 1, \ldots, (N-1) \tag{7.16}$$

- Iteration:
  Perform the local path recombination as given by (7.14) for all states $\theta = 0, 1, \ldots, (N-1)$ and all time indices $t = 1, 2, \ldots, (T_X - 1)$:

  - Optimization step:

$$\delta(s_\theta, t) = b(t, s_\theta) + \max_{i \, \in \, \Xi(s_\theta)} \{d(s_i, s_\theta) + \delta(s_i, t-1)\} \tag{7.17}$$

  - Backtracking:

$$\psi(\theta, t) = \underset{i \, \in \, \Xi(s_\theta)}{\arg\max} \{d(s_i, s_\theta) + \delta(s_i, t-1)\} \tag{7.18}$$

- Termination:
  In our left–to–right model, we will consider only paths which end in the last state of the HMM, the state s_{N-1}. After the last iteration step is done, we get:

$$\log\left(p\left(\tilde{X}, \Theta_{opt} | \Lambda\right)\right) = \delta(s_{N-1}, (T_X - 1)) \tag{7.19}$$

- Backtracking procedure:
  Beginning with time index $(T_X - 1)$, go backwards in time through the backtracking variables:
  - Initialization:
    $$t = (T_X - 1)\ ;\ \theta_{(T_X-1)} = N - 1 \tag{7.20}$$
  - Iteration:
    For $t = (T_X - 1), \ldots, 1$:
    $$\theta_{(t-1)} = \psi(\theta_t, t) \tag{7.21}$$
  - Termination:
    $$\theta_0 = \psi(\theta_1, 1) \equiv 0 \tag{7.22}$$

7.3 Modelling the Emission Probability Densities

So far, we have assumed that we can compute the value of the state–specific emission probability density $p(\vec{x}|s_\theta)$ of a HMM Λ. Now we have to deal with the question of how $p(\vec{x}|s_\theta)$ is modelled within each state of a HMM and how to compute the probability density for a given vector $\vec{x}$. As we recall, in the states of the HMM we model stationary emission processes which we assume to correspond with stationary segments of speech. Within those segments, we have to allow for the wide variability of the emitted vectors caused by the variability of the characteristic features of the speech signal.

7.3.1 Continuous Mixture Densities

How do we model the emission process ?
As we remember from chapter 2, we are dealing with feature vectors where each component is representing a continuous valued feature of the speech signal. Before we deal with the question of how to model probability density functions for multi–dimensional feature vectors, we will first look at the one–dimensional case.
Lets assume we measure only one continuously valued feature x. If we assume the measurement of this value to be disturbed by many statistically independent processes, we can assume that our measurements will assume a Gaussian distribution of values, centered around a mean value m, which we then will assume to be a good estimate for the true value of x. The values we measure can be characterized by the Gaussian probability density function. As we recall from school, the Gaussian PDF $\phi(x)$ is defined as:

$$\phi(x) = \frac{1}{\sqrt{2\pi\sigma^2}} \cdot e^{\left(-\frac{1}{2}\cdot\frac{(x-m)^2}{\sigma^2}\right)} \tag{7.23}$$

Where m represents the mean value and σ^2 denotes the variance of the PDF. These two parameters fully characterize the one–dimensional Gaussian PDF.

Gaussian Probability Density Function

In our application, we measure feature vectors in a multi–dimensional feature space, so we will use a *multivariate* Gaussian PDF, which looks like this:

$$\phi(\vec{x}) = \frac{1}{\sqrt{(2\pi)^{DIM} \cdot |\tilde{C}|}} \cdot e^{\left(-\frac{1}{2}(\vec{x}-\vec{m})' \cdot \tilde{C}^{-1} \cdot (\vec{x}-\vec{m})\right)} \tag{7.24}$$

Where $\vec{m}$ denotes the mean vector and $\tilde{C}$ denotes the covariance matrix. You know these parameters already, because we used them in computing the Mahalanobis distance (3.8) in section 3.4. Note that the the Mahalanobis distance is actually part of the exponent of the PDF (7.24).

Continuous Mixture Densities

The Gaussian PDF (7.24) can characterize the observation probability for vectors generated by a single Gaussian process. The PDF of this process has a maximum value at the position of the mean vector $\vec{m}$ and its value exponentially decreases with increasing distance from the mean vector. The regions of constant probability density are of elliptical shape, and their orientation is determined by the Eigenvectors of the covariance matrix (e.g., [Rus88]). However, for speech recognition, we would like to model more complex probability distributions, which have more than one maximum and whose regions of constant probability density are not elliptically shaped, but have complex shapes like the regions we saw in Fig. 2.1.
To do so, the weighted sum over a set of K Gaussian densities can be used to model $p(\vec{x})$ (e.g., [CHLP92, LRS85b, Rab89]):

$$p(\vec{x}) = \sum_{k=0}^{K-1} c_k \cdot \phi_k(\vec{x}) \tag{7.25}$$

Where $\phi_k(\vec{x})$ is the Gaussian PDF as in (7.24). The weighting coefficients c_k are called the *mixture coefficients* and have to fit the constraint:

$$\sum_{k=0}^{K-1} c_k = 1 \tag{7.26}$$

A PDF modelled by this weighted sum of Gaussian densities is called a *Continuous Mixture Density*, a HMM using this modelling approach is called a *Continuous Mixture Density HMM*.
Figure 7.4 shows the scatterplot of an emission process consisting of three Gaussian emission processes. The process parameters are:

$$\begin{aligned}
c_1 = 0.3 \;\; ; \;\; \tilde{C}_1 &= \begin{bmatrix} 5 & 2 \\ 2 & 2 \end{bmatrix} \;\; ; \;\; \vec{m}_1 = [1, 1.5]' \\
c_2 = 0.4 \;\; ; \;\; \tilde{C}_2 &= \begin{bmatrix} 1 & -1 \\ -1 & 5 \end{bmatrix} \;\; ; \;\; \vec{m}_2 = [-3, -1]' \\
c_2 = 0.3 \;\; ; \;\; \tilde{C}_3 &= \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \;\; ; \;\; \vec{m}_3 = [3, -3]'
\end{aligned}$$

The corresponding probability density function is visualized as a suface in fig. 7.5.

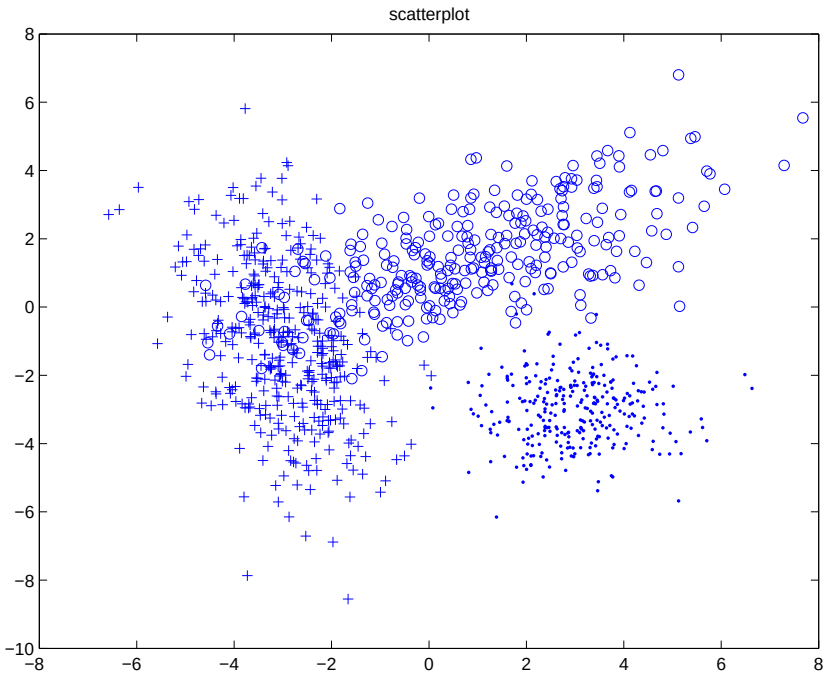

Figure 7.4: scatterplot of vectors emitted by a gaussian mixture density process

The complexity of the PDF modelled by (7.25) can be influenced by selecting the appropriate number of mixtures K and may be set individually for every state of the HMM. However, since a high number of mixtures also means that a high number of parameters has to be estimated from a given training set of data, always some compromise between the granularity of the modeling and the reliability of the estimation of the parameters has to be found.

7.3.2 Approximations

Using mixture densities leads to some practical problems:
First, the computational effort to compute all the densities for all states of a HMM might be very high.
We left out the state indices in the sections before for better readability, but we have to remember that in every state s_θ of the HMM a state–specific emission probability density function $p(\vec{x}|s_\theta)$ has to be modelled, each having K_θ gaussian densities. Since the computation of each Gaussian density involves the multiplication with the covariance matrix, this leads to a significant computational complexity.

Second, the high number of parameters demands a huge training material to reliably estimate the model parameters.

Many methods are known to reduce the computational complexity and the number of parameters as well. In the following, just some examples of the various

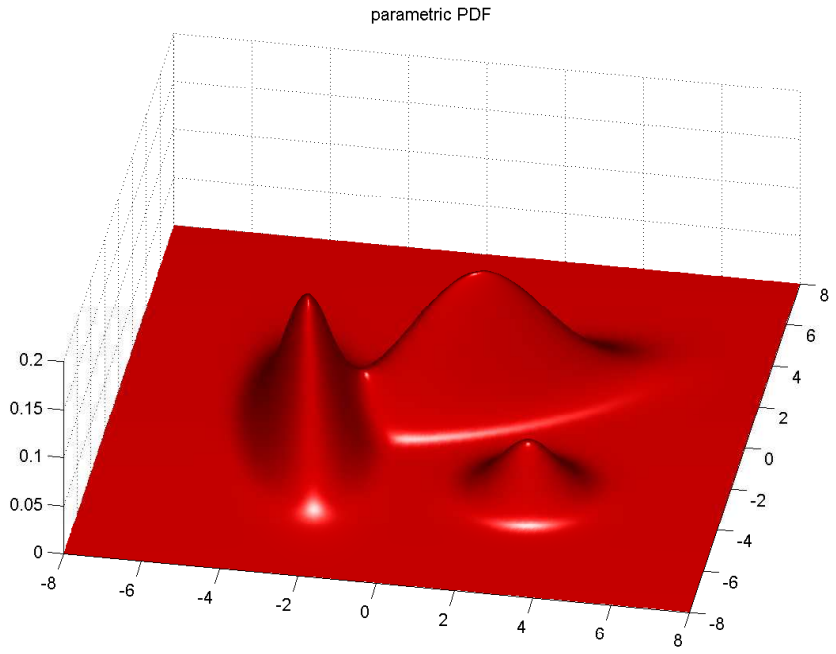

Figure 7.5: probability density function

possibilities are given.

Maximum Approximation

In (7.25), the sum of all K gaussian mixtures is computed. However, since value for a gaussian density exponentially decreases with the distance from the mean vector, one can approximate the sum over all mixtures by selecting just the one term which yields the highest contribution to the sum (e.g.,[Ney93]):

$$p(\vec{x}) \approx \max_{k=0,1,\ldots,K-1} \{c_k \cdot \phi_k(\vec{x})\} \tag{7.27}$$

Since we do not have to compute the sum over the mixture densities anymore, the log values of the mixture coefficient and the gaussian PDF can be used, which we know helps us in avoiding numerical problems:

$$\log\left(p(\vec{x})\right) \approx \max_{k=0,1,\ldots,K-1} \{\log(c_k) + \log\left(\phi_k(\vec{x})\right)\} \tag{7.28}$$

Diagonal Covariance Matrix

To reduce the number of parameters and the computational effort simultaneously, it is often assumed that the individual features of the feature vector are not correlated. Then diagonal covariance matrices can be used instead of full covariance matrices. This drastically reduces the number of parameters and the computational effort. As a consequence, a higher number of mixtures can be

used, and correlation of the feature vectors can thus (to a certain degree) be modelled by the combination of diagonal mixtures [LRS85b, CHLP92].

Parameter Tying

To reduce the number of parameters even further, some parameters can be "shared" or *tied* among several mixtures of a given state, among several states of an HMM and even among the HMMs. The idea behind parameter tying is that if two mixtures (or states or models) have tied parameters, i.e., are using the same value for that parameter, then the training material for these mixtures (states, models) can also be pooled to increase the reliability of the estimation of the tied parameter. A huge number of approaches for Models with tied parameters exists and is beyond the scope of this introduction. Just a few examples shall be given here:

- Using a single covariance matrix:
  A commonly used approach is to share the covariance matrix among all states and even among all models, while the mean vectors and the mixture coefficients are estimated individually for each state. These parameters allow a reliable estimate and therefore the number of mixtures per state can be significantly increased, thus implicitly modelling state–specific variances and correlations (e.g., [Ney93]).

- Semicontinuous HMM:
  Instead of estimating individual gaussian densities $\phi_{k,\theta,\Lambda}(\vec{x})$ for each state θ and Model Λ, The densities are shared among all models and states. Only the mixture coefficients $c_{k,\theta,\Lambda}$ are estimated individually for each state (e.g.,[XDHH90, HL92]).

- Generalised Parameter Tying:
  While the examples above were defining the set of tied parameters by an a–priori decision, it is also possible to introduce parameter tying on several levels (e.g., tied mean vectors, tied covariances, tied gaussian densities, tied mixtures, tied states, tied models), depending on the amount of training material available as well as determined on incorporated a-priori knowledge. Examples can be found in [You92, SYW94, You96, You01].

Chapter 8

Speech Recognition with HMMs

Now that we are able to compute the likelihood functions $p(\tilde{X}|\Lambda_v)$, we can perform the maximum likelihood classification (7.5) as described in section 7.1.2. In analogy to chapter 5, we will use the optimum path sequence found by the Viterbi algorithm to perform the maximum likelihood classification.

8.1 Isolated Word Recognition

First of all, we will organize our search space according to the task at hand. To do so, we will build a super-HMM, which consists of all word models Λ_v and is extended by two virtual states, a virtual start state and a virtual end state, respectively. In the following, the state index will be extended by the additional index v to identify the model Λ_v a state $s_{(\theta,v)}$ belongs to. Fig. 8.1 shows the structure of the super-HMM.

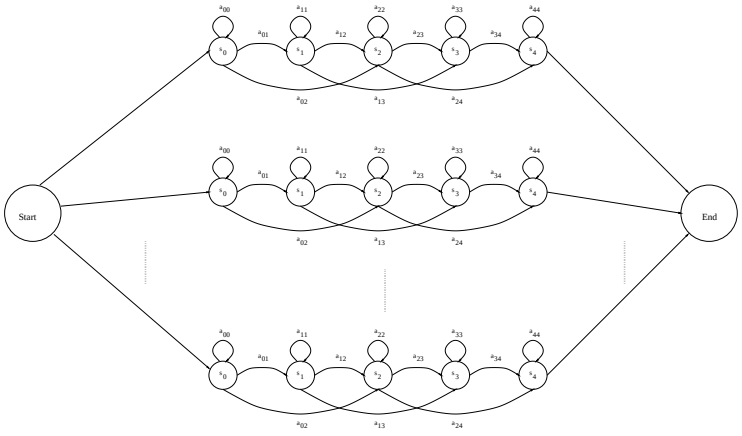

Figure 8.1: The model structure for isolated word recognition

Now we adapt the Viterbi algorithm we used in section 7.2.4 accordingly:

The virtual start state will not emit any vectors but will only provide the first states of each HMM with a predecessor for the virtual time slot $t = -1$. To initialize the Viterbi algorithm, we simply set the accumulated score for s_{start} to zero and the backtracking variable $\psi(s_{start}, -1)$ to -1. For each first state $s_{(\theta,v)}$ of each HMM Λ_v, the set of predecessor states $\Xi\left(s_{(\theta,v)}\right)$ is defined to contain only the state s_{start}. For simplicity, the state transition scores $d(s_{start}, s_{(\theta,v)})$ are set to zero instead of $log(\frac{1}{V})$.
For the end state of the super HMM, the set of predecessors $\Xi\left(s_{end}\right)$ will contain all end states $s_{((N_v-1),v)}$. The transition scores $d(s_{((N_v-1),v)}, s_{end})$ are set to zero.
Now, the Viterbi iteration (7.17) and backtracking procedure (7.18) is performed for all states $s_{(\theta,v)}$ for all HMMs Λ_v and for all time indices $t = 0, 1, \ldots, (T_X - 1)$. Due to the definition of the variables for our start state as given above, it is ensured that the first state of each model will be initialized properly for time $t = 0$. If backtracking is desired, now both the state and the model index (i, v) have to be stored in $\psi(\theta, t)$.
To terminate the Viterbi algorithm, we perform the last path recombination for the virtual time index T_X:

- Optimization step:

$$\delta(s_{end}, T_X) = \max_{(i,v) \in \Xi(s_{end})} \left\{\delta(s_{(i,v)}, (T_X - 1))\right\} \tag{8.1}$$

- Backtracking:

$$\psi(s_{end}, T_X) = \arg\max_{(i,v) \in \Xi(s_{end})} \left\{\delta(s_{(i,v)}, (T_X - 1))\right\} \tag{8.2}$$

The word index v contained in $\psi(s_{end}, T_X)$ contains the best path among all possible word ends to s_{end} and is therefore the desired classification result according to the maximum likelihood approach.

By constructing the Super HMM framework we were able to solve the classification task by finding the optimal state sequence, as we did in in chapter 5. Of course, as in that chapter, the Viterbi algorithm allows a real–time implementation.

To construct large HMMs which reflect the speech recognition task at hand from smaller HMMs and performing the recognition task at hand by searching the optimum state sequence for that HMM is one of the most important issues in the stochastic modelling framework. These super models can even contain additional knowledge sources, as e.g., a so–called *language model* which models the a–priori probabilities of sequences of words. Because with these large HMMs those knowledge sources can be fully integrated into the framework (even into the training procedures), they are also called *unified stochastic engines*, e.g.[XHH93].
We will see in the following section, that the extension of that framework to connected word recognition is straightforward.

8.2 Connected Word Recognition

8.2.1 Definition of the Search Space

If we want to perform connected word recognition as in chapter 6, we will have to construct a super HMM capable of generating an unlimited *sequence* of words from our vocabulary. This is an easy thing to do: we simply take our model for isolated word recognition and introduce a "loop" transition from the last state back to the first state of the model. As before, those two virtual states will not emit any vectors and therefore will not consume any time index. They are just used to simplify the the "bookkeeping" for the path recombinations after leaving the last states of the individual word models and the transition into the first states of the word models. The resulting model is shown in Fig. 8.2.

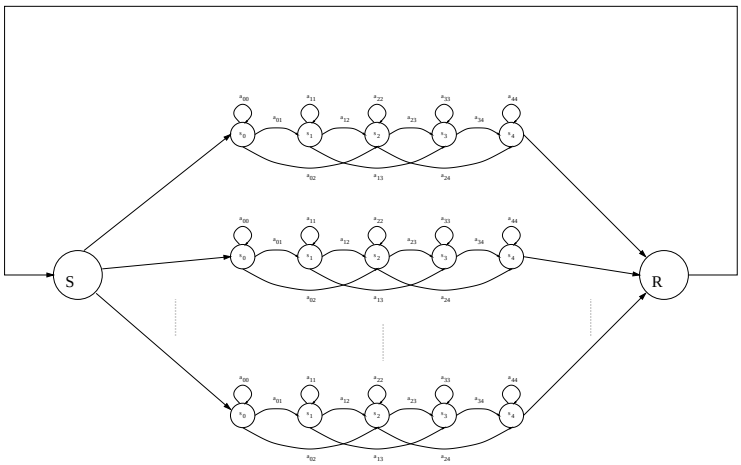

Figure 8.2: The model structure for connected word recognition

Note that this extension is in total analogy to what we did in chapter 6:

At each time index t all word models may make a transition from their last state to the virtual "R" state for path recombination. After the path recombination a virtual transition to state "S" is made and state "S" may be used as a possible predecessor state for the first states of all word models.

In chapter 6, for between–word path recombination, the first grid point of each word was allowed to select either the the "horizontal" transition or the best word end as its predecessor grid end. By introducing the variable $\gamma(j-1)$ as in (6.2) to simplify the between–word path recombination (6.3), we did in fact introduce a virtual state like S_R for path recombination.

To define the dimensions of our search space is very simple: It is spanned by all states of all word models Λ_v in one dimension and by the vectors of the vector sequence $\tilde{X}$ in the other dimension. To identify the individual states s_θ of the word models, we use again the model index v as an additional index, in analogy to the use of vector indices i and partition indices v we had in chapter 6.

8.2.2 The Viterbi Algorithm for Connected Word Recognition

Now, all we have to to is to reformulate the equations for path recombination and backtracking given in section 7.2.4 to fit to the new problem.
Let $\Xi\left(s_{(\theta,v)}\right)$ be the set of predecessor states of state $s_{(\theta,v)}$ of word model Λ_v. Note that the predecessor states are identified by the tupel (i,k) consisting of the state index i and the model index k. We extend the set of predecessors Ξ to consider also the two states s_R and s_S: For s_R, $\Xi(s_R)$ will contain the state indices of all last states $s_{(N_v-1),v}$ of all models Λ_v.
The state s_S has only the predecessor s_R and will not emit any vector. Since it is used only as a predecessor for the first states of the word HMMs, we can define S_R as their predecessor state directly, in fact merging the states s_S and s_R into s_R: For all first states of all models Λ_v, $\Xi(s_{0,v})$ will contain only the the state index of s_R and the state index for the state $s_{0,v}$ itself to allow the self–transition .
We start with the local path recombination in analogy to (7.14):

$$\delta(s_{(\theta,v)},t) = b(t,s_{(\theta,v)}) + \max_{(i,k)\,\in\,\Xi\left(s_{(\theta,v)}\right)} \left\{d(s_{(i,k)},s_{\theta,v}) + \delta(s_{i,k)},(t-1))\right\} \quad (8.3)$$

The equation for the local path recombination (8.3) holds also for the first states of each model due to the definition of the set of predecessors.

For the virtual state s_R, the recombination of the paths at the word ends is performed for the time index t. This has to be done *after* all the local path recombinations to ensure that all local paths leading to word ends are already recombined correctly before we are selecting the best word end. To emphasize this fact in section 6.4.4, we explicitly made this step at the beginning of time index j using the index of the old values $j-1$. Now that we are used to it, we just keep in mind that the recombination for the state s_R has to be tha last action for time index t:

$$\delta(s_R,t) = \max_{(i,k)\,\in\,\Xi(s_R)} \left\{\delta(s_{i,k)},t)\right\} \quad (8.4)$$

To allow backtracking of the word transitions made during the between–word path recombination (8.4), we need the backtracking arrays $w(t)$ and $t_s(t)$ as in section 6.4.4 and again we must forward the start time of the word model during the local path recombination.
Now, the Viterbi algorithm for connected word recognition with HMMs can be defined as follows:

- Initialization:
  For the first time index $t=0$, we must initialize the first columns for all models. While the first states of all models will be initialized with the log score of emitting the first vector of the utterance, all other states will be initialized with the default value $-\infty$.
  The word start time for all models at that time index is $t=0$, of course.

  - Initialize first columns:
    For all $v = 0,1,\ldots,(V-1)$:

$$\delta(s_{(\theta,v)},0) = \begin{cases} b(0,s_{(0,v)}) & , \quad \theta = 0 \\ -\infty & , \quad \theta = 1,2,\ldots,(N_v-1) \end{cases} \tag{8.5}$$

– Initialize state–level backtracking variables:
For all $v = 0,1,\ldots,(V-1)$:

$$\xi((\theta,v),0) = \begin{cases} 0 & , \quad \theta = 0 \\ -1 & , \quad \theta = 1,2,\ldots,(N_v-1) \end{cases} \tag{8.6}$$

– Init word level backtracking:

$$t_s(0) = 0\ ;\ w(0) = -1 \tag{8.7}$$

• Iteration for $t = 1,2,\ldots,(T_X-1)$:

– Local path recombination:

$$\delta(s_{(\theta,v)},t) = b(t,s_{(\theta,v)}) + \max_{(i,k)\ \in\ \Xi\left(s_{(\theta,v)}\right)} \left\{ d(s_{(i,k)},s_{\theta,v}) + \delta(s_{(i,k)},(t-1)) \right\} \tag{8.8}$$

– Backtracking at state level:
Forward the word start time associated with the path ending in $\delta(s_{(\theta,v)},t)$:

∗ Get predecessor grid point:

$$(i,k) = \operatorname*{arg\,max}_{(i,k)\ \in\ \Xi\left(s_{(\theta,v)}\right)} \left\{ d(s_{(i,k)},s_{\theta,v}) + \delta(s_{(i,k)},(t-1)) \right\} \tag{8.9}$$

∗ Forward the start time:

$$\xi((\theta,v),t) = \xi((i,k),t-1) \tag{8.10}$$

– Path recombination at the word ends:
Select the best word end at time t and enter the score into $\delta(s_R,t)$.

$$\delta(s_R,t) = \max_{(\theta,v)\ \in\ \Xi(s_R)} \left\{ \delta(s_{((\theta,v)},t) \right\} \tag{8.11}$$

– Backtracking at word level:
Store the word index and word start time of the best word end at time t:

∗ Get best word end state (i,k) at time t:

$$(i,k) = \operatorname*{arg\,max}_{(\theta,v)\ \in\ \Xi(s_R)} \left\{ \delta(s_{((\theta,v)},t) \right\} \tag{8.12}$$

∗ Store word index:

$$w(t) = k \tag{8.13}$$

∗ Store word start time:

$$t_s(t) = \xi((i,k),t) \tag{8.14}$$

- Termination:
  At $t = (T_X - 1)$, the accumulated score of the utterance can be found in $\delta(s_R, (T_X - 1))$, the index of the best word end (the last word of the utterance) can be found in $w(T_X - 1)$ and the start time of that word in $t_s(T_X - 1)$

- Backtracking Procedure:
  Beginning with time index $(T_X - 1)$, go backwards in time through the backtracking variables:

  - Initialization:

$$t = (T_X - 1) \tag{8.15}$$

  - Iteration:
    While $t > 0$:

$$w_t = w(t) \ ; \ t = t_s(t); \tag{8.16}$$

8.3 Summary

Now you know the principles of the algorithms used for simple speech recognition tasks like the recognition of connected digits. Of course, real implementations will deal with additional problems like the reduction of the computational efforts during the Viterbi search or the incorporation of silence models into the search process.

We know now how we can use the Maximum Likelihood classification for matching a given utterance against a predefined vocabulary represented by HMMs. The algorithms we used were quite straightforward and had a very regular structure. The flip side of the coin is that this simple approach has several disadvantages, of which at least some should be mentioned here:

- No matter what the utterance is, our algorithm will always produce a sequence of words out of our vocabulary.

- The connected word recognizer we described is unable to either provide us with a confidence measure for the accuracy of the recognized word sequence or detect "out of vocabulary" words.

- We will get only the best match for the word sequence and no alternative "n-best" word sequences are presented.

- During the recognition process, only the acoustic knowledge represented by the HMMs is used, no other "higher–level" knowledge sources are exploited.

- It has no knowledge about any semantic restrictions of the word sequence and is in no way capable of really "understanding" the utterance.

Chapter 9

Conclusion

This short introduction covered the basics of speech recognition up to the principles of connected word recognition. We also pointed out the disadvantages of these primitive algorithms. However, there are so many open topics we did not even come close to deal with. Among them are the following:

- We know now how the basic algorithm for connected word recognition with HMMs works. — But how does the training procedure for HMMs look like?
- If we have large vocabularies, we will not have enough training samples for the creation of word models. How can we build word models from smaller units which can be reliably estimated?
- How can we search "n-best" word sequences instead of the best word sequence only?
- How can we integrate additional knowledge sources like finite state grammars or stochastic language models into the recognition process?
- How can we reduce the computational complexity of the search algorithm so that we are able to meet real time constraints ?

We will learn more about some of these topics in the successor of this textbook which will be entitled **"Speech Recognition: Selected Topics"** and will be available soon !

Bibliography

[Bra00] R. N. Bracewell. *The Fourier Transform and its Applications*. McGraw-Hill, 2000.

[Bri87] E. Oran Brigham. *Schnelle Fourier Transformation*. Oldenbourg Verlag, 1987.

[CHLP92] L. R. Rabiner C.-H. Lee and R. Pieraccini. *Speaker Independent Continuous Speech Recognition Using Continuous Density Hidden Markov Models.*, volume F75 of *NATO ASI Series, Speech Recognition and Understanding. Recent Advances. Ed. by P. Laface and R. De Mori*. Springer Verlag, Berlin Heidelberg, 1992.

[HL92] X. D. Huang and K. F. Lee. Phonene classification using semicontinuous hidden markov models. *IEEE Trans. on Signal Processessing*, 40(5):1962–1067, May 1992.

[Jel98] F. Jelinek. *Statistical Methods for Speech Recognition*. MIT Press, 1998.

[LRS85a] S. E. Levinson L.R. Rabiner, B.H. Juang and M. M. Sondhi. Recognition of isolated digits using hidden markov models with continuous mixture densities. *AT & T Technical Journal*, 64(6):1211–1234, July-August 1985.

[LRS85b] S. E. Levinson L.R. Rabiner, B.H. Juang and M. M. Sondhi. Some properties of continuous hidden markov model representations. *AT & T Technical Journal*, 64(6):1251–1270, July-August 1985.

[LRS89] J. G. Wilpon L.R. Rabiner and Frank K. SOONG. High performance connected digit recognition using hidden markov models. *IEEE Transactions on Acoustics, Speech and Signal Processing*, 37(8):1214–1225, August 1989.

[Ney84] H. Ney. The use of a one-stage dynamic programming algorithm for connected word recognition. *IEEE Transactions on Acoustics, Speech and Signal Processing*, ASSP-32(2):263–271, April 1984.

[Ney93] H. Ney. Modeling and search in continuous speech recognition. *Proceedings of EUROSPEECH 1993*, pages 491–498, 1993.

[Rab89] L.R. Rabiner. A tutorial on hidden markov models and selected applications in speech recognition. *Proceedings of the IEEE*, 77(2):257–286, February 1989.

[RJ86] L.R. Rabiner and B.H. Juang. An introduction to hidden markov models. *IEEE ASSP Magazine*, pages 4–16, January 1986.

[Rus88] G. Ruske. *Automatische Spracherkennung*. Oldenbourg Verlag, München, 1988.

[ST95] E. G. Schukat-Talamazzini. *Automatische Spracherkennung*. Vieweg Verlag, 1995.

[SYW94] J.J. Odell S. Young and P.C. Woodland. Tree-based state tying for high accuracy acoustic modelling. *Proc. Human Language Technology Workshop, Plainsboro NJ, Morgan Kaufman Publishers Inc.*, pages 307–312, 1994.

[XDHH90] K. F. Lee X. D. Huang and H. W. Hon. On semi-continuous hidden markov modeling. *Proceedings ICASSP 1990, Albuquerque, Mexico*, pages 689–692, April 1990.

[XHH93] F. Alleva X. Huang, M. Belin and M. Hwang. Unified stochastic engine (use) for speech recognition. *Proceedings ICASSP 1993,*, II:636–639, 1993.

[You92] S. J. Young. The general use of tying in phoneme-based hmm speech recognisers. *Proceedings of ICASSP 1992*, I(2):569–572, 1992.

[You96] S. Young. Large vocabulary continuous speech recognition: A review. *IEEE Signal Processing Magazine*, 13(5):45–57, 1996.

[You01] S. Young. Statistical modelling in continuous speech recognition. *Proc. Int. Conference on Uncertainity in Artificial Intelligence, Seattle, WA*, August 2001.