AUTOMATIC SPEECH RECOGNITION USING HIDDEN MARKOV MODELS

IN4012TU
Real-time AI & Automatische Spraakherkenning
September 2003

Ir. P. Wiggers
Dr. drs. L.J.M. Rothkrantz

Data and Knowledge Systems Group

TU Delft

TABLE OF CONTENTS

1 INTRODUCTION

Speech is the most natural means of communication between humans; it can be done without any tools or any explicit education. It is one of the first skills we learn to use. Babies quickly learn how to react to the voice of their mother and they even quicker learn to produce noise when they are in need. When speaking with somebody, one does not have to focus on this person; one can look in a different direction or even perform some other task while communicating.
Speech is also the most important way of communicating. It has always been; before mankind invented writing, the spoken word was the only way of passing knowledge. Ancient poets like Homer and Ovidius originally wrote their still famous epic poems for recitation not for reading. Despite all our novel ways of communicating, like e-mail and chat, speech is still the number one means of communication, a fact once again proven by the immense popularity of cellular phones.

So it is only logical that machine interface designers in their quest for a natural man-machine interface have turned to automatic speech recognition and speech production as one of the most promising interfaces. In the last 30 years researchers from areas like psychology, linguistics, electrical engineering and computer science have worked on this subject. While the first systems could only differentiate between 'yes' and 'no', currently, speaker independent systems exist with a vocabulary of over 60000 words that can recognize continuous speech (that is complete sentences or paragraphs) with an accuracy of 95% or higher.

Although different techniques have been developed for automatic speech recognition, ranging from knowledge-based systems to neural networks, the main engine behind this progress and currently the dominant technology, has been the data driven statistical approach based on Hidden Markov models. In this syllabus an introduction to statistical speech recognition is given. Chapter two highlights the problems in speech recognition and describes the theory of statistical speech recognition. It especially concentrates on the Hidden Markov model, which is the core of the recognition process. Chapter three introduces the Hidden Markov Toolkit, a set of tools that implement the algorithms described in chapter three which can be used to develop a speech recognition system. Chapter four presents a case study of the development of a speech recognizer for the Dutch language, it shows how the theory form the previous chapters can be used and concentrates on the practical issues, like data selection, model definition and system refinement.

2 SPEECH RECOGNITION

This chapter describes the theory of speech recognition. It starts with a description of the difficulties in recognizing speech. Next an overview of the entire process is given and the subsystems of a recognizer are identified. Each of these components is then described. Especially the Hidden Markov Model, which constitutes the core of the recognition process, is described in detail.

2.1 INTRODUCTION

Why is automatic speech recognition such a difficult problem that, after 30 years of research, it still has not been solved? At first sight it may seem just a matter of classifying sounds using some typical characteristics of these sounds. This approach, called the acoustic approach was indeed tried, but only with limited results. Finding explicit characteristics of speech sounds that suffice to classify them proved extremely hard.
A second approach to recognizing, that does not directly rely on a set of characteristics is the statistical pattern recognition approach, which has already been successfully been applied to problems like the automatic recognition of handwriting. The pattern recognition approach did prove to be fruitful, but only after advanced models were developed.

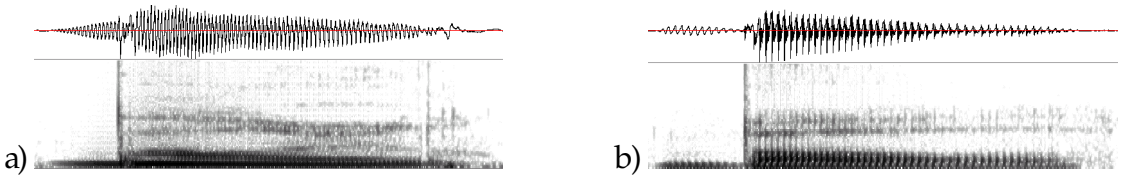


Figure 2.1- The word ball spoken by two different speakers: (a) female and (b) male

Figure 2.1 shows why the simple pattern recognition approach of classifying signals with a similar shape as the same sound, is not powerful enough to perform speech recognition. The figures show the word ball spoken by two different persons. The upper half of the figure shows the raw speech waveform, the lower half shows processed versions of the signal that highlight its formants, which are characteristic for a sound. So speech is person dependent, which is no surprise, as different voices sound different. Figure 2.2 shows the phoneme /e/ spoken by the same person in a number of words.
These pictures look very different. In this case actually context is involved, the exact sound of a speech unit, called a phoneme, depends on its neighboring phonemes. The exact shape of the speech signals also depends on the speed with which is spoken and the mood and the temper of the speaker.

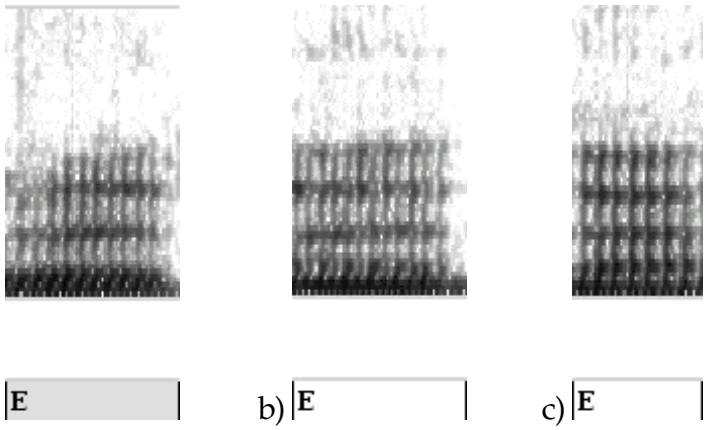


Figure 2.2 - The phoneme */e/* in three different contexts: (a) let's, (b) phonetic and (c) sentence.

2.2 FUNDAMENTALS OF SPEECH RECOGNITION

2.2.1 Components of a speech recognizer

Figure 2.3 schematically shows the speech recognition problem: someone produces some speech and we want to have a system (the box in the figure) that automatically translates this speech, a pressure waveform that is, to a written transcription.

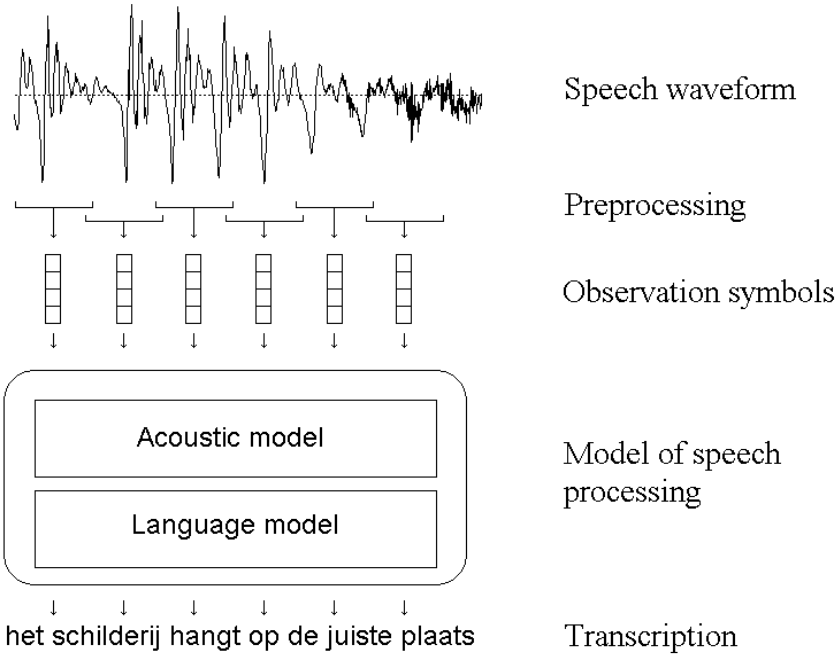


Figure 2.3 - overview of the speech recognition problem

It is possible, after some preprocessing, to represent the speech signal as a sequence of observation symbols $\mathbf{O} = o_1 o_2 \ldots o_T$ that represents a string composed of elements of a alphabet $\mathcal{V}$ of symbols. If, in addition, we have a vocabulary V of all the words w_i, $1 \le i \le |V|$, that can be uttered. Then mathematically the speech recognition problem comes down to finding the word sequence $\hat{\mathbf{W}}$ having the highest probability of being spoken, given the acoustic evidence $\mathbf{O}$, thus we want to solve

$$\hat{\mathbf{W}} = \arg\max_{\mathbf{W}} P(\mathbf{W}|\mathbf{O}) \tag{2.1}$$

Unfortunately, this equation is not directly computable since the number of possible observation sequences is sheer inexhaustible, unless there is some limit on the duration of the utterances and there is a limited number of observation symbols. But Bayes formula gives:

$$P(\mathbf{W}|\mathbf{O}) = \frac{P(\mathbf{W})P(\mathbf{O}|\mathbf{W})}{P(\mathbf{O})} \tag{2.2}$$

Where $P(\mathbf{W})$, called the language model, is the probability that the word string $\mathbf{W}$ will be uttered and $P(\mathbf{O}|\mathbf{W})$ is the probability that when word string $\mathbf{W}$ is uttered the acoustic evidence $\mathbf{O}$ will be observed, the latter is called the acoustic model. The probability $P(\mathbf{O})$ is usually not known but for a given utterance it is of course just a normalizing constant and can be ignored. Thus to find a solution to formula (2.1) we have to find a solution to:

$$\hat{\mathbf{W}} = \arg\max_{\mathbf{W}} P(\mathbf{W})P(\mathbf{O}|\mathbf{W}) \tag{2.3}$$

Consequently, a speech recognizer consists of three components: a preprocessing part that translates the speech signal into a sequence of observation symbols, a language model that tells us how likely a certain word string is to occur and an acoustic model that tells us how a word string is likely to be pronounced. In the next sections these three subsystems will be described.

2.2.2 Acoustic processing

The first step in speech recognizer design is to decide what acoustic data $\mathbf{O}$ will be observed. Therefore a front end is needed that will transform the original waveform into a sequence of symbols o_i with which the recognizer will deal. Strictly speaking this is not necessary, speech recognition could be done by performing pattern recognition algorithms directly on the speech signal, as this signal contains all information. But as mentioned before there are many possible variations in a speech signal and visually similar waveforms do not necessarily indicate perceptually similar sounds. Therefore some preprocessing may be useful to reduce the amount of noise introduced by the environment and the recording hardware and to reduce correlation in the input signal and to extract relevant features.

Many different ways to extract meaningful features have been developed, some based on acoustic concepts or knowledge of the human vocal tract and psychophysical knowledge of the human perception. Much work has been done in the field of signal processing; the most important methods here are Linear Predictive Coding and Mel Frequency Cepstral Analysis. An impression of these two techniques is given below.

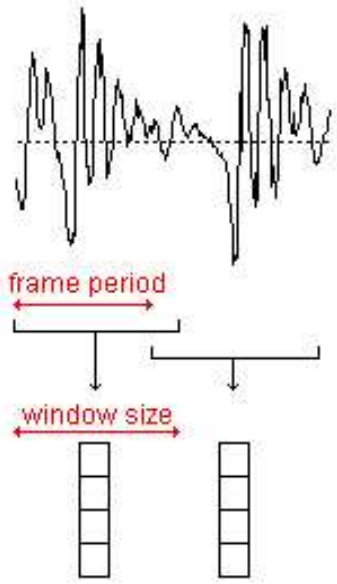


Figure 2.4 - feature extraction

Figure 2.4 illustrates the overall feature extraction process. A sampled waveform is converted into a sequence of parameter vectors at a certain frame rate. A frame rate of 10 ms is usually taken, because a speech signal is assumed to be stationary for about 10 ms, as can be seen from figure 2.5. The segment of a waveform that is used to determine each parameter vector is referred to as a window. The window size and the frame rate are independent. Normally, the window size will be larger than the frame rate, so that successive windows will overlap to make up for discontinuity in the signal introduced by the discrete sampling of the signal.

2.2.2.1 Linear predictive coding

Linear predictive coding is one of the most useful methods for encoding good quality speech at a low bit rate. It takes a mechanical view of the human speech production, assuming that the speech signal is produced by a buzzer at the end of a tube. The space between the vocal chords, called the glottis, produces the buzz, which is characterized by its intensity (loudness) and frequency, which determines the pitch of the sound. The vocal tract, that is the combination of the throat and the mouth, forms a tube, which is characterized by its resonances, called formants. According to the laws of physics these resonances will change when the shape of the vocal tract is changed for example by movement of the tongue, as the length of the tube is a multiple of the wavelengths of the formant frequencies. These formant frequencies are thus representative for a sound. Figure 2.5 shows the waveform of the phoneme /o/ in the word opera. A repeating

pattern can be seen that can be described as a large peak with two smaller peaks on top of it. The large peaks are a result of the base frequency of the signal, that is the first formant frequency, while the smaller peaks show the second formant frequency.

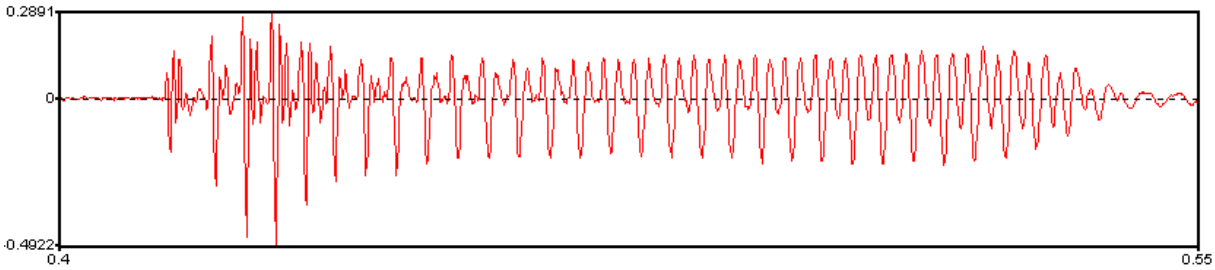


Figure 2.5 - Waveform of /o/ in the word opera

LPC analyzes the speech signal frames by estimating a filter that corresponds to the shape of the vocal tract, removing its effects from the speech signal, and estimating the intensity and frequency of the remaining buzz, called the residue.
The basic problem of LPC is to determine the 'vocal tract' filter. The solution is a difference equation, called a linear predictor, which expresses each sample of the signal as a linear combination of previous samples. The coefficients of the difference equation, the prediction coefficients, characterize the filter and thus the shape of the vocal tract. These coefficients are estimated by minimizing the mean-square error between the predicted signal and the actual signal. For LPC based speech recognition the first 12 coefficients are taken as a feature vector.

2.2.2.2 Mel-frequency cepstral analysis

One of the more common techniques of studying a speech signal is via the power spectrum. The power spectrum of a speech signal describes the frequency content of the signal over time. Typically, the peaks in a spectrum relate to the formant frequencies of a sound, which are the resonances of the vocal tract. Formant frequencies are important indications of the identity of a sound, as a matter of fact vowels can be pretty much characterized by the relative distance between the first and second formant frequency.
The first step towards computing the power spectrum of the speech signal is to perform a Discrete Fourier Transform (DFT). A DFT computes the frequency information of the equivalent time domain signal. Since a speech signal contains only real point values, we can make use of this fact and use a real-point Fast Fourier Transform (FFT) for increased efficiency. The resulting output contains both the magnitude and phase information of the original time domain signal.
Psychophysical studies have shown that human perception of the frequency content of sounds, either for pure tones or for speech signals, does not follow a linear scale. This research has led to the idea of defining subjective pitch of pure tones. Thus for each tone with an actual frequency f, measured in Hz, a subjective pitch is measured on a scale called Mel scale. As a reference point,

the pitch of a 1kHz tone, 40 dB above the perceptual hearing threshold, is defined as 1000 Mels. Other subjective pitch values are obtained by adjusting the frequency of a tone such that it is half or twice the perceived pitch of a reference tone with a known Mel frequency.
Feature extraction based on Mel Frequency Cepstral Coefficients (MFCC) utilizes the power spectrum of which the center frequency and bandwidth are scaled by the subjective Mel measure. The cepstral coefficients are then computed by taking the logarithm of the power spectrum and transforming this log spectrum to the cepstral domain using an inverse Discrete Fourier Transform. This signal can be further decorrelated by taking the cosine of the signal. Cepstral coefficients can be seen as a parametric representation of the (envelope) of the spectrum, as such they correlate with the formant frequencies and thus the shape of the vocal tract at the analyzed frame.
Usually, first and second derivatives are taken from the cepstral coefficients and added to the speech vector to account for the continuous nature of the signal.

2.2.3 Language modeling

Formula 2.3 requires that we are able to compute for every word sequence **W** the a priori probability $P(\mathbf{W})$ that the speaker wishes to utter **W**. As **W** is a string, these probabilities can be decomposed as follows:

$$P(\mathbf{W}) = P(w_1, w_2, ..., w_m) = P(w_1)P(w_2 \mid w_1)P(w_3 \mid w_1, w_2)\ldots P(w_m \mid w_{1,} w_2 \ldots w_{m-1}) \qquad (2.4)$$

Where $P(w_i \mid w_1, w_2, ..., w_{i-1})$ is the probability that w_i will be spoken given that the words $w_1 w_2 ... w_{i-1}$ were said previously. The choice of w_i thus depends on the entire history of the discourse sofar. In reality it is impossible to estimate these probabilities even for moderate values of *i* as most of these histories would be unique. For a vocabulary of size $|V|$ there are $|V|^{i-1}$ different histories. For example even for a relatively small vocabulary of 1000 words and i=3 there would be one billion different histories.
The number of probabilities to compute can be reduced drastically by considering only limited histories. This will effectively map the histories to a limited number of equivalence classes. For example when using only a two-word history the partial sentences:

```
It rained all day and …
It rained very hard all day and …
The man I just spoke to said that is rained very hard all day and …
```

Can now be considered to be equivalent as they all map to '`day and …`'. Doing so will preserve most of the model's ability to predict the next word, as most words do not really depend on a large history. Language is highly structured and limited histories are capable of capturing quite a bit of this structure, like local grammatical constraints, for example the fact that a determiner, like the is often followed by a noun and not by a verb as well as the fact that certain words tend to occur together, for example one can say strong tea but it is highly unlikely that someone will refer to tough tea. A simple but effective approach is the bigram, this model uses only one step histories.

$$P(\mathbf{W}) = P(w_1)P(w_2|w_1)....P(w_m|w_{m-1}) \quad (2.5)$$

It is now easy to estimate the probabilities $P(w_i|w_{i-1})$ simply by counting the number of times each word pair occurs in a representative corpus of strings. One problem with this procedure is that the training corpus might be missing some words, which do occur in the vocabulary the recognizer uses. To these words would be assigned an estimated probability of zero. Therefore, usually a small portion of the probability distribution is reserved for words out of the vocabulary that do not occur in the training corpus.
Most present-day speech recognizers take it one step further and use the trigram language model that has two word history equivalence classes. This language model is surprisingly powerful but requires large amounts of training data to provide values for all probabilities $P(w_i|w_{i-1}, w_{i-2})$ as can be seen in the above example. Furthermore the model tends to get very large as all the words have to be listed and for each word all possible two-word histories must be provided.
To circumvent these problems some extensions to the basic statistical language models are possible. To make up for data sparseness the trigram frequencies can be smoothed by interpolating trigram, bigram and unigram relative frequencies.

$$P(w_i|w_{i-1},w_{i-2}) = \lambda_1 f(w_i) + \lambda_2 f(w_i|w_{i-1}) + \lambda_3 f(w_i|w_{i-1}, w_{i-2}) \quad (2.6)$$

Where the weights satisfy $\lambda_1+\lambda_2+\lambda_3 = 1$.
Another, similar method to make up for the lack of training data used in many speech recognizers is backing-off. Arguing that if there is enough evidence the trigram frequency is a good estimate for the probability, if not the system should back-off and rely on bigrams and if there is not enough evidence even for those, unigrams should be used. Of course it is possible to create any *N*-gram, $N > 3$, but with it comes a combinatorial increase in model complexity and size and an unsatisfiable hunger for training data.

The big advantage of statistical language models is that they are relatively simple to obtain. All that is needed are some representative texts and for large vocabulary recognition tasks these models may be the only realistic option. However, creation of a representative corpus of text may be a difficult and time-consuming effort. If there is an existing human-human interface for the area of interest a vocabulary can be created using the transcriptions of these dialogs. If a new ASR application has to be designed for which there is no human counterpart we can simulate speech recognizing systems by using Wizard of Oz studies to create a corpus. For smaller, more specialized tasks or dialogs constraints on the form of the utterances can be found, that is a grammar can be defined. Within this grammar we can still use statistics to attach probabilities to the different paths through the grammar. These grammars offer a powerful language model because of the constraints they impose. However, it is important to note that there is no need for these grammars to be completely accurate. They can be open to counter examples, because their purpose is to distribute probability among different futures and not exact analysis of the utterances (as would be the case in for example speech synthesis). In the example below a grammar for a simple telebanking application, that can be used to manage one's bank account and make financial transactions by telephone, is defined, using EBNF notation.

```
daytime    ::= 'morgen' | 'middag' | 'avond'
greeting   ::= 'goede' daytime
digit      ::= 'een' | 'twee' | 'drie' | 'vier' | 'vijf' | 'zes' | 'zeven'
               |'acht' | 'negen'
digit0     ::= 'nul' | digit
number     ::= digit0 digit0 digit0 digit0 digit0 digit0 digit0 digit0
tenfolds   ::= 'twintig' | 'dertig' | 'veertig' | 'vijftig' |  'zestig'|
               'zeventig' | 'tachtig' | 'negentig'
hondreds   ::= [ digit ] 'honderd'
amount     ::= [hondreds] [ digit 'en' ] tenfolds
please     ::= 'alstublieft' | 'alstjeblieft' | 'graag' | 'willen'
want       ::=  'ik' ('wil' | 'wilde' | 'zou graag')
mine       ::= ['van'] 'mijn'
account    ::= ['prive'] ['bank'] rekening
specific   ::= [mine] account [['nummer'] number]
money      ::= ( ['wat'] 'geld' ) | (amount ('guldens' | 'piek'))
open       ::= 'een nieuwe' account 'openen' [please]
close      ::= specific 'sluiten'
deposit    ::= money ( ( 'op' specific 'storten' ) |
                       ( 'storten op' specific) ) ;
withdrawal ::= money ( ( 'van' specific 'opnemen' ) |
                       ('opnemen van' specific) )
transfer   ::= money 'van' specifict ['naar' specific] 'overmaken'
info       ::= ( 'saldo van' specific 'opvragen') |
               ('naar het saldo van' specific 'informeren')
command    ::= [greeting] want ( open | close | deposit | withdrawal |
               transfer | info) [please]
```

Figure 2.6 - An EBNF grammar for a telebanking application

The application is intended to recognize commands like:

```
"Goede morgen, ik zou graag drieenveertig guldens van mijn priverekening
opnemen, alstublieft"

"Ik wilde honderdzevenentwintig guldens overmaken naar bankrekening een
twee drie vier vijf zes zeven acht"
```

But a closer inspection of the grammar reveals that it also allows for sentences like:

```
"Goede middag, ik graag nul piek op van mijn rekening storten, graag"
```

that do not conform to the rules of Dutch grammar. Semantically this sentence does not make much sense either. But the point is that this sentence is highly unlikely ever to be uttered, so there is no need to make the grammar overly complicated just to prevent this sentence from occurring. And actually, it is not even desirable, as spoken language does not adhere to grammar rules per se. Making them too restrictive may lead to even bigger problems, because the system then starts recognizing things that were never said. For example, in the telebanking application the following sentence is not unlikely to be spoken:

```
"Goede morgen, uh…, goede middag, ik zou graag drieentwintig, nee toch maar
vijftig gulden naar de bank rekening van de TU Delft over willen maken"
```

Because of the hesitations and some unknown words the above grammar cannot recognize this sentence. It will, however, recognize parts of the sentence and may for example conclude that the caller wants to transfer twenty-three guilders to some bank account. It will then try to fit the sounds in `TU Delft` to some account number. Thus, the judgment whether a sentence makes sense is better left up to the application that uses the speech interface.

2.2.4 Acoustic modeling

The acoustic model determines what sounds will be produced when a given string of words is uttered. Thus for all possible combinations of word strings **W** and observation sequences **O** the probability $P(\mathbf{O}|\mathbf{W})$ must be available. This number of combinations is just too large to permit a lookup, in the case of continuous speech its even infinite. It follows that these probabilities must be computed on the fly, so a statistical acoustic model of the speakers' interaction with the recognizer is needed. The total process modeled involves the way the speaker pronounces the words of **W**, the ambience (room) noise, the microphone placement and characteristics and the acoustic processing performed by the signal processing front end. The most frequently used model these days is the Hidden Markov model but other models are possible, for instance artificial neural networks. The next paragraph describes the Hidden Markov model in detail.

2.3 HIDDEN MARKOV MODELS

2.3.1 Introduction

The Hidden Markov model (HMM) is a very powerful mathematical tool for modeling time series. It provides efficient algorithms for state and parameter estimation, and it automatically performs dynamic time warping for signals that are locally squashed and stretched. It can be used for many purposes other than acoustic modeling.

2.3.2 Markov Chains

Hidden Markov models are based on the well-known Markov chains from probability theory that can be used to model a sequence of events in time. Figure 2.7 shows such a graphical network representation of such a model, it has two states a and b and some connections indicated by arrows that show how one can get from one state to another[1]. The topology of the network shows an important property of Markov chains, namely that the next state only depends on the current state the model is in, regardless of how it got in the current state; this property is often referred to as the Markov property.
By starting in one of the two states and at each time step moving through the model following the arrows out of the current state to the other state or once again to the same state, sequences of a's and b's can be generated.

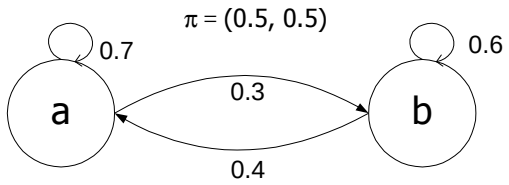


Figure 2.7 – A Markov Chain

The arrows leaving a state are annotated with a probability that indicates how likely it is that this particular transition out of the state will be chosen. As a transition has to be made the probabilities associated with all arrows leaving a state should sum to one. The distribution π indicates how likely each state is to be the start state, in Figure 2.7 both states are equally likely to be the start state. Using these probabilities a Markov model can be used for recognition. Imagine that we have two processes that produce outputs that can be encoded as sequences of a's and b's and each of these processes can be modeled by a Markov model, the one from Figure 2.7 and the one from Figure 2.8.

[1] Computer scientist may note the resemblance of this model with a finite state machine (or transition diagram) and indeed a Markov model can be interpreted as a probabilistic version of such a model.

If we now receive for example a sequence abba we can calculate for each model the probability that it generated this sequence by simply multiplying the probabilities along the path that corresponds to the sequence[2].

Probability model 1: $0.5 \bullet 0.3 \bullet 0.6 \bullet 0.4 = 0.036$
Probability model 2: $0.4 \bullet 0.5 \bullet 0.7 \bullet 0.3 = 0.042$

After which we may conclude that the output is most likely to be generated by the second process.

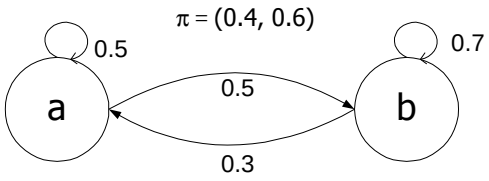


Figure 2.8 – A second Markov model

2.3.3 Definition of Hidden Markov models

In the Markov model each state corresponds to one observable event. But this model is too restrictive, for a large number of observations the size of the model explodes, and the case where the range of observations is continuous is not covered at all.
Therefore the Hidden Markov concept extends the model by decoupling the observation sequence and the state sequence. For each state a probability distribution is defined that specifies how likely every observation symbol is to be generated in that particular state. As each state can now in principle generate each observation symbol it is no longer possible to see which state sequence generated a observation sequence as was the case for Markov models, the states are now hidden, hence the name of the model. A Hidden Markov model can be defined by the following parameters:

- The number of distinct observation symbols M.
- An output alphabet $\mathcal{V} = \{v_1, v_2, \ldots v_M\}$
- The number of states N.
- A state space $Q = \{1, 2, \ldots N\}$

  States will usually be indicated by i, j a state that 'the model is in' at a particular point in time t will be indicated by q_t. Thus, $q_t = i$ means that the model is in state i at time t.
- A probability distribution of transitions between states $\mathbf{A} = \{a_{ij}\}$, where

  $a_{ij} = P(q_{t+1} = j \,|\, q_t = i) \quad 1 \le i, j \le N$

[2] Informally, multiplying (independent) probabilities may be interpreted as an and operation, while summing corresponds to an or operations. Thus in model 1, we first go to state a with probability 0.5 and next we move to state be with probability 0.3 and so on.

- An observation symbol probability distribution $\mathbf{b} = \{b_j(k)\}$ in which
  $b_j(k) = P(o_t = v_k \,|\, q_t = j) \quad 1 \le k \le C, 1 \le j \le N$
- The initial state distribution $\boldsymbol{\pi} = \{\pi_i\}$ where $\pi_i = P(q_1 = i) \quad 1 \le i \le N$

To indicate the complete parameter set of a model the notation $\lambda = (\mathbf{A}, \mathbf{B}, \boldsymbol{\pi})$ is used.
Figure 2.9 shows a three state HMM with discrete probability distributions attached to each state, capable of generating the symbols 0 and 1.
.

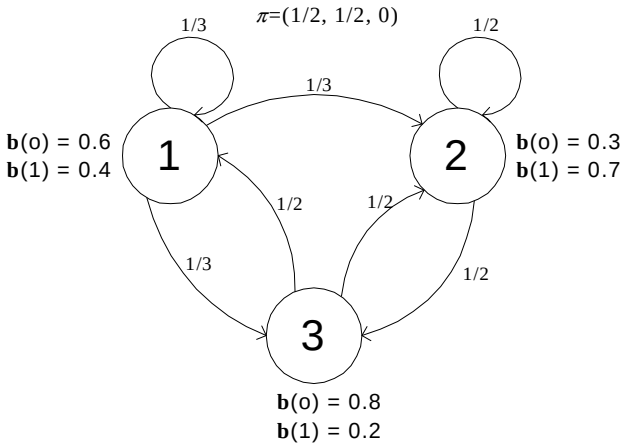


Figure 2.9 - A simple discrete Hidden Markov model

2.3.4 HMM generator of observations

Given appropriate values of N, M, $\mathbf{A}$, $\mathbf{B}$, π and an alphabet the HMM can be used as a generator to produce an observation sequence $\mathbf{O} = o_1 o_2 \ldots o_T \in \mathcal{V}^*$ by performing the following steps:

Choose an initial state $q_1 = i$ according to the initial state distribution π.
Set $t = 1$.
Choose $o_t = v_k$ according to $b_j(k)$.
$t = t + 1$, transit to a new state $q_{t+1} = j$ according to a_{ij}.
Return to step 3 if $t < T$.

Recognition with a Hidden Markov model works the other way around, as it did with Markov models. A observation sequence from some process we want to study is given, for example a speech signal represented by a sequence of vectors, and we have a number of models that represent the things we want to recognize, for example the words that can be spoken. We then want to know which of these models is most likely to generate the given observation sequence,

that is we want to know the probability that the observation sequence was produced by the model λ.
Now, when we have an observation sequence and know which model is most likely to generate the sequence, it is not clear how the model generates that sequence, because the underlying state sequence is hidden. Since practically any state sequence could generate each observation sequence there is no correct state sequence to be found here. All we can do is try to solve this problem as best as possible and seek the most likely state sequence given the observation sequence and the model.
The above discussion assumes that we already have a model available, an important question is of course how to obtain such a model. This is called the training problem, since the model parameters of a HMM can be obtained from a set of example data.
The next three sections discuss each of these fundamental question concerning Hidden Markov models in turn.

2.3.5 Forward algorithm

Imaging we would have the observation sequence 10110 and we would like to know how likely it is that this sequence is generated by the model from figure 2.7. One of the paths through the model that could generate this sequence is **q**=12312. The probability of this path and the corresponding observation sequence would be:

$$P(\mathbf{q},\mathbf{O}|\lambda) = \pi_1 b(1) a_{12} b(0) a_{23} b(1) a_{13} b(1) a_{12} b(0) \tag{2.7}$$

However, as can be easily seen there are many other paths through the model that can also generate this sequence. To get the overall probability of the observation sequence given the model we should add the probabilities of these alternative paths together:

$$P(\mathbf{O}|\lambda) = \sum_{\mathbf{q}_1\mathbf{q}_2,\ldots,\mathbf{q}_{N^t}} P(\mathbf{O},\mathbf{q}|\lambda) = \sum_{\mathbf{q}_1\mathbf{q}_2,\ldots,\mathbf{q}_{N^t}} \pi_{q_1} b_{q_1}(o_1) a_{q_1 q_2} b_{q_2}(o_2) \ldots a_{q_{T-1} q_T} b_{q_T}(o_T) \tag{2.8}$$

By regrouping the terms this may be written as:

$$P(\mathbf{O}|\lambda) = \sum_{\forall \mathbf{q}} P(\mathbf{q}|\lambda) \prod_{t=1}^{T} P(o_t|q_t,\lambda) \tag{2.9}$$

As can be seen from these equations for an observation sequence of length T for each state sequence **q** about $2T$ calculations have to be performed. At each time step there are N different states that can generate a given observation, therefore there are N^T different state sequences that can generate an observation sequences. It follows that the complexity of formula 2.9 is $\mathcal{O}(2TN^T)$.
For realistic values of N and T this quickly becomes infeasible, therefore a more efficient procedure is needed. Fortunately, such a procedure exists; it is called the forward procedure. The algorithm is belongs to the class of dynamic programming algorithms, instead of considering

each state sequence in turn it calculates the values for all subsequences at each time step in parallel, using the results from the previous time step. This is highly efficient as many paths share the same sub-paths, for example the paths 1231, 1232 and 1233 only differ in the last state.
For this procedure the probability $\alpha_t(i)$ is defined as the probability of being in state i at time t and having observed the partial observation sequence $o_1 o_2 \ldots o_t$ sofar, given the model λ.

$$\alpha_t(i) = P(o_1 o_2 \ldots o_t, q_t = i \mid \lambda) \tag{2.10}$$

Now $\alpha_t(i)$ can be computed inductively, as follows:
Initialization:

$$\alpha_1(i) = \pi_i b_i(o_1) \quad 1 \le i \le N \tag{2.11}$$

Induction:

$$\alpha_{t+1}(j) = \left[\sum_{i=1}^{N} \alpha_t(i) a_{ij} \right] b_j(o_{t+1}) \quad 1 \le t \le T-1, t \le j \le N \tag{2.12}$$

Termination:

$$P(O \mid \lambda) = \sum_{i=1}^{N} \alpha_T(i) \tag{2.13}$$

Step one initializes the forward probabilities as the joint probability of state i and the initial observation o_1. The induction step follows from the fact that the product $\alpha_i(t) a_{ij}$ is the probability of the event that $o_1 o_2 \ldots o_t$ are observed and state j is reached at time t+1 via state i at time i. Summing this product over all the N possible states, i $1 \le i \le N$, at time t results in the probability of j at time t+1 with all of the accompanying previous partial observations. Now all that is left to do is to take account for observation o_{t+1} in state j, this is done by use of the probability $b_j(o_{t+1})$.

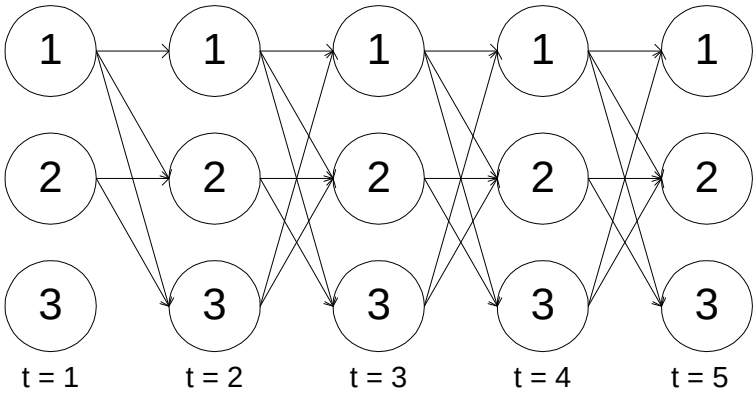


Figure 2.10 - The forward algorithm visualized by a trellis

The forward algorithm can be visualized by a so-called trellis that shows the time evolution of the process. Figure 2.10 shows a trellis for the output sequence 10110 generated by the HMM of Figure 2.9. This figure shows the main thought behind the forward algorithm. At each time step all model states are considered, because there are only *N* nodes at each time step all possible state sequences will remerge into these *N* nodes no matter how long the observation sequence. At time t each calculation only involves the *N* previous values of $\alpha_{t-1}(i)$, because each of the *N* grid points van be reached from only the *N* grid points at the previous time slot. So this procedure only requires on the order of N^2T calculations. Rather than $2TN^t$ as required by the direct calculation.
There is one practical remark to make on the forward procedure: as the calculations involve mainly multiplications the probabilities $P(O|\lambda)$ may become extremely small, in particular if the observation sequence becomes longer or if the observation sequence is very unlikely for the model. To avoid underflow, usually the logarithm of $P(O|\lambda)$ is computed, which has the effect that all multiplications become additions.

2.3.6 Viterbi algorithm

The forward algorithm tells us how likely it is that a sequence of observations is generated by a model λ. However, in some cases the individual states of a Markov model may have some meaning, for example the phonemes in a word, and we might be interested in the sequence of states that is most likely to have generated the observation sequence $O=o_1o_2...o_T$. This boils down to maximizing $P(\mathbf{q}|\mathbf{O},\lambda)$, or equivalently maximizing $P(\mathbf{q},\mathbf{O}|\lambda)$ as:

$$\max_{\mathbf{q}} P(\mathbf{q}|\mathbf{O},\lambda) = \max_{\mathbf{q}} \frac{P(\mathbf{q},\mathbf{O}|\lambda)}{P(\mathbf{O})} = \max_{\mathbf{q}} P(\mathbf{q},\mathbf{O}|\lambda) \tag{2.14}$$

The last step follows form the fact that the probability of the observation sequence can be seen as a constant.
Now we define $\delta_t(i)$ as the best path (the path with the highest probability) from the start into some state *i* at time *t*.

$$\delta_t(i) = \max_{q_1,q_2,..q_{t-1}} P(q_1q_2...q_{t-1}, q_t = i, o_1o_2...o_t|\lambda) \tag{2.15}$$

Notice that

$$\delta_T(i) = \max_{\mathbf{q}=q_1,q_2,...,q_T} P(\mathbf{q},\mathbf{O}|\lambda) \tag{2.16}$$

Is the value we are looking for.
$\delta_t(i)$ can be calculated using a recursive procedure similar to the forward algorithm, but this time using a maximization over previous states instead of a summing procedure. This algorithm is

known as the Viterbi algorithm. The optimal path can be found by keeping track of the argument i that maximized $\delta_t(j)$ in equation 2.15

Initialization:

$$\delta_1(i) = \pi_i b_i(o_1) \quad 1 \le i \le N \tag{2.17}$$

$$\psi_1(i) = 0 \tag{2.18}$$

Recursion:

$$\delta_t(j) = \max_{1 \le i \le N} \left[\delta_{t-1}(i) a_{ij} \right] b_j(o_t) \quad 2 \le t \le T, 1 \le j \le N \tag{2.19}$$

$$\psi_t(j) = \arg\max_{1 \le i \le N} [\delta_{t-1}(i) a_{ij}] \tag{2.20}$$

Termination:

$$\tilde{P} = \max_{1 \le i \le N} [\delta_T(i)] \tag{2.21}$$

$$\tilde{q}_T = \arg\max_{1 \le i \le N} [\delta_T(i)] \tag{2.22}$$

Path backtracking:

$$\tilde{q}_t = \psi_{t+1}(\tilde{q}_{t+1}) \quad t = T-1, T-2, \ldots, 1 \tag{2.23}$$

Once again the algorithm can be visualized using a trellis. Figure 2.11 shows the result of applying the Viterbi algorithm to the HMM of Figure 2.9 to find the state sequence corresponding to **O** = 10110.

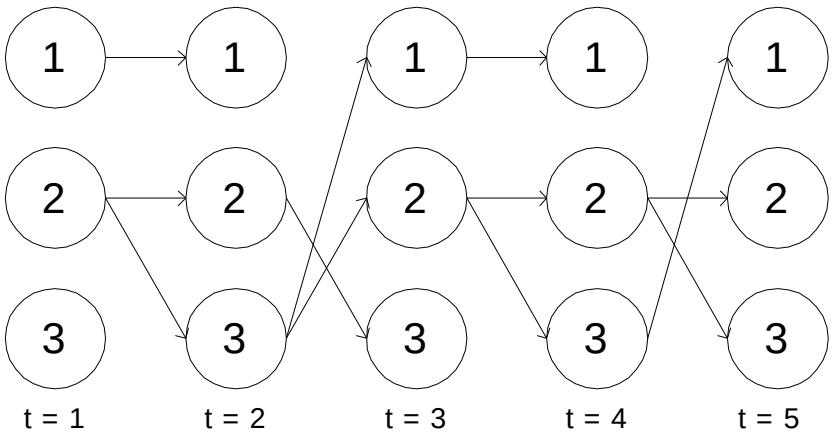


Figure 2.11 - The Viterbi algorithm visualized by a trellis

The calculations below show how the Viterbi path through the trellis is found.

At t = 1 the δ variables are initialized:

$$\delta_1(1) = \pi_1 b_1(1) = \frac{1}{2} \times 0.4 = 0.2$$
$$\delta_1(2) = \pi_2 b_2(1) = \frac{1}{2} \times 0.7 = 0.35$$
$$\delta_1(3) = \pi_3 b_3(1) = 0 \times 0.8 = 0$$

At t = 2:

$$\delta_1(1)a_{11} = 0.2 \times \frac{1}{3} = 0.6667$$
$$\delta_1(2)a_{21} = 0.35 \times 0 = 0$$
$$\delta_1(3)a_{31} = 0 \times \frac{1}{2} = 0$$

Thus $\delta_2(1) = \delta_1(1)a_{11}b_2(0) = 0.02667$

similarily:

$$\delta_2(2) = \delta_1(2)a_{22}b_2(0) = 0.0525$$
$$\delta_2(3) = \delta_1(2)a_{23}b_3(0) = 0.14$$

$$\delta_3(1) = \delta_2(3)a_{31}b_1(1) = 0.0037$$
$$\delta_3(2) = \delta_2(3)a_{32}b_2(1) = 0.01715$$
$$\delta_3(3) = \delta_2(2)a_{23}b_3(1) = 0.0049$$

$$\delta_4(1) = \delta_3(1)a_{31}b_1(1) = 0.0037$$
$$\delta_4(2) = \delta_3(2)a_{22}b_2(1) = 0.0715$$
$$\delta_4(3) = \delta_3(2)a_{23}b_3(1) = 0.0049$$

$$\delta_5(1) = \delta_4(3)a_{31}b_1(0) = 0.00147$$
$$\delta_5(2) = \delta_4(2)a_{22}b_2(0) = 0.0025725$$
$$\delta_5(2) = \delta_4(2)a_{23}b_3(0) = 0.00686$$

At $t = 5$ $\delta_5(3)$ gives the highest probability, the trace back gives state sequence $\mathbf{q} = \{2,3,2,2,3\}$ as a solution. Notice that the most likely sequence cannot be identified before the end of the trellis is reached. It should be clear from the trellis that the Viterbi algorithm takes $O(N^2T)$ calculations. By taking the logarithms of the model parameters, the Viterbi algorithm can be implemented without the need for any multiplications.

As an aside it can be noted that the Viterbi algorithm is not the only possible way to find the optimal state sequence, since there are several possible optimality criteria. For example we could also decide to choose the states q_t that are individually most likely at each time t, which would maximize the expected number of correct individual states but may result in an invalid state sequence. Furthermore the algorithm relies heavily on the Markov property of the underlying model. At each time step t it assumes that the most likely path into the current state will be part of the most likely path over the entire model through this state.

The next section describes a way of estimating the parameters of a HMM, but the Viterbi algorithm can also be used to find an (initial) estimate of the models state distribution parameters, given an observation sequence $\mathbf{O} = o_1 o_2 \ldots o_T$ and a particular HMM whose parameters are to be estimated. The procedure that accomplishes this is called Viterbi alignment. It consists of two steps. In the first step the Viterbi algorithm is used in the normal way, most likely sequence of states $\mathbf{q} = q_1 q_2 \ldots q_n$ that generated $\mathbf{O}$ is found. In the second step once again the concept of a HMM as generator of speech vectors is used. Each state q_i is thought of as having generated the corresponding subset $\mathbf{O}_i$ from the observation sequence. Now q effectively segments $\mathbf{O} = \mathbf{O}_1 || \mathbf{O}_2 || \ldots || \mathbf{O}_n$. These subsets can then be used to calculate the state probabilities.

2.3.7 Baum-Welch algorithm

In the discussion sofar the parameters of the models, that is the transition probabilities and the observation probabilities have been considered as a given. But in practice we usually do not know these values and we therefore would like to have a method that can automatically determine these parameters $\lambda = (A, B, \pi)$ in such a way that the model best matches the observation data it is supposed to represent. mathematically this means we want a method that has the following maximum likelihood property:

$$\hat{\lambda} = \arg\max_{\lambda} P(\mathbf{O}|\lambda) \tag{2.24}$$

$\hat{\lambda}$ would allow the HMM to account best for the observed data O. What we really want is for $\hat{\lambda}$ is to account for as yet unseen data, so the best we can do here is try to use a training set that is as representative as possible. There is no known way to analytically solve for the model parameter set that maximizes the probability of the observation sequence. But we can choose $\lambda = (\mathbf{A}, \mathbf{B}, \pi)$ such that its likelihood $P(\mathbf{O}|\lambda)$ is locally maximized using a sample of the type of data the model is supposed to generate. This can be done by using an iterative procedure known as the Baum-Welch algorithm or Forward-Backward algorithm. Basically, the algorithm counts the average number of times a state is visited and the number of times a transition from a state *i* to another state *j* is made as well as the average number of times that a specific observation symbol is generated in a state. From these counts the transition and observation probabilities are derived. To be able to calculate these counts we first have to define a backward variable $\beta_t(i)$ analogue to the forward variable $\alpha_t(i)$ from section 2.3.3:

$$\beta_t(i) = P(o_{t+1}o_{t+2}...o_T|q_t = i, \lambda) \tag{2.25}$$

That is the conditional probability that $o_{t+1}o_{t+2}\ldots o_T$ are observed and the system starts in state $\mathbf{q}_j$ at time t, given the model λ. $\beta_t(i)$ can be computed inductively:

Initialization:

$$\beta_T = 1 \quad 1 \le i \le N \tag{2.26}$$

Induction:

$$\beta_t(i) = \sum_{j=1}^{N} a_{ij} b_j(o_{t+1}) \beta_{t+1}(j) \quad 1 \le i \le N \tag{2.27}$$

The initialization step arbitrarily defines $\beta_t(i)$ to be 1 for all *i*.

Now we can define the probability of being in state i at time t given the entire observation sequence and the model as:

$$\gamma_t(i) = \frac{\alpha_t(i)\beta_t(i)}{P(\mathbf{O}|\lambda)} \tag{2.28}$$

Intuitively, this formula states that the probability of being in a state can be obtained by taking the probability of all paths that pass through this state. We find all paths trough a state by combining all ingoing paths with every outgoing path, hence the multiplication of the forward and backward probabilities. The nominator in this formula, the probability of the observation sequence, can be interpreted as the probability of all possible paths through the model for this observation sequence and follows from the forward algorithm. It ensures that the γ_t sum to one. Similarly, we can also define the probability of being in state i at time t and state j at time t+1, given the model and the observation sequence by:

$$\begin{aligned}\xi(i,j) &= \frac{P(q_t = i, q_{t+1} = j, \mathbf{O}|\lambda)}{P(\mathbf{O}|\lambda)} \\ &= \frac{\alpha_t(i) a_{ij} b_j(o_{t+1}) \beta_{t+1}(j)}{P(\mathbf{O}|\lambda)}\end{aligned} \tag{2.29}$$

Figure 2.12 schematically shows the reasoning behind this formula, the probability of getting to state j from state *i* is given by a_{ij} while $\alpha_t(i)$ summarizes all possible ways that state *i* may have been reached, in *j*, o_{t+1} is observed with probability $b_j(o_{t+1})$ and finally all possible ways out of *j* are given by $\beta_{t+1}(j)$.

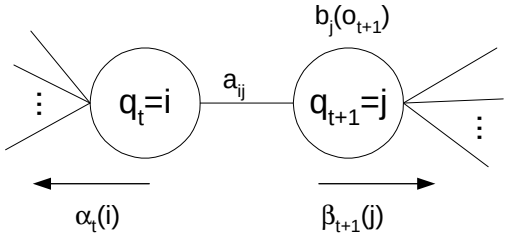


Figure 2.12 – The probability of going from state i at time t to state j at time t+1

If $\gamma_t(i)$ is summed over the time index t, the expected number of times that state i is visited is obtained or equivalently the number of transitions made from state i. Similarly, summation of $\xi_t(i,j)$ over t can be interpreted as the expected number of transitions from state i to state j. Now, using the concept of counting event occurrences, we can estimate a_{ij} as the expected number of transitions from state i to state j normalized by the expected number of transitions from state i:

$$\tilde{a}_{ij} = \frac{\sum_{t=1}^{T-1} \xi_t(i,j)}{\sum_{t=1}^{T-1} \gamma_t(i)} \tag{2.30}$$

And similarly, $b_j(k)$ can be estimated by dividing the expected number of times in state j at which symbol o_k was observed by the expected number of times the system is in state j:

$$\tilde{b}_j(k) = \frac{\sum_{t=1, o_t = o_k}^{T} \gamma_t(j)}{\sum_{t=1}^{T} \gamma_t(j)} \tag{2.31}$$

The initial state distribution for state j is simply equal to the expected frequency with which state j is visited:

$$\tilde{\pi}_j = \gamma_1(j) \tag{2.32}$$

Since the left hand side of these equations also appears on the right hand side we have to use an iterative procedure to improve the model parameters. After starting with an initial guess for A, B, and π, $\hat{\lambda}$ is used in place of λ in each iteration until the values stop changing within certain limits. Training with multiple observation sequences is straightforward. The counts ξ and γ are calculated for each observation sequence the model parameters can now be estimated by averaging over these counts, i.e.:

$$\tilde{a}_{ij} = \frac{\sum_{w=1}^{W} \sum_{t=1}^{T-1} \xi_t(i, j)}{\sum_{w=1}^{W} \sum_{t=1}^{T-1} \gamma_t(i)} \tag{2.33}$$

Where W is the number of observation sequences, the new formulas for b and π change accordingly.

Up to this point the algorithms were explained terms of a finite alphabet of observation symbols and thus discrete probability functions could be used to generate these observations. But as mentioned in section 2.2 in speech recognition continuous signal observation vectors are used. Older system, however, did use a limited codebook of observation symbols. During preprocessing vectors were then mapped to one of the observation symbols using a k-means segmentation. In modern state of the art systems the observation vectors are put directly into the Hidden Markov models, therefore multivariate continuous probability density functions are needed to model the distribution of the observations. The form of the density functions then becomes

$$b_j(o_t) = \mathcal{N}(o_t, \mu_j, \mathbf{U}_j) \tag{2.34}$$

Where $\mathcal{N}$ is an elliptical symmetric density (e.g. Gaussian) with mean vector μ_j and covariance matrix U_j for state j. It can be shown that the re-estimation formulas for these parameters are of the form:

$$\mu_j = \frac{\sum_{t=1}^{T} \gamma_t(j) o_t}{\sum_{t=1}^{T} \gamma_t(j)} \tag{2.35}$$

$$\mathbf{U}_j = \frac{\sum_{t=1}^{T} \gamma_t(j)(o_t - \mu_j)(o_t - \mu_j)'}{\sum_{t=1}^{T} \gamma_t(j)} \tag{2.36}$$

2.3.8 HMM topology for speech recognition

Given a sufficiently large and representative training set, the parameters of a HMM can be estimated as described in the last section, but there is no (known) way to automatically estimate the transition structure of a HMM. Thus the topology of an HMM has to be designed using knowledge of the situation and to some extent, designer's intuition.
The first question that requires an answer here is what kind of unit one HMM should represent. Many choices can be made, for example phrases, words, syllables, phonemes or other sub word units. In fact it is possible to use HMMs to model any unit of speech. Even if the speech units are poorly selected, HMMs have the ability to absorb the suboptimal characteristics within the model parameters; this might of course limit the performance of the system.
Words seem to be the most natural units to model, because they are what we want to recognize and the language model also uses words as basic units. Indeed, recognizers that use word-level models perform rather well. Part of this success is due to the fact that they are able to capture within-word phoneme coarticulation effects. Actually it is shown [1] that because of these effects, the larger the unit, the better the recognition will be. However, as there are many unique words, training data is needed for each of these words, making this kind of system not easily extendible. So for large vocabulary natural speech recognition word units are not really an option. But for small well-defined vocabularies, for example a set of commands, they are well suited. Usually left-to-right model topologies are used in which the number of states depends on the number of phonemes in the word. One state per phoneme is a good rule of thumb.
If sub word units are used, data can be shared among words. This way not all the dictionary entries have to be present in the training phase, the vocabulary of the system is then easily extendible. There are many possible sub word units, typical models include syllable models or linguistically defined sub word units, such as phone models and acoustically defined sub word models, called fenone models.
Linguistically defined sub word models use human specific knowledge for partitioning the parameter space. Acoustically defined sub word units use automatic algorithms to explore the acoustic similarities. Hybrid models, using both acoustic and linguistic knowledge also exist, chapter 4 will discus such a model, called the triphone model.

Phone models are the most used sub word units. Since there are only 40 to 50 phonemes in languages like Dutch and English, HMMs based on phone models can be adequately trained. Most topologies used in speech recognition are based on the assumption that there are three phases in the pronunciation of a phone. In the first phase the vocal tract is changing shape to pronounce the phone, this is called the on-glide of the phone. In this phase there may be some overlap with the preceding phone. In the second phase the sound of the phone is assumed to be pure and in the third phase the sound is released and the vocal tract starts to transit to the next phone. This is called the off-glide, some overlap with the next phone may occur here, the process is schematically shown in Figure 2.13.

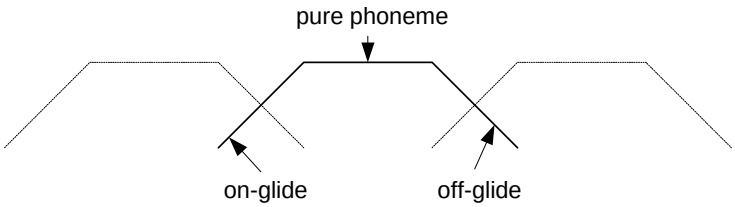


Figure 2.13 – Three phases of a phone

This suggests that at least three states should be used in a phone HMM. Furthermore as a word or even a sentence can be seen as a large string of phones, a left-to-right model structure seems to be necessary, back-loops do not fit very well in this picture.
Adding more states means introducing more parameters and thus more degrees of freedom. Variations in a phoneme can be modeled more accurate but this also introduces a need for more training data to avoid undertraining. And a model should not be too large, a five state model does not work for phones that only occupy three time frames, so in larger models there should always be a 'short-cut' that can handle the shortest example in the training data.
Figure 2.14 shows three model topologies that have successfully been used in various speech recognizers. The first model (2.15a) is a simple three state left-right model with state dependent output probabilities. The first and last smaller circles in the figure represent entry and exit states, these are so called null-states, they do not generate observations and are only used to concatenate the models. The second model (2.15b) has five states, but provides transitions that skip the succeeding state; therefore it is possible to pass through only three steps. This model also has state dependent output probabilities. The last model (2.15c), which is the model used by IBM [2], has seven states and twelve transitions with transition dependent output probabilities. Three groups of output probabilities are tied, corresponding with the three phases in a phoneme. In the figure the begin phase (on-glide) is marked with B the middle phase with M and the end phase (off-glide) with E. As a consequence this model only has three different probability distribution functions.

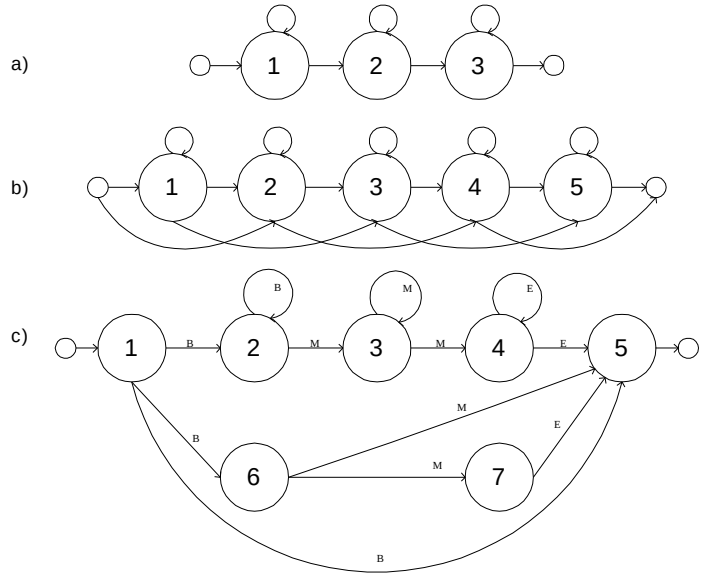


Figure 2.14 - Model topologies for phoneme units

2.3.9 Fitting the pieces together

The previous sections discussed language models and acoustic models but once we have a language model and a set of word level or phone level HMMs how do these pieces fit together? To answer this question we have to notice that a language model can be seen as a network of states (the words) connected by transitions with probabilities attached to them. In other words a language model can be seen as a Markov Model. Taking advantage of the fact that embedding HMMs into an HMM leads to a new HMM we can replace the word states by the corresponding word or phone level HMMs resulting in one huge HMM. In case of phone level HMMs, the phone models have to be concatenated to form a word, before substituting. Figure 2.15 shows a bigram language model for a sequence of the words *one* and *two* with its word models instantiated.

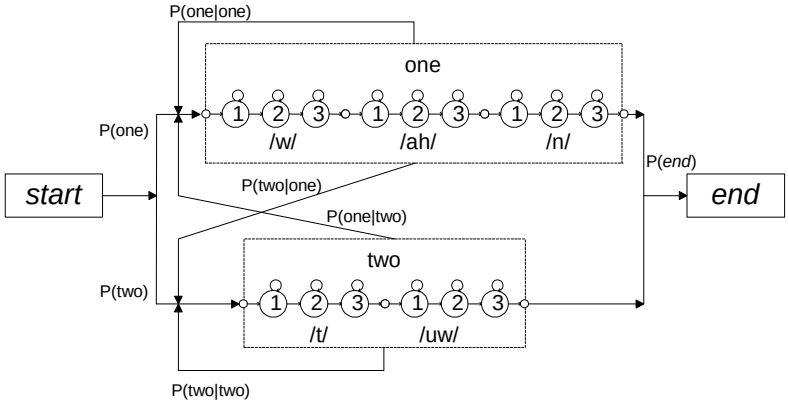


Figure 2.15 - Composite HMM for Viterbi recognition

Now the Viterbi algorithm can be used to find the most likely path through this composite HMM given a sequence of acoustic observation vectors. This path will then lead through a sequence of words that specify the recognized word string. Actually, this method is not guaranteed to find the most likely word sequence, as the probability of a word sequence given an observation sequence would follow from the forward-backward algorithm, which includes all paths that correspond to that **W**. To find the most likely sequence of words this would have to be done for all candidate word sequences. This would computationally be very expensive. Luckily in practice it is very rare that the word string corresponding to the most probable set of paths is the one actually spoken but the one corresponding to the most probable path is not, which justifies the use of the much cheaper Viterbi algorithm.

2.3.10 The Beam search

Problem with this approach is that for practical vocabularies even simple grammars result in a large composite model that has just too many states, making the Viterbi algorithm very slow. But it is possible to reduce the state space without compromising the results too much. This is done by a beam search, hypotheses that are below a certain probability level are pruned. At each trellis stage i the maximum probability P_{i-1}^m of the states at stage i-1 is determined $P_{i\text{-}1}^m = \max \delta_{i-1}(q)$. This value serves as a basis for a dynamic threshold:

$$\tau_{i-1} = \frac{P_{i-1}^m}{K} \tag{2.35}$$

Where K is a suitable chosen constant. Then all states q' on trellis level *i*-1 are eliminated such that $\delta_{i\text{-}1}(q') = 0$ for all states q' that satisfy $\delta_{i\text{-}1}(q') < \tau_{i\text{-}1}$. This purge of improbable paths reduces drastically the number of states entering the comparison implied by the max function in the recursion 2.15 without significantly affecting the values $\delta_i(q)$.

2.3.11 Token passing algorithm

To find the recognized word sequence at the end of the Viterbi search more information is needed beyond the log probability of the path normally calculated by the Viterbi algorithm. An efficient alternative formulation of the algorithm called the token passing model allows for the incorporation of other (meta) information.
In this algorithm each state *i* of the composite HMM at time *t* holds a single movable token which contains, amongst other information, the partial log probability $\delta_t(i)$. This token represents a partial match between the observation sequence $o_1 o_2 \ldots o_t$ and the model subject to the constraint that the model is in state *j* at time *t*. The recursion step of equation (2.19) is then replaced by the equivalent token passing step which is executed at each time frame *t*. The key steps in this algorithm are as follows:

1. Pass a copy of every token in state i to all connecting states *j*, incrementing the log probability of the copy by $\log[a_{ij}] + \log[b_j(o_t)]$
2. Examine the token in every state and discard all but the token with the highest probability.

The history of a token route through the network may be recovered efficiently as follows, for each word instance in the composite HMM a word end null-state is added and every token carries a pointer called a word end link. When a token is propagated from the exit state of a word (indicated by passing through a word end node) to the entry state of another, that transition represents a potential word boundary. Hence a record called a Word Link Record is generated in which is stored the identity of the word from which the token has just emerged and the current value of the token's link. A pointer to the newly created WLR then replaces the token's actual link, effectively creating a linked list of word boundaries.

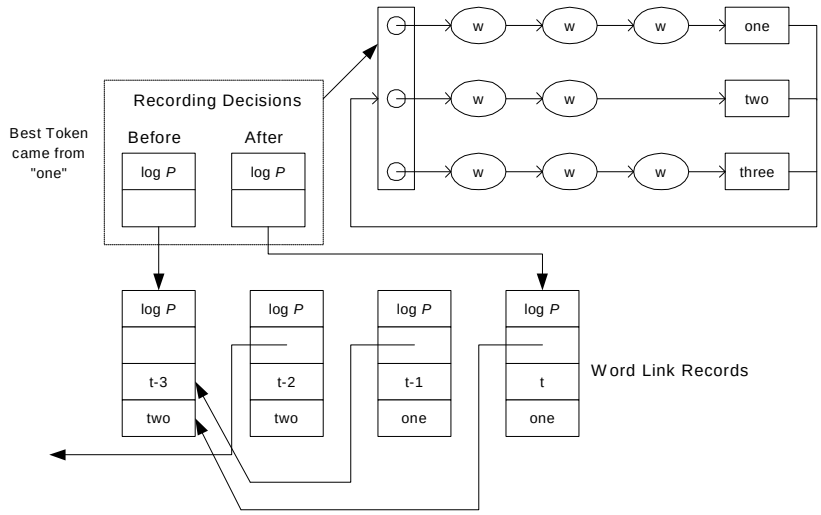


Figure 2.16 - token passing using Word Link records

Once all of the unknown speech has been processed the WLRs attached to the link of the best matching token can be traced back to give the best matching sequence of words. At the same time the positions of the word boundaries can also be extracted if required.

2.3.12 N-best search

The Viterbi algorithm finds the most likely single path through the word network. As mentioned before this is not guaranteed to be the optimal solution, and even if it were it still does not have to be the right solution. Therefore it is often desirable to find the *N* most likely word sequences given the observed acoustic data **O**. For instance, to reprocess the data using more refined models or to parse the alternate hypotheses using syntactic or semantic expert system rules.

The required *N*-best search would differ from the Viterbi search in one aspect only: Instead of retaining only one path leading into each trellis state, each trellis state is split into *N* sub-states, one for each of the *N* most likely paths leading into the unsplit state from the split state of the trellis' previous stage.
The *N*-best search can be incorporated in the token passing algorithm by saving more than just the best token at each word boundary. The resulting lattice *N*-best algorithm is sub optimal, because the use of a single token per state limits the number of different token histories that can be maintained. This limitation can be avoided by allowing each model state to hold multiple tokens and regarding tokens as distinct if they come from different preceding words. This gives a class of algorithms called word *N*-best, which has been shown empirically to be comparable in performance to an optimal *N*-best algorithm [9].

2.3.13 Adaptive training

Although the training and recognition techniques described in this chapter can produce high performance recognition systems when trained on a large corpus of speech data, these systems can be improved upon by customizing the HMMs to the characteristics of a particular speaker. By collecting data from a speaker and training a model set on this speaker's data alone, the speaker's characteristics can be modeled more accurately. Such systems are known as speaker dependent systems, and on a typical word recognition task, may have half the errors of a speaker independent system. The drawback of speaker dependent systems is that a large amount of data (typically hours) must be collected in order to obtain sufficient model accuracy.
Rather than training speaker dependent models, adaptation techniques can be applied. In this case, by using only a small amount of data from a new speaker, a good speaker independent model set can be adapted to better fit the characteristics of this new speaker. This can be done using maximum likelihood linear regression (MLLR), a technique that computes a set of linear transformations by solving a maximization problem using the Expectation-Maximization technique. These transformations will reduce the mismatch between an initial model set and the adaptation data by shifting the means and alter the variances in the initial system so that each state is more likely to generate the adaptation data.
Model adaptation can also be done using a Bayesian approach, in which the speaker independent model parameters and the feature vectors in the adaptation evidence are combined to estimate a new model set. To know how much the model parameters should be changed by the evidence an occupation likelihood for the model state is needed. This can be obtained by running the Viterbi algorithm. Now the next logical step is of course to integrate the Bayesian adaptation approach in the Viterbi algorithm, this way the model set can be adapted on-line during recognition. Each time an utterance is recognized the system is adapted a little more, so the system incrementally gets tailored to the speaker's voice.

3 THE HIDDEN MARKOV TOOLKIT

Subject of this chapter is the Hidden Markov Toolkit, a collection of software libraries and tools for manipulating Hidden Markov models. The architecture of the toolkit will be described and an overview of the tools will be given.

3.1 INTRODUCTION

It may be evident from the last chapter that implementing the algorithms needed to perform the different stages of speech recognition involves quite some work, especially since a lot of optimizations have to be realized to make the algorithms efficient enough for practical use. Fortunately, there are several toolkits available now that implement the algorithms needed in speech recognition, allowing a recognizer designer to focus on the important tasks involved in building a speech recognizer like data preparation, training and recognizers evaluation. One of these toolkits is the Hidden Markov Toolkit (HTK).

HTK is a portable software toolkit for building and manipulating systems that use continuous density Hidden Markov models. It has been developed by the Speech Group at Cambridge University Engineering Department.
HMMs can be used to model any time series and the core of HTK is similarly general purpose. However, HTK is primarily designed for building HMM based speech processing tools, in particular speech recognizers. In can be used to perform a wide range of tasks in this domain including isolated or connected speech recognition using models based on whole word or sub-word units, but it is especially suitable for performing large vocabulary continuous speech recognition.
HTK includes nineteen tools that perform tasks like manipulation of transcriptions, coding data, various styles of HMM training including Baum-Welch re-estimation, Viterbi decoding, results analysis and extensive editing of HMM definitions.

Example A. a training iteration

```
HERest -T 1 -p 8 -B -C d:\htk\P\config1.cfg -S d:\htk\P\train8.scr
       -I d:\htk\P\plab\triph8.mlf -H ts1\hmms.mmf -M ts2 -d ts1
       -t 250.0 150.0 1000.0 -s stat.txt triph.lst
```

Example B. recognition

```
HVite -T 1 -n 3 3 -s 9 -C ..\wdnexp.cfg -H t17m3\hmms.mmf
      -H t15m3\mono.mmf -l * -i rec.mlf -w d:\htk\P\wdsbi.lat
      -S ..\test.scr d:\htk\P\poly2.dic ctriph4m.lst
```

Figure 3.1 - some HTK commands

HTK tools are designed to run with a traditional command-line style interface, each tool has a large number of required and optional arguments and most tools require one or more script files. Figure 3.1 shows some examples of HTK tools in action. Although this style of command-line working results in complex commands, that actually have more resemblance with programming languages than with commands, it has the advantage of making it simple to write shell scripts or programs to control HTK tool execution. Furthermore it allows the details of system construction or experimental procedure to be recorded and documented.

3.2 HTK SOFTWARE ARCHITECTURE

Much of the functionality of HTK is build into library modules, they ensure that every tool interfaces the outside world in exactly the same way and they provide a programming environment for the creation of custom tools or the integration of recognizer functionality in an application. Figure 3.2 shows the library modules and their purpose. User input/output and interaction with the operating system is controlled by the library module HShell and all memory management is controlled by HMem. General mathematical support is provided by Hmath and the signal processing operations needed for LPC and MFCC speech analysis are in HSigP.
Each of the file types required by HTK has a dedicated interface module. HLabel provides the interface for label files, HLM for language model files, HNet for networks and lattices, HDict for dictionaries, HVQ for VQ codebooks and HModel for HMM definitions.
All speech input and output at the waveform level is via HWave and at the parameterized level via HParm. As well as providing a consistent interface, HWave and HLabel support multiple file formats allowing data to be imported from other systems. Direct audio input is supported by HAudio and simple interactive graphics is provided by HGraf. HUtil provides a number of utility routines for manipulating HMMs while HTrain and HFB contain support for the various HTK training tools. HAdapt provides support for the various HTK adaptation tools. Finally, HRec contains the main recognition processing functions.

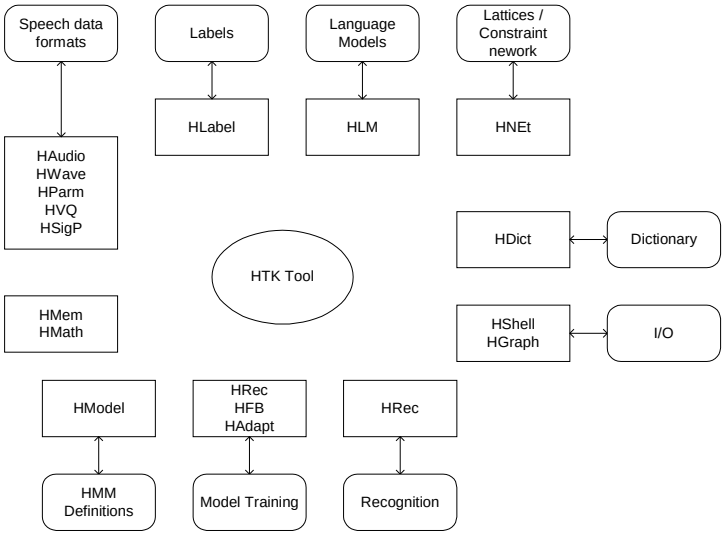


Figure 3.2 - HTK libraries

Fine control over the behavior of these library modules is provided by setting configuration variables or by configuration files that contain a complete list of configuration variables.

3.3 OVERVIEW OF THE HTK TOOLS

Figure 3.3 gives an overview of the HTK tools subdivided into groups according to the processing stages in which they are used. These tools all have a similar interface, which is described, in the next subsection. A short description of each tool is given in the subsequent sections.

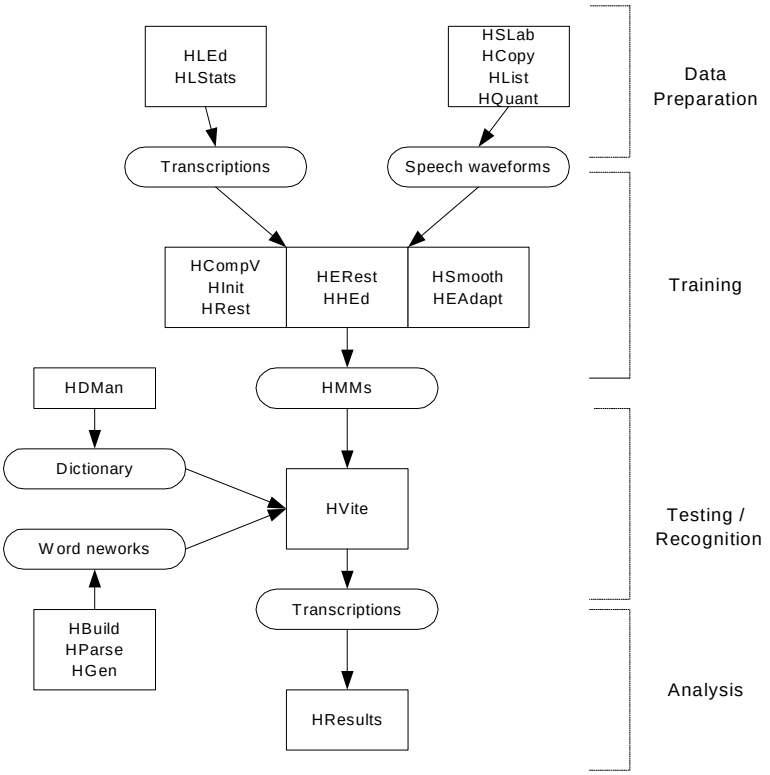


Figure 3.3 -The HTK tools

3.3.1 Generic Properties of a HTK Tool

As was said in the introduction HTK tools are designed to run with a traditional command-line style interface. The layout of the commands is the same for all tools. Each tool has a number of required arguments plus optional arguments. The latter are always prefixed by a minus sign. As an example, the following command would invoke the mythical HTK tool called HFoo

```
HFoo -T 1 -f 34.3 -a -s myfile file1 file2
```

This tool has two main arguments called file1 and file2 plus four optional arguments. Options are always introduced by a single letter option name followed where appropriate by the option value. The option value is always separated from the option name by a space. Thus in the example, the value of the -f option is a real number, the value of the -T option is an integer number and the value of the -s option is a string. The -a option has no following value and it is used as a simple flag to enable or disable some feature of the tool. Options whose names are a capital letter have the same meaning across all tools. For example, the -T option is always used to control the trace output of a HTK tool.

In addition to command line arguments, the operation of a tool can be controlled by parameters stored in a configuration file. Configuration files are indicated by the -C option. For example, if the command

```
HFoo -C config -f 34.3 -a -s myfile file1 file2
```

is executed, the tool HFoo will load the parameters stored in the configuration file config during its initialization procedures. Configuration parameters can sometimes be used as an alternative to using command line arguments. Most tools, especially editing tools, like HHed which edits Hidden Markov models, or HLEd which edits transcription files also need a script that tells the tool which steps to perform and how to edit the data provided to it. The available scripting commands are tool specific.

3.3.2 Data Preparation Tools

HSLab is an interactive label editor for manipulating speech label files. It can be used both to record the speech and to manually annotate it with any required transcriptions. An example of using HSLab would be to load a sampled waveform file, determine the boundaries of the speech units of interest and assign labels to them. Alternatively, an existing label file can be loaded and edited by changing current label boundaries, deleting and creating new labels. HSLab is the only tool in the HTK package, which provides a graphical user interface.
Although all HTK tools can parameterize waveforms on-the-fly, in practice it is usually better to parameterize the data just once. The tool HCopy is used for this. As the name suggests, HCopy is used to copy one or more source files to an output file. Normally, HCopy copies the whole file, but a variety of mechanisms are provided for extracting segments of files and concatenating files. By setting the appropriate configuration variables, all input files can be converted to parametric form as they are read-in. Thus, simply copying each file in this manner performs the required encoding.
The tool HList can be used to check the contents of any speech file and since it can also convert input on-the-fly, it can be used to check the results of any conversions before processing large quantities of data.
HLEd is a script-driven label editor which is designed to make transformations to label files, like translating word level label files to phone level label files, merging labels or creating triphone labels. HLEd can also output files to a single Master Label File MLF, which is usually more convenient for subsequent processing.
HLStats can gather and display statistics on label files and where required, HQuant can be used to build a VQ codebook in preparation for building discrete probability HMM system.

3.3.3 Training Tools

HTK allows HMMs to be built with any desired topology. HMM definitions can be stored externally as simple text files and hence it is possible to edit them with any convenient text editor. With the exception of the transition probabilities, all of the HMM parameters given in the prototype definition are ignored. The purpose of the prototype definition is only to specify the overall characteristics and topology of the HMM. The actual parameters will be computed later by the training tools. Sensible values for the transition probabilities must be given but the training process is very insensitive to these. An acceptable and simple strategy for choosing these probabilities is to make all of the transitions out of any state equally likely.

If segmented transcriptions are available the tools HInit and HRest provide isolated word style training using the fully labeled data as bootstrap data.
HInit can be used to provide initial estimates of whole word models in which case the observation sequences are realizations of the corresponding vocabulary word. Alternatively, HInit can be used to generate initial estimates of seed HMMs for sub-unit based speech recognition. In this latter case, the observation sequences will consist of segments of continuously spoken training material. HInit will cut these out of the training data automatically by simply giving it a segment label. In both of the above applications, HInit normally takes as input a prototype HMM definition, which defines the required HMM topology i.e. it has the form of the required HMM except that means, variances and mixture weights are ignored. The transition matrix of the prototype specifies both the allowed transitions and their initial probabilities. Transitions, which are assigned zero probability, will remain zero and hence denote non-allowed transitions. HInit estimates transition probabilities by counting the number of times each state is visited during the alignment process.
HRest performs basic Baum-Welch re-estimation of the parameters of a single HMM using a set of observation sequences. HRest can be used for normal isolated word training in which the observation sequences are realizations of the corresponding vocabulary word or it can be used for isolated model training for sub-unit based speech recognition. In this latter case, the observation sequences will consist of segments of continuously spoken training material. HRest will cut these out of the training data automatically by simply giving it a segment label. In both of the above applications, HRest is intended to operate on HMMs with initial parameter values estimated by HInit.
HERest is used to perform a single re-estimation of the parameters of a set of HMMs using an embedded training version of the Baum-Welch algorithm. Training data consists of one or more utterances each of which has a transcription in the form of a standard label file (segment boundaries are ignored). For each training utterance, a composite model is effectively synthesized by concatenating the phoneme models given by the transcription. Each phone model has the same set of accumulators allocated to it as are used in HRest but in HERest they are updated simultaneously by performing a standard Baum-Welch pass over each training utterance using the composite model.
The tool HHEd is a HMM definition editor which will clone models into context-dependent sets, apply a variety of parameter tyings and increment the number of mixture components in specified distributions. To improve performance for specific speakers the tools HEAdapt and HVite can be used to adapt HMMs to better model the characteristics of particular speakers using a small amount of training or adaptation data.

3.3.4 Recognition Tools

HTK provides a single recognition tool called HVite, which uses a token passing algorithm like the one described in the previous chapter to perform Viterbi-based speech recognition. HVite takes as input a network describing the allowable word sequences, a dictionary defining how each word is pronounced and a set of HMMs. It operates by converting the word network to a phone

network and then attaching the appropriate HMM definition to each phone instance. Recognition can then be performed on either a list of stored speech files or on direct audio input. HVite can support cross-word triphones and it can run with multiple tokens to generate lattices containing multiple hypotheses. It can also be configured to rescore lattices and perform forced alignments.
The word networks needed to drive HVite are stored using the HTK standard lattice format. This is a text-based format and hence word networks can be created directly using a text-editor. However, this is rather tedious and hence HTK provides two tools to assist in creating word networks. Firstly, HBuild allows sub-networks to be created and used within higher level networks. Hence, although the same low level notation is used, much duplication is avoided. Also, HBuild can be used to generate word loops and it can also read in a backed-off bigram language model and modify the word loop transitions to incorporate the bigram probabilities.
As an alternative to specifying a word network directly, a higher level grammar notation can be used. This notation is based on the Extended Backus Naur Form (EBNF) used in compiler specification The tool HParse is supplied to convert this notation into the equivalent word network. Whichever method is chosen to generate a word network, it is useful to be able to see examples of the language that it defines. The tool HSGen is provided to do this. It takes as input a network and then randomly traverses the network outputting word strings. These strings can then be inspected to ensure that they correspond to what is required. HSGen can also compute the empirical perplexity of the task.
Finally, the construction of large dictionaries can involve merging several sources and performing a variety of transformations on each sources. The dictionary management tool HDMan is supplied to assist with this process.

3.3.5 Analysis Tool

A tool called HResults compares recognition results with original transcriptions. It uses dynamic programming to align the two transcriptions and then counts substitution, deletion and insertion errors. Options are provided to ensure that the algorithms and output formats used by HResults are compatible with those used by the US National Institute of Standards and Technology (NIST). As well as global performance measures, HResults can also provide speaker-by-speaker breakdowns, confusion matrices and time-aligned transcriptions. For word spotting applications, it can also compute Figure of Merit (FOM) scores and Receiver Operating Curve (ROC) information.

4 DEVELOPMENT OF A SPEECH RECOGNIZER

This chapter describes the design and construction of a speech recognizer for the Dutch language. First the requirements for the system are stated and the development platform and tools are described. Next an outline of the steps needed to build a speech recognizer is given. Subsequent sections explain these steps in more detail. Whenever appropriate some background theory is explained, however, the emphasis here is on the differences between practice and the theory described in chapter two. The final sections give an evaluation of the system.

4.1 INTRODUCTION

This chapter describes the design and construction of a speech recognizer for the Dutch language that was built in the Knowledge Based Systems group at Delft University of Technology.
There are many possible types of speech recognizers, ranging from task-specific word recognizers or word spotting systems to large vocabulary recognizers for natural free speech. The system that was built here was intended to be as general as possible, so that it can be used, with slight modifications and some adaptive training, as a speech interface for many different, more specific tasks and applications. An example of such an application is the telebanking system introduced in 2.2.3 that can be used to conduct financial transactions by telephone. Moreover the system should also provide a baseline system for further research on the subject of speech recognition. An example for this category, and the immediate cause for building this system, is the integration of multiple modalities in a Hidden Markov based speech recognizer.
To meet these requirements the recognizer is designed to recognize large vocabulary continuous speech and is speaker independent. To make sure that the systems vocabulary was not limited in any way to some fixed set of words it was decided to create a phoneme based recognizer. This way the system is easily extendible. A different vocabulary only involves a change of language model and possibly adding some words to the pronunciation dictionary.
The final system uses context dependent models, but as this system was built and refined incrementally, actually a whole set of recognizers has been created, ranging from a simple monophone recognizer to a sophisticated multiple mixture triphone system.

4.2 THE DEVELOPMENT ENVIRONMENT

The system was developed on the Microsoft Windows platform using a variety of tools, programs and scripts. The first set of scripts that was written can be described as meta scripts, because they contain the calls to the different tools and provide these tools with their numerous attributes, scripts and with the output obtained from other tools. This approach was taken to make reproduction of the entire process easy. This is not just useful for future attempts to build similar systems, but it also turned out to be a necessity for keeping the process manageable and for recovering from errors.

By far the largest set of tools and software libraries was taken from the HTK toolkit described in the last chapter. The choice of using HTK, rather than implementing the speech recognition algorithms in some general purpose programming language was based on two facts.

Although implementing the basic algorithms, necessary for performing speech recognition, like the Viterbi algorithm, is straightforward, given the formulas in chapter two, it is much harder, and thus more time consuming, to create implementations efficient enough to be useful in practice. As will be seen later on in this chapter training a system using the highly optimized HTK tools still takes vast amounts of time.

HTK offers a nice and flexible data structure for defining and manipulating Hidden Markov models. The main advantage of this structure is that it uses a text based definition language, which allows for manual manipulation of the models or manipulation by custom made tools. A property that proved very valuable, not only during the construction of the speech recognizer, but also in later experiments that were concerned with the integration of models from different modalities in the speech recognizer.

As was described in the last chapter most HTK tools are script-driven. More than 100 of these scripts were created to control the development process. A number of these scripts were created by programs written in C++ that filled the gaps in the development process not covered by one of the tools. This includes a program for data selection, some tools for creating a initial acoustic model set and a tool for creating scripts for triphone clustering. Further explanations of the tools will be given in the next sections, when appropriate.

4.3 Outline of the development process

As was described in Chapter two a Hidden Markov model based speech recognizer consists of three parts.

- A preprocessing part, which extracts relevant features from the speech signal.
- A language model that tells how likely a certain word string is to occur.
- An acoustic model, which computes how a word string is likely to be pronounced.

This subdivision gives some clues on the decisions that have to be made and the steps that have to be performed to build a recognizer. In the case of the preprocessor decisions have to be made concerning the data format of the audio streams and on the type of preprocessor that is to be used. To build a language model a vocabulary has to be chosen and it should be decided what kind of grammatical structure the language model should impose on the utterances.
With regard to the acoustic model decisions have to be made concerning the unit of speech one model represents, the number of states in such a model, the way these states are connected and concerning the type of distribution functions. These acoustic models should then be trained and refined using a set of examples. So a training data set has to be prepared.

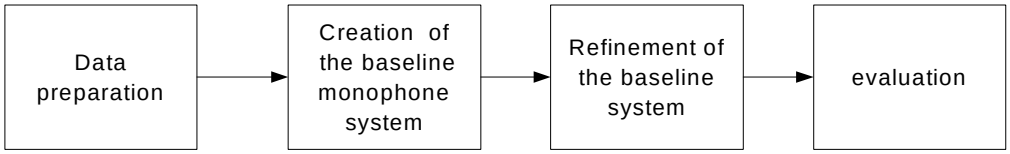


Figure 4.1 - The development process

As shown in figure 4.1 the overall development process can be divided in four stages. In the data preparation stage training data is collected and prepared and the design decisions concerning the models are taken. In the initial training stage, a simple, baseline speech recognition system is build. In the refinement stage this systems is then incrementally refined to make it more advanced and robust. In the final stage the performance of the system is evaluated. The next four sections describe how these four stages were performed to build the speech recognizer.

1.3 DATA PREPARATION

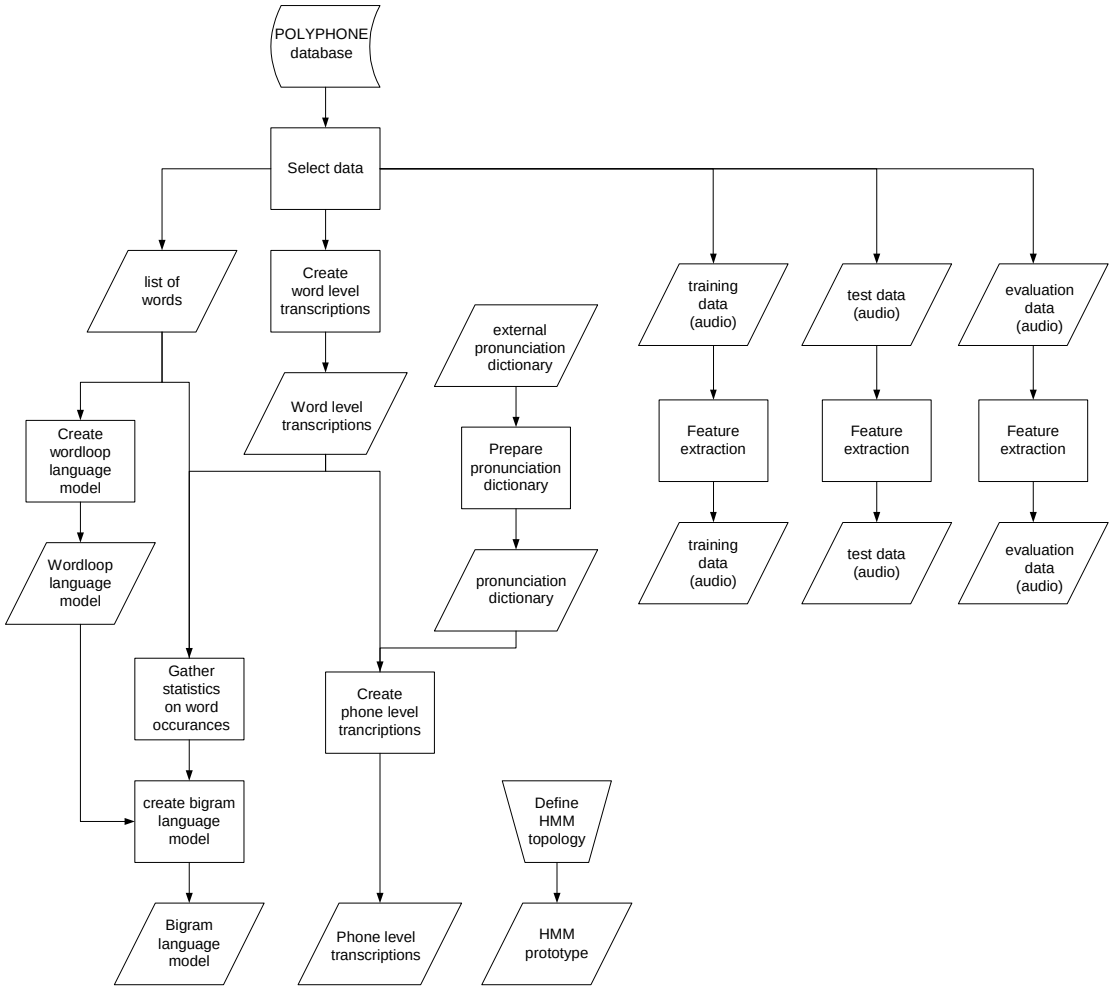


Figure 4.2 - Steps performed during data preparation

The data preparation phase involves a number of inter-related tasks a shown in figure 4.2. Obviously, speech data is needed for training, testing and evaluating the models in the process; therefore the first step involved selection of appropriate speech recordings. For each of these recordings a textual transcription was created. Further a language model was created and a HMM topology for the acoustic models was defined. The exact steps and their relations are now described in detail.

4.3.1 Data selection

In order to build a robust large vocabulary speech recognizer it is crucial that all acoustic (sub-word) models receive enough training examples, taking into account the many variations that can occur in speech, due to, for example, variation in speed of speech, different gender and different accents. It follows that a training corpus should be sufficiently large, consisting of speech samples spoken by men and women from different dialect regions. Since the sound of a sub-word unit is also influenced by its context the speech samples should include all phonemes in as many different phonetic contexts as possible. In practice this means that, depending on the quality of the samples and the complexity of the acoustic models, about ten- to thirty-thousand (phonetically rich) sentences are necessary.
Recording such a data set is a major undertaking involving sub-tasks like selection of participants, recording the data and the most time-consuming parts post-processing and transcribing the data. Fortunately, many of these speech corpora are now commercially available. One of the corpora available for the Dutch language is the Polyphone database, which is used to select the training data for the recognizer as it contains speech samples that fulfill the requirements stated above. It is rather large; it contains telephone speech from 5050 different speakers and 222075 speech files, based on 44 or in some cases 43 items per speaker. The speakers in the database were selected from all geographic regions. The ratio between male and female speakers is almost fifty-fifty. The utterances contain all Dutch phonemes in as many phonetic contexts as the designers of the database could find. But the set also has some disadvantages that all stem from the fact that the Polyphone database was recorded with automatic voice-interactive telephone services in mind. Most speech files therefore contain examples of phrases useful for this kind of applications, this includes street names, bank-accounts, numbers and answers to yes-no questions. Training a large vocabulary recognizer on these samples may result in a recognizer that performs well on recognizing numbers and 'yes' and 'no' but which generalizes very poor to other words. To avoid these problems only nine items per person were used, five phonetically rich sentences and four application sentences. The latter group consists of sentences that contain an application word, that is, a word that is often used in normal speech. Since the polyphone database is recorded over a telephone line many samples are of poor quality, or contain background noise or even background speech. To ensure well-trained models that can be used for recognizing speech using for example a PC microphone spoken by an average person, only sessions that fit the following profile were used:

- Men and women from all regions.
- Only native speakers that did not live abroad (this may be a bit rigorous, as not all foreigners have an accent, and a small accent shouldn't be a problem, but it avoids manually going through thousands of utterances).
- No utterances that contain background noise or background speech. But mouth noises, like smacking, sniffing or loud breaths and verbal hesitations like uh are allowed.
- Only sentences that are assessed to have quality level OK in the polyphone database, which means that the text can be clearly understood, i.e. no stuttering, no disturbing hesitations and no mispronunciations or foreign pronunciations.

Manually selecting utterances that adhered to this profile is practically impossible. The data in the polyphone database is stored on 10 CD-roms each with about 500 directories, called sessions, with one speaker per session. Information concerning the speakers and the utterances is listed in three large plain text format tables. No data selection tools whatsoever are included in the database. To overcome this problem a program was written, called Pdy2Htk, which automatically selects a subset of the polyphone database according to some selection criteria it is provided with. This program provides a command line interface, in a similar fashion as the HTK tools. At this command line a number of options can be specified:

- A subset of the database that should be searched.
- A caller profile: gender, age interval, dialect region, education level, foreign / lived abroad.
- A subset of the utterances in each session.
- A quality profile: Cordless phones allowed, allowed types of noise, allowed levels of quality.
- A number of output options.

The program creates based on the selected options a number of scripts needed by some HTK tools in later steps, which will be described in relevant sections. Furthermore it creates a list of all the words in the selected set and a either file containing all the transcriptions of the selected utterances index by the name of the utterances or one file per utterance with the same name as the utterance but a different file type. The program also creates unique filenames for the selected utterances. Three different data sets were extracted from the Polyphone database this way. A training set, a development set, which was used for testing and fine tuning during development of the system and an evaluation test set to evaluate the final performance of the system. The development set contained persons and sentences that did not occur in the training set. And the evaluation test set contained persons that did neither occur in the training or development set, but its phonetically rich sentences did also occur in the training set.

```
TRAIN - PolyPhone to HTK Summary:

4022 sessions processed
  2883 sessions OK
  1139 sessions skipped

25954 utterances processed
  22626 utterances OK
  1632 utterances skipped because of bad transcriptions
  1689 utterances skipped because of bad sound quality

  7 utterances missing

DEVELOP - PolyPhone to HTK Summary:

500 sessions processed
  340 sessions OK
  160 sessions skipped

3060 utterances processed
  2673 utterances OK
  200 utterances skipped because of bad transcriptions
  187 utterances skipped because of bad sound quality

EVAL - PolyPhone to HTK Summary:

528 sessions processed
  368 sessions OK
  160 sessions skipped

3313 utterances processed
  2885 utterances OK
  198 utterances skipped because of bad transcriptions
```

Figure 4.3 -Results Polyphone 2 HTK

4.3.2 Feature vectors

The audio files on the Polyphone CDs are saved as waveforms. In particular, they are stored in compressed CITT A-law format, the format used in telephone systems. The first step in speech recognition is the extraction of feature vectors. Theoretically this can be done on the fly during training, as is the case during real-time recognition. But for training purposes it makes sense to do this once and for all and save the feature vectors in a file, as each utterance is processed a number of times during training.
Preprocessing of the data files was done in two stages. First, the original waveform files were translated to intermediate audio files in the SPHERE format. This was done by the Praat program, an audio analysis program developed at the University of Amsterdam, which provides extensive scripting possibilities. The scripts that guided this program were created by Pdy2Htk during data selection. This first step was necessary, because the program used in the second step,

the HTK tool HCopy, cannot read the A-law audio format. HCopy is a tool for translating audio files to feature vector files using one of the many variants of either linear predictive coding or Mel scale cepstral coefficients. It takes as its parameters a list summing all the files it has to copy and the new filenames they should be copied to and a script containing information on input and output formats. The filename list was provided also by Pdy2Htk.
The utterances were encoded to Mel-frequency cepstral coeffient vectors. This choice was based on literature and earlier experiments. Each vector contained twelve cepstral coefficients with log energy and delta and acceleration coefficients added, all scaled around zero by subtracting the cepstral mean from all vectors. Which resulted in 39 dimensional feature vectors. A sampling rate of 10 ms was used and a overlapping window of 25 ms.

4.3.3 Phoneme set

As was already mentioned in the introduction, the basic speech units in the system are phonemes. This implies that a phoneme set should be chosen. Unlike for example with the characters in the alphabet, there is no such thing as the phoneme set for a given language. Existing sets differ in the number of details they provide. For example the so-called plosive sounds, like /p/ could be modeled as one sound or as two sounds. Namely a closure /cp/, that is a short period of silence because the mouth is closed before a plosive is uttered and the /p/ sound that is produced as soon as the mouth is opened.
However, there are a few standard phoneme sets, used for example in dictionaries. One of those is the SAMPA notation, which covers all distinguishable phoneme sounds in the Dutch language. It provides 43 different phonemes, which is a fairly reasonable number for the use of a model set for speech recognition.
The phonemes from the SAMPA set were adopted as phoneme set in this project, but they were renamed for convenience in later steps. Table 4.1 gives an overview of the phonemes including an example for each phoneme to show which sound it represents.
Furthermore three other phonemes were added. The first, sil, models (longer periods) of silence, these silences occur between sentences or when a person is not speaking at all. The second phoneme, sp, also represents silence, but only periods of short duration. This is the kind of silence that occurs between words. The reason that a distinction was made between these two types of silence is that they really represent two different phenomena. One represents periods of pure silence (which in this context always means background and environment noise) that can greatly vary in length while the other represents optional periods of silence shorter than the duration of a phoneme. Because these silences mainly occur between words there is a slight influence from the neighboring sounds on this phoneme.
The reader may have noticed that utterances containing mouth noise were not excluded during data selection. This was done on purpose, to enable the creation of the final phone that was added, nn, which models all kinds of mouth noise and verbal hesitation. The idea behind the inclusion of this phone was that real, natural speech always contains mouth noise and modeling this may improve the results in real-life environments.

Table 4.1 - Phone set

SAMPA notation	used notation	example	phonetic transcription
I	i	pit	p i t
E	e	pet	p e t
A	a	pat	p a t
O	o	pot	p o t
Y	y	put	p y t
@	at	gemakkelijk	g at m a k at l at k
i	ie	vier	v ie r
y	yy	vuur	v yy r
u	u	voer	v u r
a:	aa	naam	n aa m
e:	ee	veer	v ee r
2:	eu	deur	d eu r
o:	oo	voor	v oo r
Ei	ei	fijn	f ei n
9y	ui	huis	h ui s
Au	ou	goud	x ou t
E:	eh	crème	k r eh m
9:	euh	freule	f r euh l at
O:	oh	roze	r oh z at
p	p	pak	p a k
b	b	bak	b a k
t	t	tak	t a k
d	d	dak	d a k
k	k	kap	k a p
g	gg	goal	gg oo l
f	f	fel	f e l
v	v	vel	v e l
s	s	sein	s ei n
z	z	zijn	z ei n
x	x	toch	t o x
G	g	goed	g u t (also: x u t)
h	h	hand	h a n t
Z	zj	bagage	b a g aa z at
S	sh	show	sh oo u
m	m	met	m e t
n	n	net	n e t
N	nn	bang	b a nn
l	l	land	l a n t
r	r	rand	r a n t
w	w	wit	w i t
j	j	ja	j aa

4.3.4 Transcriptions and pronunciation dictionary

To monitor the progress that is made during training and to evaluate the performance of the final system textual transcriptions of what was really said in the test and evaluation utterances were

needed. As was already mentioned these were created by Pdy2Htk. But there is more to it, the acoustic models represent phones, so in the training phase examples of phones will be needed to adjust the parameters of these models. But the examples in the set of training files constitute complete sentences, that is, a string of phones. To ensure that the right part of an utterance is used to train a model a transcription of the utterance at the phone level is needed. These phone level transcriptions can be created from the word level transcriptions by using a pronunciation dictionary that gives for each word the sequence of phones that make up the pronunciation of this word. This is exactly the way it was done during this project. First of all a pronunciation dictionary had to be prepared. With the polyphone database comes a dictionary that includes almost all words used in the polyphone utterances. This dictionary was used as a starting point for the pronunciation dictionary. Some changes had to be made to the format of the dictionary because the HTK transcription editor HLEd needs a particular form of dictionary.
First all characters were transformed to lowercase and punctuation symbols (\" and \') were changed and in some cases removed because HTK has difficulty with transcriptions containing certain non-alphanumeric characters. This resulted in some strange words like 'patient', 'reeel', 'savonds' but this is not really a problem, as there is no good reason why the spelling internal to the system should be the proper spelling (as it is output to a human reader). Next stress-markers and syllable information were removed from the dictionary and the phonetic transcriptions were transformed to the right format, using the new phone names.
Having a phone set, a dictionary and word level transcription files the phone level transcriptions could be prepared. It turned out that the word level transcriptions contained words that did not occur in the dictionary. Words, for which the transcription was clear, because they were similar to word already in the dictionary, were added to the dictionary, otherwise the transcriptions and corresponding utterances were removed from the data set (in total 7 items were removed).
Using the word level label files the phone set and the dictionary, the phone level label files were created, using the HLed tool. This tool takes a list of all label files that are to be processed, a dictionary and a script containing the commands that should be executed as its arguments. It this particular case it substituted each word in the label files with the first phonetic transcription it found for this word in the dictionary. The phone level transcriptions obtained this way were unsegmented, that is they did not contain any information on the time intervals at which a phoneme is spoken. This means that there is no way to know exactly which part of an utterance represents a certain phone and should be used to train this phones HMM. This makes model training a little harder, than when segmented data would be available, but not impossible as will be described in the sections on training.

4.3.5 Language models

The language model is a critical part of a speech recognition system. It tells the system how likely it is that a certain string of words is uttered. By doing so, it imposes a grammar onto the system. Furthermore, the language model is the glue that holds together the set of acoustic models in the word network that is ultimately used for recognition. In this project a number of language models at various levels of sophistication was created for each of the data sets.

First simple wordlist grammars were generated, using the wordlists provided by Pdy2Htk. These word networks are not really grammars at all, since every word is equally likely and a sentence is just a string of these words. In order to put some grammatical constraints on the language model, the grammars were transformed to backed-off bigram language models. Using the word level transcriptions and statistics on the number of occurrences of each word and of each word combination were calculated according to the following formulas:

$$p(i,\ j) = \begin{cases} (N(i,\ j) - D) / N(i) & if\ N(i,\ j) > t \\ b(i)\, p(\ j) & otherwise \end{cases} \tag{4.1}$$

Where N(i,j) is the number of times word j follows word i and N(i) is the number of times that word i appears. D is the so-called discount constant, which is used to deduct a small part of the available probability mass from the higher bigram counts to and distribute it amongst the infrequent bigrams.
When a bigram count falls below the threshold t, the bigram is backed-off to the unigram probability suitably scaled by a backed-off weight b(i) in order to ensure that all bigram probabilities for a given history sum to one. The unigram probability is computed using:

$$p(i) = \begin{cases} N(i) / N & if\ N(i) > u \\ u / N & otherwise \end{cases} \tag{4.2}$$

Where u is a constant called unigram floor count and N is the total number of words:

$$N = \sum_{i=1}^{L} \max(N(i), u) \tag{4.3}$$

The backed-off weight follows from:

$$b(i) = \frac{1 - \sum_{j \in B} p(i,\ j)}{1 - \sum_{j \in B} p(i)} \tag{4.4}$$

In which B is the set of all words for which p(i,j) has a bigram.
These statistics were then used to create backed-off bigram language models for the training, test and evaluation sets, using the HDMan tools which translated the gathered statistics into HTK Standard Lattice Format, that are used for storing word models and multiple hypotheses from the output of a speech recognizer. For testing purposes a simple phoneme language model was also created.

4.3.6 HMM prototype

The last step in data preparation was the selection of a Hidden Markov Model topology for the acoustic models. Since a phoneme based recognizer was build a model represents a phoneme. A topology consisting of non-emitting start and end states and three emitting states, using single

Gaussian density functions, was chosen. The states were connected in a left-to-right way, with no skip transitions. The model is shown in figure 4.4

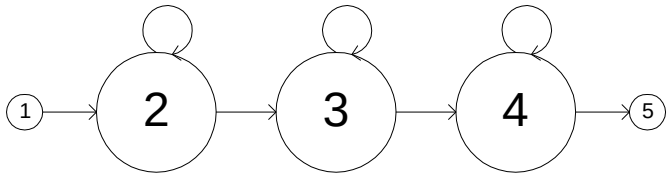


Figure 4.4 - The acoustic HMM topology

This is a rather simple model topology, but earlier experiments with three and five state models showed that the three state model performs as well, and in some cases even slightly better than the five state model, despite the fact that is has fewer parameters. These experiments showed that it works best to start with a simple model with only a few parameters and then increasing the complexity of the model as training progresses. So this is the approach that was also taken during this project.

4.4 TRAINING

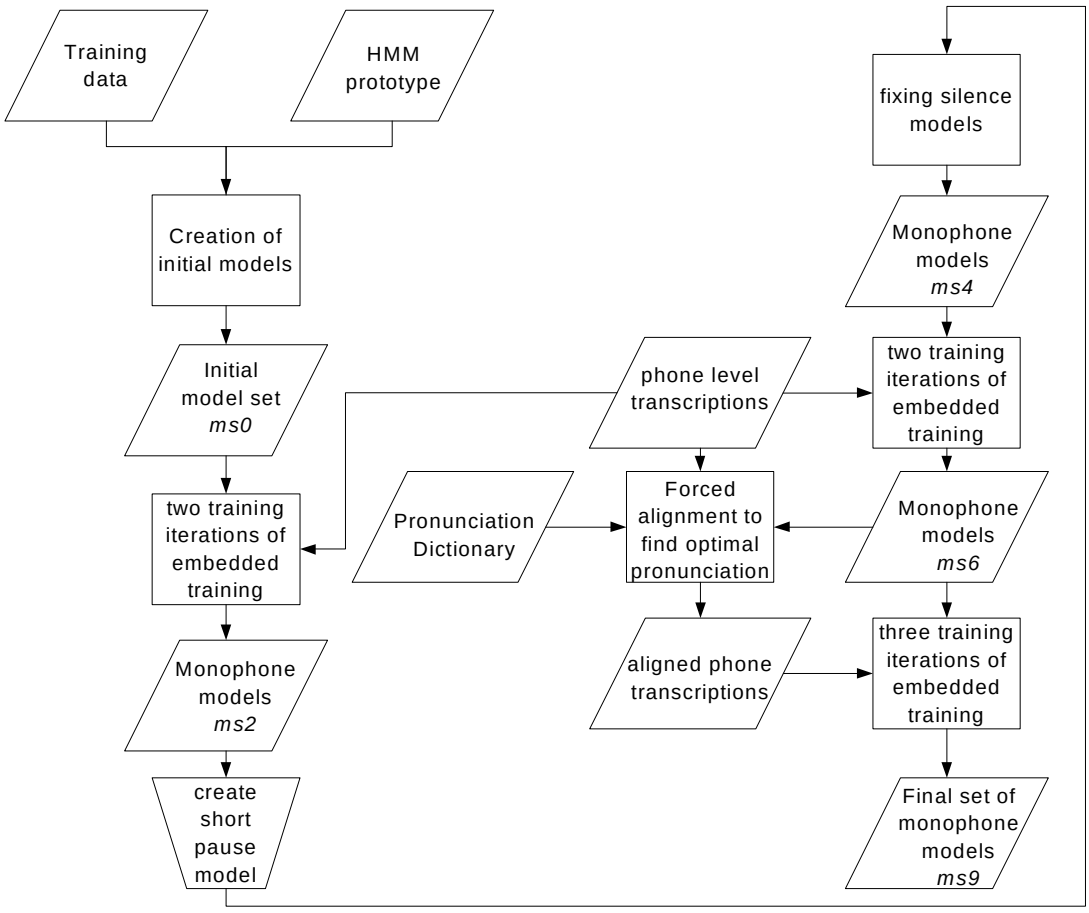


Figure 4.5 - Steps performed during training

With all data sources in place the actual training of the acoustic Hidden Markov Models could start. Figure 4.5 shows the steps that were taken to train a monophone speech recogizer. First, a set of acoustic models was created and trained. Next the system was made more robust by adding sophisticated silence models. Thereupon the system was trained a little more and Viterbi alignment was used to find the most likely transcriptions. These transcriptions were then used to train the monophone system. These steps will now be described in detail.

4.4.1 Initial models

As explained in chapter 2.5 training of HMMs can be done using the Baum-Welch algorithm, but to ensure proper and fast convergence of the models sensible initial values have to be calculated for the transitions parameters and the means and variances of the state density functions before the Baum-Welch algorithm can be used. HMMs are pretty sensitive to initial values so this stage is crucial for the entire process.
Good initial models can be obtained by using the concept of a HMM as a generator of speech vectors. The training examples of the phones corresponding to the model whose parameters are to be estimated can be viewed as the output of this model. Thus if the state that generated each vector in the training data was know, then the unknown means and variances could be estimated by averaging all the vectors associated with each state.
This principle can be implemented by using the Viterbi algorithm in an iterative scheme. In every iteration the Viterbi algorithm is used to find the most likely state sequence corresponding to each training example, then the HMM parameters are re-estimated. This process is repeated until no further increase in the log likelihood calculated by the Viterbi algorithm is obtained.
Drawback of this algorithm is that it requires segmented transcription data to collect all examples corresponding to a certain model. As mentioned before, in the case of the Polyphone data segmented labels were not available, therefore an alternative initialization technique was used, called the flat start scheme. Which comes down to an uniform segmentation of the data, by computing the global mean and variance for each feature and setting all the Gaussians of all the models to have this mean and variance. For this purpose a tool called HCompVSet was written. This tool uses code from the HTK tool HcompV to compute the global mean and variance over the entire data set. Subsequently it creates definitions for each model in the set.
Apart from the model set a variance floor vector was created which was equal to 0.01 times the global variance. This vector was used to set a floor on the variances estimated in subsequent steps. If a variance might fall below this floor it will be set equal to the floor variance.

4.4.2 Embedded re-estimation

Once the initial model set is available the re-estimation of the model parameters can start. Standard procedure would be to process each model in turn by selecting the training examples corresponding to the model and using the Baum-Welch re-estimation algorithm to update the models. The HTK tool HREst takes this approach. But this algorithm also requires segmented labels to locate the examples corresponding to a model and thus could not be used in the case of

the polyphone recognizer. And actually, it was not even desirable to use the standard algorithm in this case, since the recognizer was supposed to recognize continuous speech and not isolated words or phonemes. Therefore an embedded re-estimation strategy was used, that simultaneously updated all of the HMMs in the system using all of the training data. In the embedded training algorithm each training utterance is processed in turn, the associated transcription is used to construct a composite HMM which spans the whole utterance. This composite HMM is made by concatenating instances of the phone HMMs corresponding to each label in the transcription. The Forward-Backward algorithm is then applied and the sums needed to form the weighted averages are accumulated. When all of the training files have been processed, the new parameter estimates are formed from the weighted sums and the updated HMM set is created.
This embedded procedure is implemented in the HTK tool HERest which performs exactly one iteration of the algorithm each time it is ran, in contrast to the standard procedure which iterates until convergence is reached. In the case of large data sets a single iteration of embedded training may take several hours to compute. Therefore, HERest provides many optimizations to speed up the training process. It is capable of pruning the A and B matrices using a beam search-like approach. By this means, a factor of 3 to 5 speed improvement and a similar reduction in memory requirements can be achieved. Furthermore HERest includes features to allow parallel operation where a network of processors is available. The training set can be split into separate chunks that are processed in parallel on multiple processors.
For the initial models created in the last step the re-estimation process was performed twice. Pruning was not enabled during any of the re-estimation cycles in the entire process to ensure that all models were trained as optimal as possible given the data. But, although only one processor was used, the data set was split in eight subsets of about 2500 utterances each. This way the available processor time could better be utilized, as smaller continuous time spans were needed. Inspection of the source code of HERest revealed that it loaded its, rather large, scripts completely into memory. By splitting up the data sets the scripts were also split up in a number of smaller scripts. This reduced the memory usage of HERest and the time taken to load data in to memory, making it considerably faster.

4.4.3 Fixing the silence models

After the models received a little training and roughly started to take shape the system was adapted to make it more robust. One of the models created by HCompV Set, is the silence model sil. This model thus has the same topology as the other phones. But it is supposed to take care of periods of silence that can vary greatly in length, from a few milliseconds up to a few seconds, that is. Therefore a transition was added from the second state to the fourth and back from the fourth state back to the second. This had to be done after the system received some training during re-estimation, because otherwise the sil model might have absorbed a large part of the utterance.

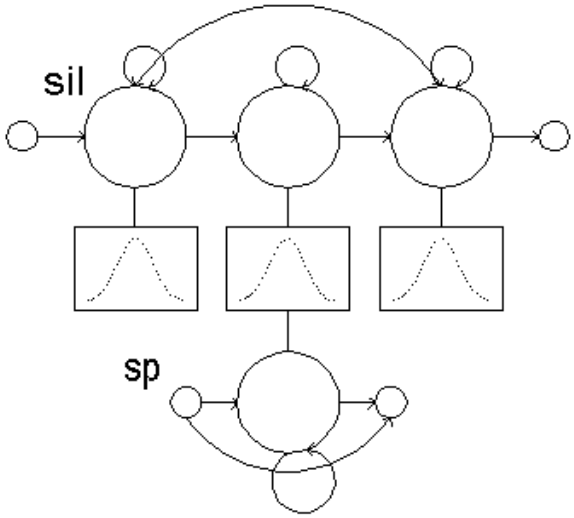


Figure 4.6 - The sp model is tied to the middle state of the sil model

Training sofar was done without the short pause model sp. This was done because of the special nature of the model. This model is meant to take care of optional silences between words, to make sure that these periods of silence are not assigned to other models during training, which would compromise these models. But if it were included right from the start in the training process exactly the opposite might happen. In some cases there may be no or not enough silence between words, then the sp model would pick up some vectors belonging to its neighboring phones. Since the sp model has a very short duration and practically all combinations of phones can occur as its neighbors. This may result in a very poor model that might suppress the performance of the whole system, as it is one of the most frequently occurring models. So in this particular case it is beneficial to explicitly define what type of data this model is supposed to represent. A nice way to do this, is to tell the system that the sp model resembles the silence model sil. This was realized by creating a one state sp model, which has a direct transition from entry to exit node, a so called tee-model. Its one and only state was then tied to the middle state of the silence model, so these two states now shared the same set of parameters, as shown in figure 4.6. After fixing the silence models two re-estimation cycles were performed.

4.4.4 Alignment

Many words, particularly function words, have more than one pronunciation. In the construction of the phone level label files in the data preparation phase the first pronunciation encountered in the dictionary for each word was chosen. This was not necessarily the right pronunciation. To fix this problem the models created so far were used to create new transcriptions using pronunciations that fitted the acoustic data embodied in the models. This was done by performing Viterbi alignment.
For each utterance a network including all alternative pronunciations in parallel was constructed, using the corresponding word level label file and the dictionary. The Viterbi decoder, implemented in the tool HVite then searched and segmented the best matching path through the

network and constructed a lattice which included model alignment information. This lattice/path was then converted to a new segmented phone level transcription. Subsequently another two passes of the re-estimation procedure were performed.

4.4.5 Monophone system

Another final re-estimation cycle resulted in a set of models sufficiently trained to pass for a simple speech recognizer. To check the performance of this system the Viterbi decoding algorithm was used, implemented by the tool HVite, which uses a token passing approach similar to the one described in chapter two, to perform recognition on the development test set.
The transcriptions output by the Viterbi algorithm were compared to the original word level transcription files using the HResult analysis tool, which uses a dynamic programming-based string alignment procedure that is fully interchangeable with the one used in the standard US NIST scoring package. The analysis tool computes the percentage of words correctly recognized as

$$Correct = \frac{H}{N} \times 100\% \tag{4.5}$$

In which H is the number of labels recognized correct and N is the total number of labels.
The word accuracy, which takes into account the fact that some of the words classified as correct may be in fact insertion errors, is computed by

$$Accuracy = \frac{H - I}{N} \times 100\% \tag{4.6}$$

where I is the number of insertion errors.
In this test and in all other tests conducted during development described below, the first half of the development test set was used. This comprises 240 utterances spoken by about 100 different persons. The subset contained 1060 different words, a bigram language model containing these words was used in these tests.

Table 4.2 - recognition results monophone system

System	Percentage of words recognized	Word accuracy percentage
Initial model set after re-estimation (Ms2)	17.15%	-77.17%
System with fixed silence models (Ms6)	30.00%	-48.75
Final monophone system (Ms9)	38.34%	-28.42%

Table 4.2 shows the recognition results from the monophone system. To show the progress that has been made during the various steps the results from earlier steps are also included. Although the improvements made were considerable, the final systems recognized twice as much words as the initial system and the word accuracy has improved in a similar fashion, the overall results were very modest. The accuracy was even negative. Slight improvements could have been

realized by further re-estimation cycles, but the bottleneck of this system is that it is a rather simple continuous speech recognizer. To obtain significant improvement in performance more advanced techniques to refine the system were necessary. These refinements are the subject of the next section.

4.5 REFINEMENT

The monophone system has a number of shortcomings. In the first place, all of the models have the same shape. Further, the system does not take into account linguistic effects like coarticulation, since it uses phoneme units, that are too fine grained to model these effects. Finally, the system does not make up for possible unbalances in the training data, some models may receive much training, while others may receive only little training because they only have a small number of examples. There are a number of ways to refine a speech recognizer system: mixture component splitting, HMM cloning, generalized parameter tying and data driven or tree based clustering. To improve the system a combination of several of these techniques was applied.

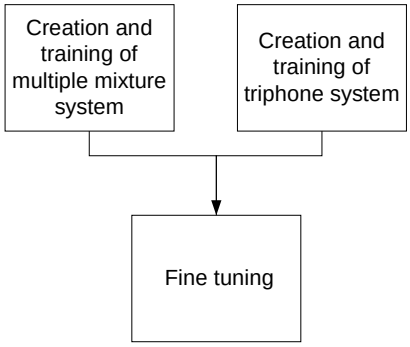


Figure 4.7 - Steps performed to refine the system

Figure 4.7 shows the sub-phases that were performed in this phase. Multiple mixture models were created to make up for model inaccuracies. Triphone models were created to better model coarticulation. These model sets were merged and the resulting model was fine-tuned to find the optimal combination of acoustic models and language model.

4.5.1 Multiple Mixtures

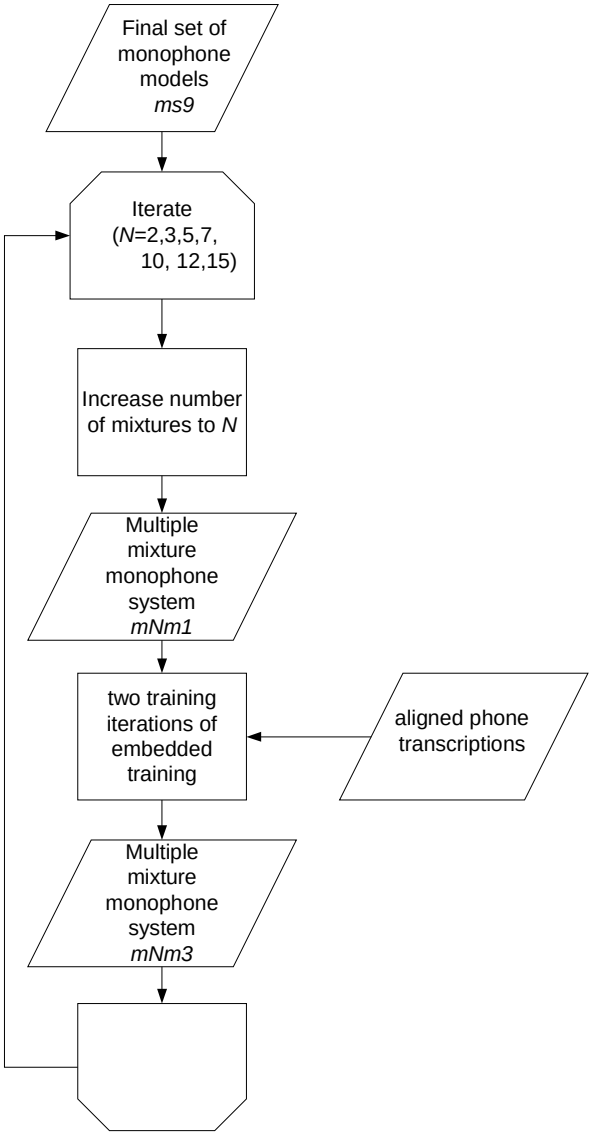


Figure 4.8 - Steps performed to create muliple mixture models

The monophone system used Gaussian distributions to model the 'observation functions', but as a matter of fact these do not adequately reflect the speech process. They are not capable of modeling the variations in speech that are due to, for example, the difference in pitch between male and female voices. To overcome this problem more realistic density functions have to be found. Alternatively this can be achieved by using a large number of states based on simple densities like the Gaussian density. The latter option implies that an appropriate topology for the phone models has to be found, which is not an easy task. The first option can be achieved by a range of techniques from multivariate statistics, but an interesting solution lies in the use of Gaussian mixture densities, which can approximate any continuous probability density function in the sense of minimizing the error between two density functions. The Gaussian mixture

density function is composed by taking the superposition of a number of Gaussian densities each with its own mean and variation and its own mixture weight, as shown in figure 4.9.

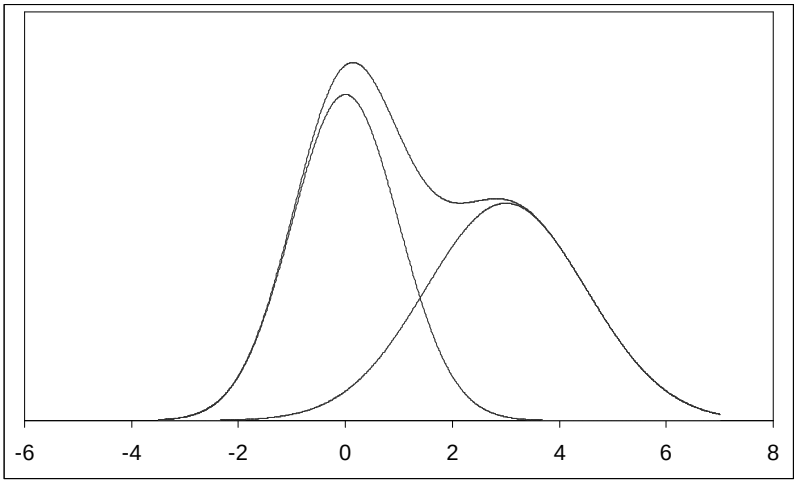


Figure 4.9 - a multiple mixture Gaussian

The observation probabilities are thus computed by

$$b_j(o_t) = \sum_{m=1}^{M} c_{jm} \mathcal{N}(o_t, \mu_{jm}, \mathbf{U}_{jm}) \tag{4.7}$$

The big advantage of this approach is that with only some slight modifications the Baum-Welch algorithm can also estimate the mixture component weights c_{jm}, because each M-component Gaussian mixture state can be viewed as a set of single Gaussian substates. Each with its ingoing transitions weighted by the corresponding mixture weight and its outgoing transition parameter equal to one, as can be seen in the figure below.

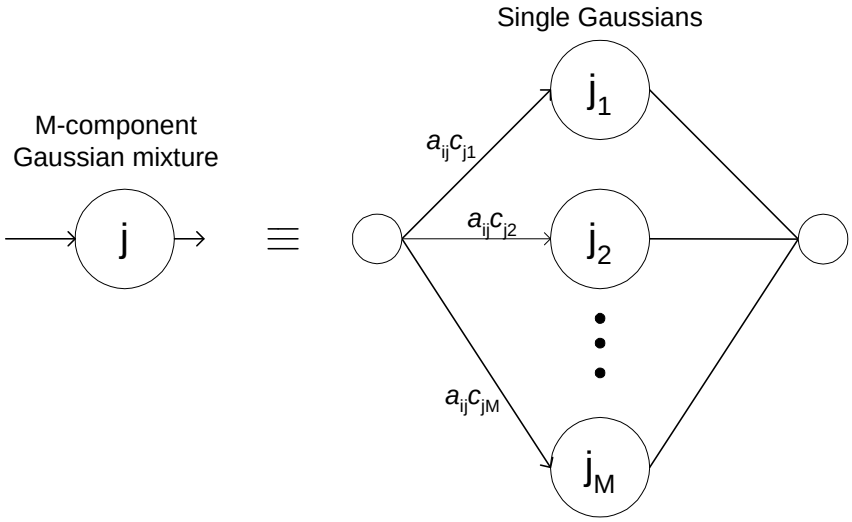


Figure 4.10 - A Gaussian mixture represented as a set of single Gaussians

Different models and different states no longer have to have the same type of distribution. Therefore multiple mixture systems may improve recognition results considerably. However, mixture incrementing is not without dangers, the more mixtures are used the better the models fit to the training data, in the end this may result in models that overfit the training data and generalize poor to other data. During mixture incrementing some component weights may become very small, resulting in defunct mixture components. Defunct mixtures often indicate that not enough training data is available to further increase the mixtures of a model.
So the best strategy to adopt here is to increment the mixture components in stages, by incrementing by one or two mixtures a time, then re-estimating, checking recognition results on a test data set and incrementing the mixtures again until the optimum is found.

In figure 4.8 the steps that were performed to build a multiple mixture monophone system are shown. Taking the single Gaussian monophone system from the last section as a basis the mixtures were incremented in seven steps until a 15-mixture system was obtained. This was done by a script for the HMM editor HHEd. The script processes each state of the all the models one by one, for the distribution in a state it repeatedly splits the mixture with the largest weight until the required number of components is obtained. Splitting was performed by copying the mixture and dividing the weights of both copies by 2. The means were offset by plus or minus 0.2 times the variance to prevent that one component would be floored while the other would remain as before. To prevent defunct mixtures a floor mixture weight was defined, mixture weights were not allowed to fall below this floor. But during the first steps it turned out that the number of floored mixtures increased rapidly. To reduce this number and prevent overfitting the data the minimum number of examples necessary to allow for mixture incrementing of a model was also incremented in each step.

Table 4.3 - Monophone mixture systems

Number of mixtures	Percentage of words recognized	Word accuracy percentage
2	40.11%	-25.42%
3	42.70%	-16.33%
5	45.74%	-7.57%
7	47.92%	-1.81%
10	51.54%	4.77%
12	54.50%	9.54%
15	56.81%	15.51%

4.5.2 Context dependent models (triphones)

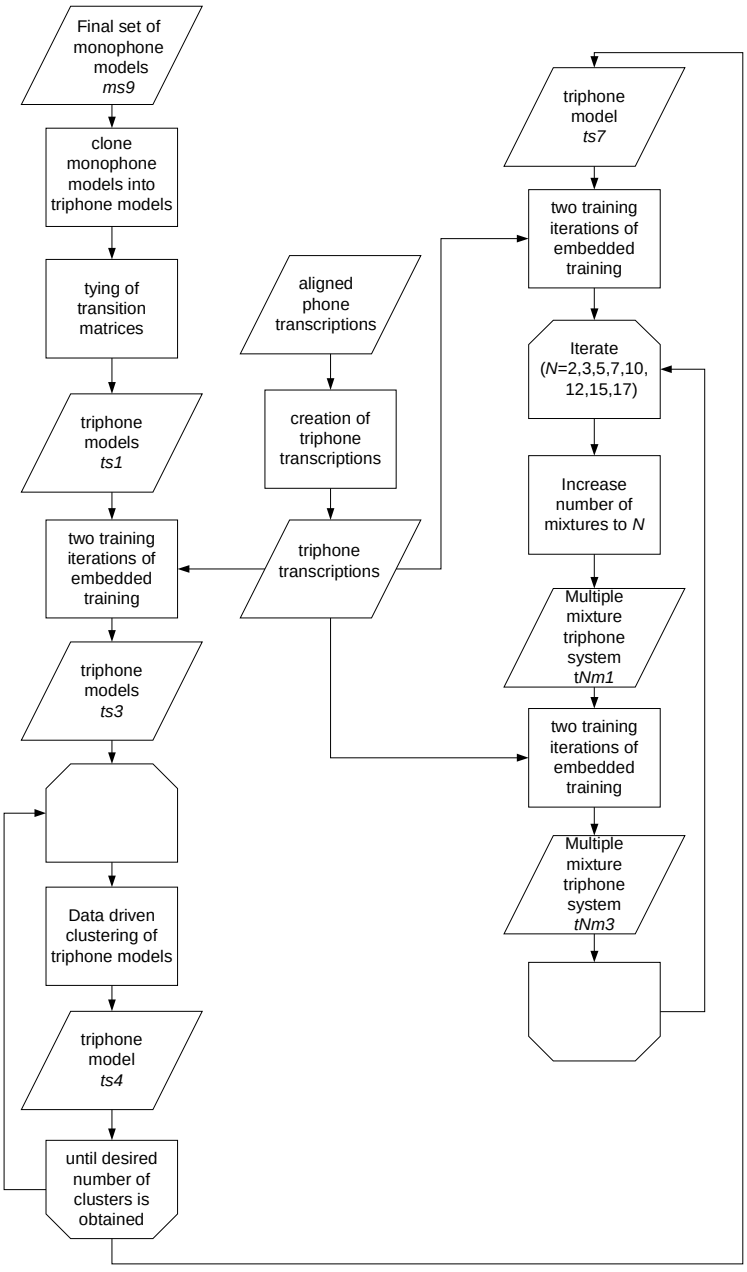
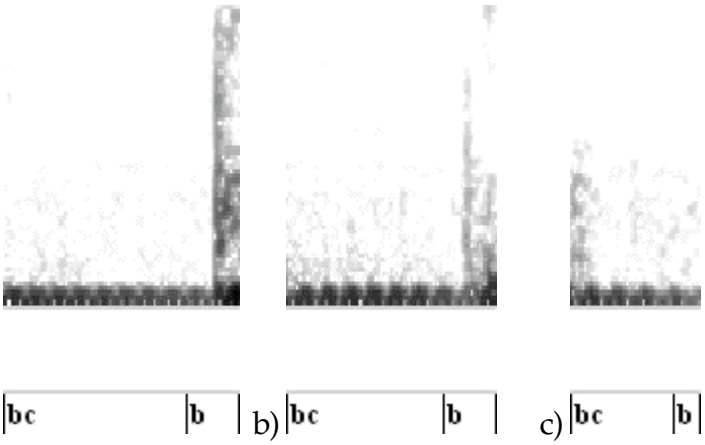


Figure 4.11 - The creation of multiple mixture triphone models

The phone models described in the systems so far did not depend in any way on their context. A phoneme is assumed to sound more or less the same in every situation. Actually, this is not the case since in normal speech, articulations are made quickly and consecutively and therefore modified by neighboring articulations, as can be seen in figure 4.12.



Figuur 4.12 - The phoneme **b** and corresponding closure **bc** in words (a) big, (b) rubber and (c) tube

To capture these effects, called coarticulations, models are needed that take into account the context of a phone. There are many ways to model coarticulations, like for example, modeling all context dependent phones, called allophones by using linguistic theories. But most present-day recognizers have settled on using triphones. Triphones model the context by taking into consideration the left and right neighboring phones. If two phones have the same identity but different left or right context they are considered as different triphones.
Before building a set of context dependent models it is necessary to decide whether cross-word triphones or word internal triphones are to be used. Cross-word triphones are more powerful, since they also take into account coarticulation between words, but because of the large number of possible different cross-word triphones they require large amounts of training data. There are far less word internal triphones, as words are not made up from random phone combinations, so their training data requirements are more modest, but still huge.
In this project it was decided to use word internal triphones. First triphone transcriptions were created that were needed to train the triphone system. This was done by translating the aligned transcriptions created in section 4.4.4 and by replacing the start and end phone in a word by a bi-phone and all other phones by triphones. The sp, sil and mn phones remained monophones. The triphone labels were of the form:

<left context>-<phone name>-<right context>

For example the transcription of the word `het` became:

h+e h-e+t e-t.

As a side effect of this process a list containing all triphones was created. This list was used to create a script for the HMM editor HHEd, that cloned each HMM as often as possible and renamed it to a triphone. So, at the end of this process all triphones were exact copies of the corresponding monophone.

This system contained 8570 triphones, each with 3 states and transition matrix and state distributions belonging to it, the number of parameters in this system is thus enormous. To deal with this problem the number of parameters in the system had to be reduced, to find the right balance between compactness and acoustic accuracy of the individual models.
Parameter tying can do this, when two or more parameter sets are tied, all the owners of the tied set share the same set of parameter values. Figure 4.6 already showed how the states of two silence models were tied.

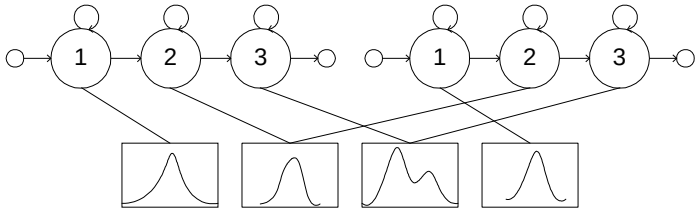


Figure 4.13 - Two state tied HMMs

Figure 4.13 shows two HMMs that share the same distribution functions. During training the parameters of the distribution function attached to the second state of both models will be updated using examples from the first and the second model. Thus there will be more training data available than there would be for two separate distributions. As a consequence however the models will of course become more similar as they will only differ in the distributions of their first state and the transition parameters. The figure below shows at which point models can be tied.

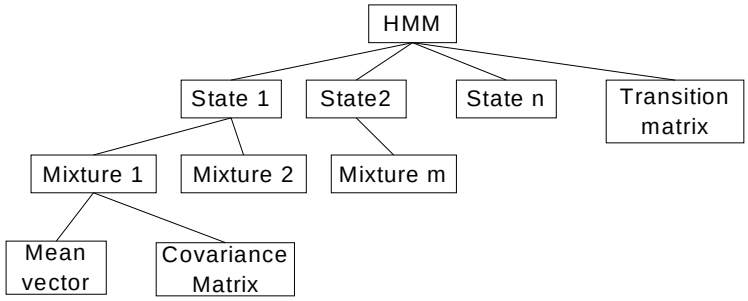


Figure 4.14 - Potential tie points

For context dependent models it can be beneficial to share one transition matrix across all variants of a phone rather than having a distinct transition matrix for each. And applying the argument that context will not greatly affect the center state of triphone models, one way to reduce the total number of parameters without significantly altering the models' ability to represent the different contextual effects might be to tie all of the center states across all models derived from the same monophone.
Although these explicit tyings can have some positive effect they are not very satisfactory. Tying all center states is too severe and the problem of undertraining for the left and right states remains. Therefore most triphone recognizers use a clustering technique to decide which states to tie. Basically there are two mechanisms: Data driven clustering and Tree-based clustering.

Data driven clustering uses a top-down hierarchical clustering procedure: Initially all states are placed in individual clusters. The pair of clusters which when combined would form the smallest resultant cluster are merged. This process repeats until the size of the largest cluster reaches a certain threshold. The size of the clusters is defined as the greatest distance between any two states. A weighted Euclidean distance between the means of the (single) Gaussians is used.
On completion of the clustering and tying procedures many of the models may be effectively identical, they are then, so-called generalized triphones. This effect can be exploited to reduce the number of physical HMM's by tying complete models. The resulting models are examples of a hybrid acoustic / linguistic sub-word models, mentioned in chapter 2.3.6.
Tree-based clustering uses phonetic knowledge to come to a clustering. A phonetic decision tree is constructed in which a yes/no question is attached to each node. Initially all states are places at the root node of the tree. Depending on each answer, the pool of states is successively split and this continues until the states have reached the leaf nodes. All states in the same leaf node are then tied. Figure 4.15 shows how the middle states of all triphones corresponding to phone /at/ are tied using a decision tree.
Tree-based clustering may lead to better results than data-driven clustering, but this is only true when enough and the right phonetic questions are formulated. This requires a lot of linguistic knowledge and experience, as this was not the subject of this project it was decided to use data driven clustering, in combination with tying of the transition matrices to allow for model tying. To find the right balance between the number of models and their modeling accuracy about 15 different clusterings were tried, using different clustering-thresholds to find the right number of triphones. Out of these clusterings five different systems were build and each of them was re-estimated twice. For each system first the transition matrices of the triphones that corresponded to the same monophone were tied.

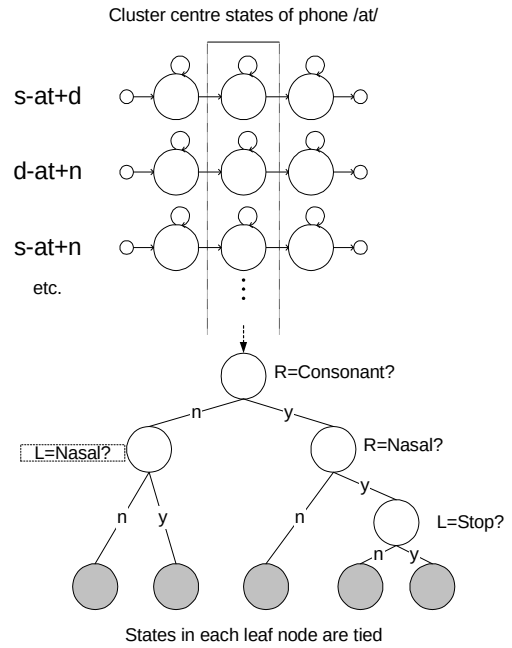


Figure 4.15 - Decision tree-based clustering

Then the models were clusters as described above and finally models that were effectively identical, because their states were in the same clusters and they had the same transition matrix, were tied. All this was done by a script for the HMM-editor HHEd. These scripts are rather long and complicated because all states and matrices that participate in the clustering have to be specified explicitly. Therefore a program, TriScr, was written that, given a list of triphone names and clustering parameters, automatically creates the script.

Table 4.4 - Triphone systems

Triphone system	Number of triphones	Percentage of words recognized	Word accuracy percentage
Ts7c1	101	40.44%	-25.46%
Ts7c2	563	46.48%	-10.28%
Ts7c3	1050	49.57%	-4.77%
Ts7c4	2526	54.92%	5.68%
Ts7c5	8570	62.32%	18.59%

As can be seen from table 4.4 the results get better as more triphones are used, but unfortunately this also means a larger model set. The size of Ts7c5 system, which used no clustering at all, was over 30 Mb, while the size of the Ts7c1 systems was only 518 Kb. Furthermore, in the Ts7c5 system there were more than 1000 models which had only one example in the training data, so the risk of overtraining was quite real with this system. The Ts7c4, which booked fairly reasonable results, had at least three models per cluster and in most case more. Each cluster had at least four examples in the training data. This system seemed to provide a good balance between the number of parameters and the modeling accuracy. It contained less than one third of the original triphones, but was still large enough to model different contexts, therefore it was chosen to be further developed during the subsequent steps.
However, before these steps can be performed one more problem has to be solved. A limitation of the data-driven clustering procedure is that it does not deal with triphones for which there are no examples in the training data, this may give rise to problems during recognition. This may be avoided by careful design of the training database (as long as word-internal triphones are used) but a little research showed that this was not an option in this case. The training data contained 8570 triphones out of 10205 in the dictionary. Using parts of the testing or evaluation database or relaxing the data selection requirements would only partially solve the problem, as the evaluation data set and the test data set together only contained 394 additional triphones. And doing so would of course have introduced new problems, since the test results would then be biased. Even if this would have helped this solution is still not very satisfactory because there is always the possibility that one day in some recognition job a triphone may show up that is not even in the dictionary. Then it would not be reasonable to claim that the resulting recognizer would be a general continuous speech recognizer.
To overcome these problems an approach that could be described as 'backed-off triphone approach' was thought out. In this approach the original monophone models augment the triphone models. During word network construction triphone models are used whenever

available, otherwise the corresponding monophone is used. This is implemented by tying all triphones that have no model of their own to the corresponding monophone. This was done by manually manipulating the HMM definitions since there are no tools that support this particular type of tying. Essentially, the monophones became generalized triphones. But they are less specialized than the other generalized triphones because they contain all corresponding triphones that are now in other sets. Actually they are trained on all the triphones but the ones they represent, but being monophones they lack most context information, so they are general enough to cover the unseen triphones. This results in a robust recognizer that uses triphones most of the time and does not break down when an unknown triphone is encountered.
As with the monophone system the modeling accuracy of the generalized triphone system can be improved by incrementing the mixture components to get better density functions. This was done for the Ts7c4 triphone system. Once again the mixtures were incremented by two or three at a time, with two re-estimation cycles between increments. The minimal number of examples needed to update a model was increased in each step to reduce the number of floored mixtures and prevent overfitting.

Table 4.5 - Multiple mixture triphone systems

Number of mixtures	Percentage of words recognized	Word accuracy percentage
2	56.27%	8.02%
3	58.82%	15.63%
5	61.00%	22.21%
7	64.05%	26.82%
10	66.64%	30.93%
12	68.61%	34.92%
15	70.55%	38.46%
17	71.00%	39.57%

4.5.3 Fine tuning

Once the models were sufficiently trained the complete system was fine tuned using a held-out test set, the steps taken are shown in figure 4.16. The relative levels of insertion and deletion errors can be controlled by adding a fixed word insertion penalty p. A negative value of p results in less word transitions. A large positive value of p would give many short words in the output. These kind of errors and substitution errors can be further controlled by using a grammar scale factor s. Every language model log probability x will be converted to sx-p before being added to the tokens emitted from the corresponding word-end node. The grammar scale factor regulates, in a way, the relative influences of the language model and the acoustic model. A grammar scale factor bigger than one reduces the number of insertion errors and tempers the relative influence of the language model with respect to the acoustic model. A grammar scale factor smaller than one increases the relative influence of the language model.

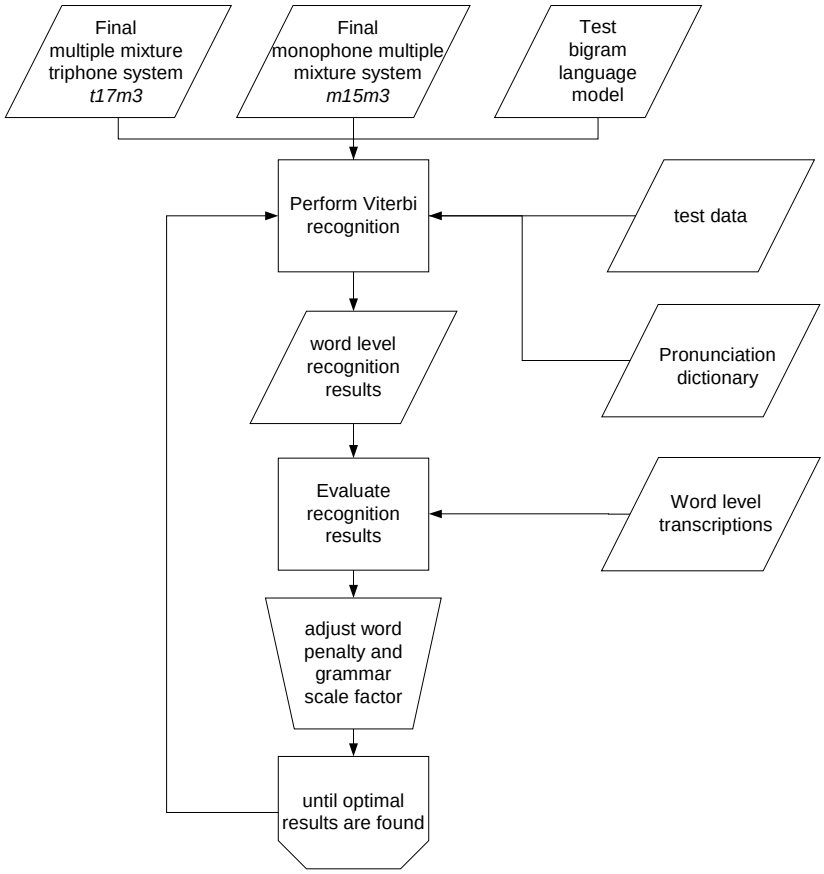


Figure 4.16 - Fine-tuning of the final system

The final acoustic model used was a combination of the 17-mixture generalized triphone set and the 15-mixture monophone set. Varying the word insertion penalty did not improve the overall recognition results, in fact the system proved rather insensitive to this value. Large positive or negative insertion penalties resulted in a decrease in performance of about 2%. The grammar scale factor had a more positive effect. Increasing the grammar scale factor improved the recognition results considerably. Its maximum was found at s = 9, giving the following results, for the final system:

Table 4.6 – Recognition results

Percentage of words correct	95.27%
Word accuracy percentage	89.59%
Percentage of sentences correct	38.43%

4.6 EVALUATION

As can be seen in the last section the generalized triphone multiple mixture system performs rather well. Its results improved from 17% word recognition in the first step of the development process to 95% in the final step. The improvements in word accuracy were even more dramatic; the fist set of trained models had a word accuracy of -77.17%, while the final systems reached 89.59%. But all these results are based on a 1000 word test set, that was used during the process

to decide which steps to take and to tune parameters like the cluster size in the triphone set, the number of mixtures and the grammar scale factor. So these results are in a way compromised and likely to be a bit too optimistic. To test the robustness of the system and to see how well it will perform on other data a number of evaluation tests were run.
First recognition was performed on part of the evaluation test set created in the data selection step. This set contained 100 sentences, each of which was spoken by a different person. The bigram wordnetwork used contained 5017 different words. The test gave the following result:

Table 4.7 – Recognition results

Percentage of words correct	93.55%
Word accuracy percentage	88.76%
Percentage of sentences correct	32.56%

So the recognizer generalizes very well to speakers it was neither trained nor tuned on even when al larger word network is used.

4.6.1 Using a different data test set

Although the evaluation data did not occur in the training or development test set it also came from the Polyphone database, so it was recorded under similar conditions as the other two sets, in particular it was recorded over a telephone line. As mentioned in 4.1 the speech recognizer should recognize speech recorded by a PC microphone and it should be easily adaptable to specific tasks and applications. To test how well the recognizer generalizes to other data and other environments a small dataset was recorded at TU Delft using a digital video camera. This data set will be described in more detail in chapter 6, for the moment the following information is sufficient. The part of the data set used consisted of 5 different persons, all computer science students at TU Delft, four male students and one female student. From each person 4 or 5 recording sessions were used. Each session contained 23 sentences, ten of which were phonetically rich sentences, similar to those in the Polyphone database. The other sentences contained a sequence of short words, a sequence numbers, a spelled word or a command from a telebanking application that adhered to the grammar of Figure 2.5. The dataset was split in a training set containing about 500 utterances, that is, about 100 per person and a test set containing 30 utterances. All utterances were stored in the CITT A-law audio format and subsequently MFCC feature vectors were extracted in the same way as described in section 4.3.2. First the system was tested without any further training, which gave the following results:

Table 4.8 – Recognition results

percentage of words correct:	87.30%
word accuracy percentage	84.59%
percentage of sentences correct	36.36%

Although still reasonable, the performance clearly decreased in comparison to the performance obtained in the last two sections. These results were to be expected, since the data set was

recorded using a video camera in a quiet laboratory, while the Polyphone on which the system was trained was recorded over a normal telephone line. So the ambient noise, which is modeled in the acoustic HMMs, will be quite different, which means that the acoustic vectors in the data set are still similar to those produced by the corresponding HMMs, but there is some distortion, causing some classification mistakes. To make up for this effect, the system was adapted to the new situation by re-estimating once, using the training part from the recorded data set. This time recognition produced the following results:

Table 4.9 – Recognition results

percentage of words correct:	96.76%
word accuracy percentage	95.41%
percentage of sentences correct	60.61%

The performance thus increased considerably. Actually, it is better than any of the results obtained in earlier tests; especially the word accuracy and percentage of correct sentences are very good. The explanation for these results lies in the fact that the system not only adapted to the background noise in this data set, but it also adapted to the voices of these five persons. In fact the system has become a speaker dependent system. The voices it is adapted to are recognized very well, but now recognition of other speakers might give some trouble. To show these effects two more tests were performed. In the first test the system was adapted using only 4 different persons. Recognition was then performed using data from the fifth person. The results below show that although the performance is not as good as the ones in the previous test, the system is still better than the unadapted system. So it adapted to the new environment. Since the person used in the test was not part of the training set, the system is still capable of generalizing and performing speaker independent recognition.

Table 4.11 – Recognition results

percentage of words correct:	91.80%
word accuracy percentage	93.44%
percentage of sentences correct	47.62%

To show that the system adapted to the new environment recognition was performed once again on the Polyphone test set using the system that was adapted to four persons, giving the following dramatic results:

Table 4.12 – Recognition results

percentage of words correct:	32.16%
word accuracy percentage	-21.44%
percentage of sentences correct	36.36%

Thus the system no longer recognizes the data it was developed with, it has completely adapted to the new environment. That the results are this bad is due to the fact that the polyphone data contains much more noise, since it is recorded over a telephone line, than the data set used here.

Recognizing the PC recorded data with an unadapted Polyphone trained system worked because from the systems point of view these were just very high quality recordings. But from the point of view of the adapted system the polyphone data set contains very noisy recordings, indeed many sounds were classified as mouth noise. As was mentioned, this set also contained sentences that adhered to the telebanking grammar from chapter 2. Using the system that was adapted to four persons recognition was performed on these sentences. A word network that implements the grammar from figure 2.5 was used. This gave rise to the following results:

Table 4.13 – Recognition results

percentage of words correct:	75.30%
word accuracy percentage	72.59%
percentage of sentences correct	68.00%

One would expect the percentage of correct words to be higher as only a small vocabulary is used and the syntax of the sentences is constrained. A sentence-by-sentence inspection of the results showed what was going on here. In most cases the system did recognize the right sentence, but when it made a mistake it often recognized a completely wrong sentence in which only a few words were correct. So about 25 percent of all sentences are responsible for most word errors.

REFERENCES

1. Rabiner, L.R., Juang, B. H., Fundamentals of Speech Recognition, Prentice Hall, Englewood Cliffs, N.J., 1993
2. Jelinek, F., Statistical Methods for Speech Recognition (Language, Speech, and Communication) MIT Press, January 1999
3. Alphen, P. van, Hidden Markov Models in Speech Recognition, Academic Press
4. Huang, X.D., Ariki, Y., Jack, M.A., Hidden Markov Models for Speech Recognition, Edinburgh University Press, 1990
5. Furui, S., Digital Speech Processing, Synthesis and Recognition, Second Edition Revised and Expanded, Marcel Dekker, February 2001
6. Jurafsky, D., Martin, J. H., Van der Linden, K., Speech and Language Processing, An Introduction to Natural Language Processing, Computational Linguistics and Speech Recognition, Prentice Hall, January 2000
7. Junqua, J., Robust Speech Recognition in Embedded Systems and PC Applications, Kluwer International Series in Engineering and Computer Science, Secs 563, Kluwer Academic Publishers, May 2000
8. Barksdale, K., Rutter, M., IBM ViaVoice for the Office Professional, Speech Recognition Series South-Western Educational Publishing, November 2000

9. Young, S., Kersaw, D., Odell, J., Ollason, D., Valtchev, V., Woodland, P. C., The HTK Book (for HTK Version 3.0), Cambridge University Engineering Department
10. Woodland, P.C., Odell, J., Young, S.J., Large Vocabulary Continuous Speech Recognition Using HTK, in Proc. ICASSP 1994
11. Woodland, P.C., Leggetter, C.J., Odell, J., Valchev, V., Young, S.J., The Development of the 1994 HTK large vocabulary speech recognition system, Cambridge Univerity Engineering Department, in Proc. ICASSP 1995
12. Woodland, P.C., Odell, J., Young, S.J., Tree-Based Tying for High Accuracy Acoustic Modelling, Cambridge Univerity Engineering Department, in Proc. ARPA Human Language Technology Workshop, March 1994
13. Woodland, P.C., Young, S.J., The HTK Tied-state Continuous Speech Recognizer, Cambridge Univerity Engineering Department, Proc. Eurospeech '93
14. Woodland, P.C., Young, S.J., Broadcast News Transcription Using, Cambridge Univerity Engineering Department, in Proc. ICASSP '97
15. Grocholewski, S., Acoustic Modeling for Polish, International Workshop Speech and Computers, SPECOM'2000, St. Petersburg, September 2000

16. Damhuis M., Boogaart T., in 't Veld, C., Versteijlen, M.,W. Schelvis, W., Bos, L., Boves L., Creation and Analysis of the Dutch Polyphone Corpus, Proceedings ICSLP '94, pp. 1803-1806, 18-22 September 1994,Yokohama, Japan
17. Boogaart, T.I., Bos L., Boves, L., Use of the Dutch Polyphone Corpus for Application Development, Proc. IVTTA'94, pp. 145-148, 26-27 september 1994, Kyoto, Japan