Lehrstuhl für Mensch-Maschine-Kommunikation

Technische Universität München

Optimization of algorithms for large vocabulary isolated word recognition in embedded devices

Sergey Astrov

Vollständiger Abdruck der von der Fakultät für Elektrotechnik und Informationstechnik der Technischen Universität München zur Erlangung des akademischen Grades eines Doktor-Ingenieurs genehmigten Dissertation.

Vorsitzender:	Univ.-Prof. Dr.-Ing. K. Diepold
Prüfer der Dissertation:	1. apl. Prof. Dr.-Ing., Dr.-Ing. habil. G. Ruske
	2. Univ.-Prof. Dr.-Ing. H. Ney, Rheinisch-Westfälische Technische Hochschule Aachen

Die Dissertation wurde am 22.08.2006 bei der Technischen Universität München eingereicht und durch die Fakultät für Elektrotechnik und Informationstechnik am 26.02.2007 angenommen.

Abstract

Automatic speech recognition requires high processing power and a high amount of memory. Main algorithms in speech recognition (computation of emission probabilities and Viterbi search) are very memory and computation consuming. Modern workstations, personal computers and servers have sufficient memory and processing power, but embedded devices are limited in these resources. Speech recognition in embedded devices should have an acceptable trade-off in memory, processing power consumption and recognition quality. Several memory saving approaches and fast algorithms were investigated and the following results were achieved:

The memory consumption of acoustic models after coding is decreased by 67% (reduction from 104 to 34 KB). The relative increase of word error rate in recognition is less than 10%. The fast computation of emission probabilities requires three times less computations than the baseline algorithm. The emission computation task requires only 8.2 MHz for speech recognition with a 30-word vocabulary, the baseline algorithm requires at least 28.9 MHz on an ARM microcontroller. The new search process on isolated word recognition tasks with a vocabulary of 1500 words requires less than 17 MHz on an ARM processor and 160 KB of memory.

The fast computation of emission probabilities and the compact coding of acoustic model parameters is based on a streams approach. A set of 24-dimensional vectors from acoustic models is divided into streams: in case of 3-dimensional (3-D) streams, the first stream contains 1st, 2nd and 3rd components (dimensions) of vectors, the second stream contains 4th, 5th and 6th components of vectors, and so on. All 3-D stream vectors within each stream are coded by means of vector quantization. Only one shared codebook is used for all streams instead of several codebooks for each dimension, this decreases the memory consumption further.

Distances between feature vector and vectors from acoustic models must be computed during the recognition. This process is performed every 15 ms and requires high amount of computations. For acoustic models with streams these computations are accelerated. In the first step, all possible distances are computed for all stream vectors from the codebook and stored in memory. This is possible because the codebook has a limited number of vectors. In the second step, the distances between feature vector and vectors from acoustic models are computed as a sum of the partial distances of stream vectors. For 3-D streams the computation costs are reduced by 66%.

In order to accelerate the search process, a tree structure is combined with a word stem structure. The new search algorithm takes advantages from both approaches. In a tree structure the words starting with identical phonemes are processed together, the merged word parts with identical phonemes are processed only once during a search iteration, thus, the computation is accelerated. The tree structure requires less memory than the linear structure because the phonemes in similar word parts are stored in memory only once. From the word stem search the new algorithm takes an advantage of stems (linear sequences of HMM states): the regular linear structures of stems are fast to process, the data for every stem is stored compactly in memory that is why the memory cache is used efficiently.

The presented algorithms were tested. With these algorithms the large vocabulary speech recognition becomes possible for embedded devices.

Acknowledgments

I would like to express my deepest gratitude to my supervisors, Professor Harald Höge and Professor Günther Ruske, for giving me guidance and freedom to undertake my research. Their experience and theoretical background were fundamentally important for my work described in this thesis.

I wish to thank Siemens AG, in particular Professional Speech Processing department for the appropriate working environment. I would like to thank also Siemens "Youth and Knowledge" program that supported me, and especially the former coordinator of this program, Eberhard Wildgrube.

Moreover, I would like to thank all former and present colleagues for creating a friendly and stimulating atmosphere and providing fruitful ideas and support during the work. Especially, I would like to thank:

Dr. Josef G. Bauer for assistance and consultations,

Dr. Bernt Andrassy for a very friendly cooperation in writing several publications,

Dr. Petra Witschel for discussions and her help on language modeling,

Panji Setiawan and Ekaterina Timoshenko for their time during discussions and providing helpful suggestions.

My special thanks go to my family for giving me encouragement during the writing of this thesis.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

A man-machine communication in hands-free and eyes-free situations requires voice input and output. User-friendly interactive applications with voice interface could be implemented in cars, where the driver's eyes and hands are busy. An SMS dictation in mobile phones and a voice control in automobiles (e.g. route planers) become very attractive. An application menu structure in modern embedded devices and mobile phones could be very complex. The menu navigation through the menu hierarchy may be accelerated by giving voice commands.

Currently speech recognition technology reached the level that meets user demands. The speech recognition has a good accuracy but requires a high amount of memory and a high number of computations to be performed. Most resource consuming algorithms in speech recognition are the computation of emission probabilities and the Viterbi search. During the computation of emission probabilities the number of computations may reach 30% of the number of computations of the whole recognition process. More than tens of millions expressions in the form of $(a + b)^2$ have to be performed every second. The Viterbi search for a large vocabulary speech recognition may require several megabytes of memory to store the pronunciation lexicon and several megabytes for the search space in order to store probabilities of search paths. The Viterbi search requires a microprocessor available to perform hundreds of millions operations per second for a large vocabulary speech recognition in real time.

Modern workstations and servers have several gigabytes of memory and processors running on high clock frequency (several gigahertz). Embedded systems on the other hand have limited system resources: 4-256 megabytes of memory and processing power about 50-500 MHz clock frequency. These system resources are shared between several applications. In embedded devices several tasks may be processed simultaneously, for example, navigation application, GSM connection, menu navigation, speech recognition, calendar, task and contact management, etc. The speech recognition is not the main process, that is why speech recognition applications have even less resources than listed above. The pro-

cessing power is also limited by the battery capacity because the decrease of CPU clock frequency saves battery energy.

Manufactures offer today devices with low vocabulary isolated word recognizers like mobile phones or PDAs with voice interface. Name dialing based on suited for this purpose dynamic time warping (DTW) algorithm is implemented in many modern mobile phones.

The speaker-independent speech recognition in embedded devices based on hidden Markov models (HMM) becomes attractive. Such recognizers may be used for digits dialing, command-and-control applications, manipulations with telephone book, song title selection, formatted speech dialogs and SMS dictation. The greatest advantage of HMM-based recognizers is that no training is required from the user.

The main technical problem of the speech recognition in embedded devices is a disbalance between available and required system resources. A trade-off between recognition accuracy, required memory and processing power has to be found. This problem may be solved by using fast algorithms and memory saving coding schemes.

Acoustic units are modeled with high accuracy in state of the art speech recognizers. These models occupy a high amount of memory, from hundreds of kilobytes to several megabytes. Speech recognizers in embedded devices use less precise acoustic models that occupy several hundreds of kilobytes or less. The memory requirements may be reduced further by using compression algorithms which should have properties and satisfy requirements listed below:

- Compression may be performed off-line with no restrictions to the compression time.
- The decompression algorithm should be fast and performed ”on the fly”. The algorithm should allow random decoding: any value should be obtained without having to decompress the whole file or array with acoustic models.
- The compression algorithm may introduce coding errors, but the recognition performance should not substantially degrade.

The performance of the search algorithm may be improved by using non-linear lexicon structures, e.g. trees. In the tree lexicon equal prefixes from different words are shared, the memory requirements and computation costs are lower than in case of a linear structure. The fast search algorithm should have the following properties:

- The loss of recognition accuracy should be very low.
- Data structures and algorithms must be optimized for the architecture of modern microcontrollers used in embedded devices.

In this work the following results were achieved:

- The reduction of the memory demands for acoustic models by 66% was achieved by coding of parameters. Coding was performed by means of vector quantization and streaming. The set of multidimensional vectors were split into several subsets of lower dimensions. For example, in case of 3-D streams the first 3 dimensions of all vectors form the first subset (stream), the second group of 3 dimensions form the second subset, and so on. Then these subsets were coded by means of vector quantization. The novel approach is that only one shared codebook was used instead of several independent codebooks for each stream. One shared codebook requires less memory than all independent codebooks.

- The memory consumption of vector weights of acoustic models was reduced by 50%. The weights were coded by means of scalar quantization, recursive coding procedure or special mapping of values into their square root values.

- The computation of emission probabilities (distances between the feature vector and acoustic model vectors) was accelerated 3 times by employing stream coding.

- The coding does not degrade the recognition performance, the relative increase of word error rate is less than 10%.

- The fast word stem based tree search was developed for the large vocabulary speech recognition. The new search requires 3 times less memory and performs 3-5 times faster than the baseline linear search algorithm. With the frame dropping approach (elimination of non-speech signal from the consideration by the search algorithm) the recognition is performed 5-12 times faster than with the baseline search.

The results of this work show that large vocabulary speech recognition becomes possible in modern embedded devices with limited system resources.

In this thesis the main technical problem of speech recognition in embedded devices is considered in details. The objectives stated above are reached by means of new algorithms described in the following chapters.

Chapter 2 describes fundamentals of speech recognition in embedded devices. First, the theoretical aspects of the statistical speech recognition are presented. Then the structure of a typical embedded device is considered. Special attention is paid to parameters that require high amount of memory and algorithms that consume most of processing power. Fundamentals of coding theory that may be used to reduce memory consumption by acoustic models are described in the second part of the chapter.

In Chapter 3 state of the art coding of lexicon structures and acoustic parameters are considered. Then the fast computation algorithms in speech recognition are discussed. The advantages and disadvantages of existing approaches are described. Finally, objectives of the research are formulated.

Chapter 4 describes the experimental setup. Properties of speech databases, test sets and baseline acoustic models are considered in detail. The description of used databases and test sets is important for comparison purposes. The recognition results of different recognizers may be compared if they use the same database and similar test sets.

The principles of memory saving coding of HMM parameters are considered in Chapter 5. The developed coding techniques are described in detail. The results of the lossless coding and the Shannon's noiseless coding theorem are used as a first estimation of achievable compression performance. Then lossy coding approaches are considered. The coding approaches are tested on two tasks: coding of Gaussian mean vectors and coding of weights.

In Chapter 6 the reduction of computational complexity of the emission computation algorithm is explored. The algorithms based on streams approach and vector quantization are considered and tested. The combination of these algorithms may accelerate computation further. Experiments are made for different languages and acoustic model sizes.

Chapter 7 is dedicated to fast search algorithms in speech recognition. Firstly, theoretical aspects of the search are considered in detail. Then a tree search and a word stem based search approach are described, the advantages and disadvantages are shown. The combination of these approaches is investigated. Special attention is paid to the frame dropping approach which was used in experiments. The frame dropping provides the search algorithm only with speech frames and drops non-speech frames. The modifications of the search algorithms suitable for the speech recognition in embedded systems are described. The experimental results are discussed.

In Chapter 8 the main achievements of the research are summarized. An outlook on future work is given.

Chapter 2

Fundamentals of speech recognition for embedded devices

Speech recognition for embedded systems concerns mainly two topics: the general statistical speech recognition which is discussed in Section 2.1 and the source coding including scalar and vector quantization (see Section 2.2).

2.1 Fundamentals of statistical speech recognition

2.1.1 Baseline algorithms in speech recognition

Automatic speech recognition is considered nowadays as a pattern recognition problem. According to the Bayes decision rule [Duda and Hart 1973] the word sequence $W = w_1, \ldots, w_N$ should be chosen to maximize the posterior probability of the observed sequence of acoustic vectors $X = x_1, \ldots, x_T$:

$$W' = \arg\max_W p(W|X) \tag{2.1}$$

Using Bayes equation, the posterior probability can be written as:

$$p(W|X) = \frac{p(W) \cdot p(X|W)}{p(X)} \tag{2.2}$$

Then Equation 2.1 can be rewritten as:

$$W' = \arg\max_W \frac{p(W) \cdot p(X|W)}{p(X)} \tag{2.3}$$

Here the a priori probability $p(X)$ of the acoustic vector sequence is a constant, it has no influence on the optimization problem and may be omitted. Finally, the decision rule can

be rewritten as:

$$W' = \arg\max_{W} p(W) \cdot p(X|W) \tag{2.4}$$

In order to solve the Equation 2.4 it is necessary to estimate the acoustic model with probability distributions $p(X|W)$ and the language model that provides the a-priori probabilities $p(W)$ of the word sequence W. These probabilities are used by an automatic speech recognition system with a typical architecture shown in Figure 2.1 (see [Ney and Ortmanns 2000]).

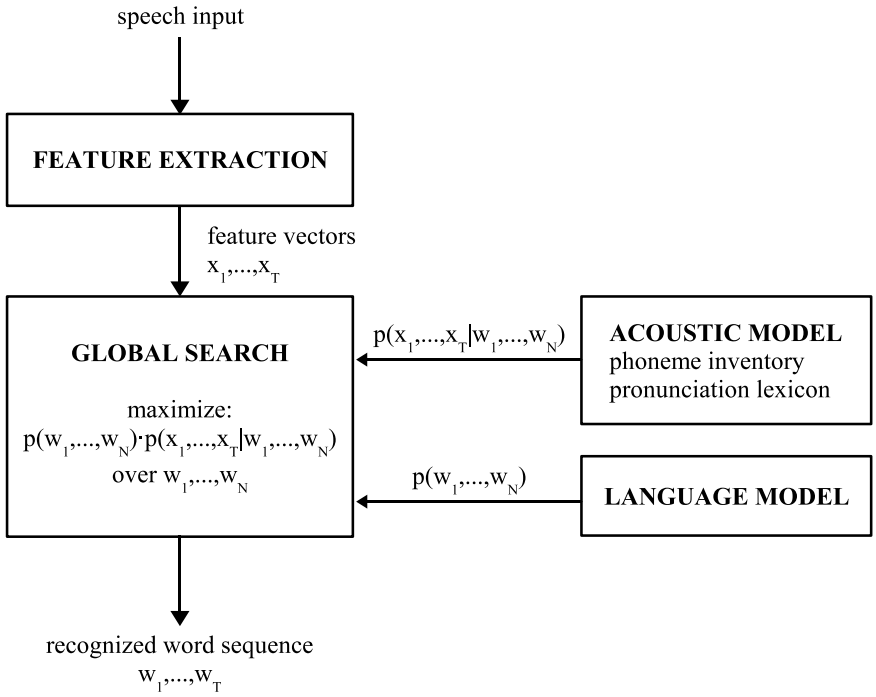

Figure 2.1: *Principal architecture of an automatic speech recognition system*

The structure of the recognizer consists of four general parts:

- The *feature extraction* unit performs the transformation of a speech signal to a sequence of feature vectors X. The feature vectors are generated periodically, every 15 ms, for example.

- The *acoustic model* part describes the probability to observe a sequence of acoustic vectors X given the hypothesized word sequence W and consists of two parts: the set of acoustic models for the smallest sub-word units (typically phonemes) and the pronunciation lexicon that describes how the words are composed from the sub-word units.

- The *language model* part covers knowledge of the language (syntax, semantics) and provides the a priori probability of the hypothesized word sequence.

- The *global search* unit finds the word sequence of a maximum posterior probability according to the decision rule, see Equation 2.4.

2.1.2 System architecture of embedded devices

The typical system architecture of embedded devices is considered by the example of a mobile phone. Figure 2.2 presents only system units which are needed for speech recognition. The feature extraction for speech recognition may be realized on a microcontroller. Alternatively, the feature extraction may be realized on the digital signal processor (DSP), then feature vectors have to be passed to the microcontroller for further processing.

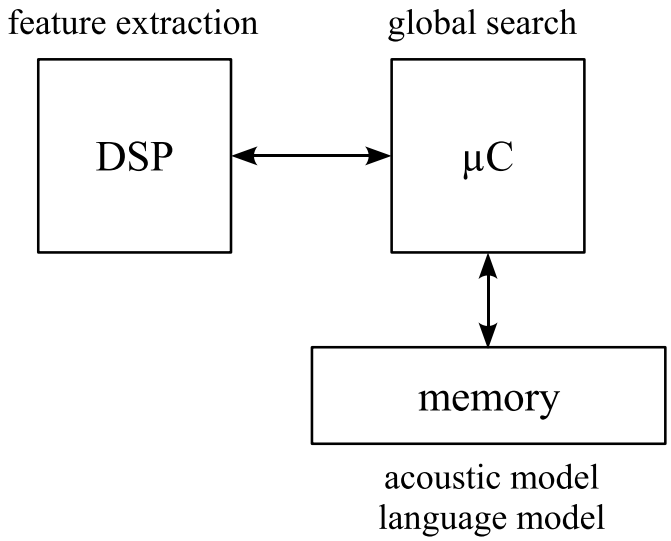

Figure 2.2: *Speech recognition hardware in embedded devices: the microcontroller (μC) gets feature vectors from the DSP and has access to knowledge sources (language and acoustic models) stored in the memory unit*

The microcontroller performs main control functions in embedded devices. Speech recognition is executed in parallel with other applications (for example, GSM, route navigation, organizer functions), all these processes share memory and processing power. The acoustic and language models are stored in the rewritable flash memory. The content of the flash memory could be changed easily. Thus, it is possible to update data and software for different languages, mobile network providers, countries, etc.

Modern embedded devices have more system resources than desktop computers and workstations 10 years ago. A typical embedded system has a microprocessor running with 50-500 MHz core clock and 4-256 MB of working memory. These system resources are still not enough to perform large vocabulary (more than 10 000 words) speech recognition in real time.

2.1.3 Feature extraction

A speech signal from the microphone cannot be used directly in the classifier, it should be processed and the relevant features should be derived. The main goals of the feature extraction are listed below:

- good separation of different acoustic units;
- reduction of feature distortions induced by recording environment, noise, signal transmission channels, etc.;
- data rate reduction: feature vectors consume less memory than audio data.

In this work the feature extraction algorithm from the VSR Very Smart Recognizer ® (VSR) is used [Varga et al 2002; Köhler 2000], see Section 3.3 for details. Firstly, the analog signal from the microphone is digitized. For telephone applications with bandwidth of 3.4 kHz the speech is sampled at 8 kHz and each sample is coded by 12-15 bits. A first order high pass filter (preemphasis) is applied to the digitized audio data to amplify high frequencies.

Assuming that the speech signal during the short period (10-30 ms) is quasi-stationary, the signal is segmented into overlapping frames. Frames of speech are 32 ms long with 15 ms shift. Each frame consists of $N = 256$ samples.

The samples of each frame are multiplied by a window $w(n)$. In VSR the Hamming window is used:

$$w(n) = 0.54 - 0.46 \cos\left(\frac{2\pi n}{N-1}\right) \tag{2.5}$$

where $n = 1, 2, \ldots, N$ is the current sample index.

The speech samples of each frame are transformed into frequency domain by means of fast Fourier transform (FFT). From the real and imaginary parts the power spectrum is obtained. Then the spectrum is limited to the band from 180 Hz to 3400 Hz. Outside this band the spectrum is set to a low threshold value which is advantageous for the telephone speech. Now a noise reduction may be applied, the additive noise may be eliminated by the spectral subtraction algorithm [Martin 1994] or by the minumum least square amplitude estimator [Beaugeant and Scalart 2001].

The power spectrum is filtered by the set of critical band filters — mel filters. The bandwidths of filters are linearly distributed over the band from 0 to 1000 Hz and logarithmically above 1000 Hz.

The logarithms of the mel filtered power spectrum are calculated, then a discrete cosine transformation is performed. In the next step the cepstral smoothing is applied. Because of

the sensitivity of the low order cepstral coefficients to overall spectral slope and the sensitivity of the high-order cepstral coefficients to noise, it had become a standard technique to weight the cepstral coefficients by a tapered window in order to minimize these sensitivities.

In the next step a channel compensation is performed. A long-term average spectrum is subtracted from the speech signal. In [Hauenstein and Marschall 1995] the recursive adaptive estimation of the average spectrum is described. As a result, it is possible to use already trained acoustic models with different transmission channels, speech databases and speakers without significant loss of recognition accuracy.

The use of derivatives of feature vectors leads to improvement of the recognition rate [Furui 1986]. On the one hand the dynamics of the speech signal is considered, on the other hand the feature vector becomes a higher time context. The discrete derivations Δ and $\Delta\Delta$ are computed as:

$$\Delta u_t = u_t - u_{t-3} \tag{2.6}$$

$$\Delta\Delta u_t = \Delta u_t - \Delta u_{t-3} = u_t - 2u_{t-3} + u_{t-6} \tag{2.7}$$

In such a way the vector v_t consists of normalized cepstral weights, Δ- and $\Delta\Delta$-spectral coefficients:

$$v_t = \begin{pmatrix} u_t \\ \Delta u_t \\ \Delta\Delta u_t \end{pmatrix} \tag{2.8}$$

Two consecutive vectors v_{t-1} and v_t are concatenated into one supervector v_{supt}. From the vector v_{supt} the mean vector μ is subtracted. Then the linear discriminative analysis (LDA) [Hauenstein and Marschall 1995; Haeb-Umbach et al 1993] is applied: vectors $v_{supt} - \mu$ are multiplied with the LDA matrix $\mathbf{A}^\intercal$. In such a way the resulting feature vector x of dimension D is computed as:

$$x_t = (x_{t,1}, x_{t,2}, \ldots, x_{t,D})^\intercal = \mathbf{A}^\intercal(v_{supt} - \mu) \tag{2.9}$$

The LDA transformation has the following advantages:

- The resulting coefficients are sorted by their ability to discriminate acoustic units. Thus, it is possible to take the first 24 coefficients from the resulting vector without the degradation of recognition rate. The decrease of the dimensionality of feature vectors leads to a faster processing of frames.

- After LDA transform feature vectors have equal variances for all dimensions. This is especially advantageous because the computation of emission probabilities is performed faster when the state probabilities are modeled by mixtures of Gaussians with

diagonal covariance matrix where all variances (placed on the diagonal) are equal, see Section 2.1.4 for details.

2.1.4 Acoustic modeling

A typical speaker independent speech recognizer employs the HMM approach [Rabiner and Juang 1993]. The HMM $\lambda(\Pi, A, B)$ is a chain of states, the states generate observation vectors x_t. The transition from HMM state s into state s' is described by transition probabilities $A_{s,s'}$ with the following condition:

$$\sum_{s'} A_{s,s'} = 1, \forall s \tag{2.10}$$

The HMM is characterized also by emission probabilities $B_s(x_t)$; $B_s(x_t)$ represents the probability of the observation x_t conditioned on state s at time t. The initial state probabilities are described by probabilities Π_s of being in state s at start time $t = 1$.

In the speech recognition linear left-right HMM structures are used. Such a HMM with $S = 5$ states is shown in Figure 2.3.

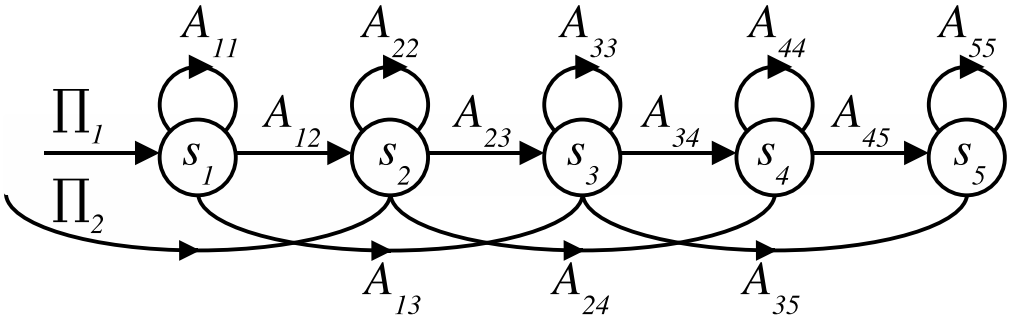

Figure 2.3: *The left-right HMM consists of five HMM states ($S_1 \ldots S_5$); the transition from state s into state s' occurs with probability $A_{s,s'}$; states S_1 and S_2 have initial state probabilities Π_1 and Π_2 respectively*

Modeling of transition probabilities

In the left-right HMM shown in Figure 2.3 from each state s only three transitions are possible:

- into state s itself with probability $A_{s,s}$;
- into next state $s + 1$ with probability $A_{s,s+1}$;
- into state $s + 2$ (skip-transition) with probability $A_{s,s+2}$.

To simplify the estimation of transition probabilities it is supposed that for all states within one HMM structure the following equation takes place:

$$A_{s,s} = A_0, \quad A_{s,s+1} = A_1, \quad A_{s,s+2} = A_2, \quad \forall s \tag{2.11}$$

Here, all transitions of the same type (self-loop, transition into the next state or skip transition) have the same transition probabilities. Transition probabilities between states are not explicitly trained. In VSR the penalty (logarithmically transformed probability) for a transition to the immediate succeeding state is set to 0 while penalties for a self loop and one state skip are set to some fixed value [Bauer 1997].

A word may be represented by a phoneme based or a whole word model [Rabiner and Juang 1993]. In the first case the whole word models are used as a basic speech unit. The whole word models have two disadvantages. First, to obtain reliable whole word models, the number of word utterances in the training set needs to be sufficiently large. Each word from the vocabulary should occur in the training set several times for each possible phonetic context (for each combination of previous and following words). Second, the phonetic content of the individual word is overlapped in case of a large vocabulary. The similar phonetic units of individual words are treated independently.

A word may be represented also as a sequence of subword units. The important advantage is that any word may be modeled by the sequence of subword units (usually phonemes), it is not necessary to have utterances of this word in the training data. The lexicon is flexible, new words may be added to the lexicon without new training. In a phoneme based modeling each HMM state in a word is assigned to a phoneme or a part of a phoneme. A phoneme may be represented as a one-segment model for silence or a three-segment model for phonemes. A silence segment is always modeled by one HMM state. The segments within phonemes have 2 HMM states with the same emission pdfs. In VSR the word HMMs may have also other structures: each phoneme consists of three segments, and each segment has only one HMM state. In this case the HMM structures have no skip-transitions ($A_{s,s+2} = 0$), in such a way a word has 2 times less transitions than in case of 2-state segments. The processing of word HMMs is accelerated with only low decrease of the recognition accuracy.

Modeling of emission probabilities

The set of emission probabilities $B_s(x)$ is the most memory consuming part of the HMM description. Emission probabilities are modeled by mixtures of D-dimensional continuous probability density functions (pdf) or by discrete probability functions. The continuous pdfs usually assume Gaussian distributions defined by mean vectors and covariance matrices. The emission probability of the state s is modeled by the sum of several weighted Gaussian

pdfs with M_s nodes:

$$B_s(x) = \sum_{m=1}^{M_s} C_{s,m} p(x \mid s, m) \tag{2.12}$$

where $C_{s,m}$ denotes a weight coefficient of the pdf $p(x|s,m)$; $\sum_{m=1}^{M_s} C_{s,m} = 1$ for every state s.

In the following emission probability of the state s is modeled by the sum of M_s Gaussian pdfs with mean vectors $\mu_{s,m}$ and covariance matrices $\Sigma_{s,m}$. Such a distribution is defined as:

$$p(x \mid s, m) = N(x, \mu_{s,m}, \Sigma_{s,m}) \tag{2.13}$$

$$N(x, \mu_{s,m}, \Sigma_{s,m}) = \frac{1}{\sqrt{(2\pi)^D \, |\Sigma_{s,m}|}} e^{-\frac{1}{2}(x-\mu_{s,m})^T \Sigma_{s,m}^{-1} (x-\mu_{s,m})} \tag{2.14}$$

where D is the dimensionality of observation vectors x as defined in Equation 2.9, mean vectors $\mu_{s,m}$ and covariance matrix $\Sigma_{s,m}$.

The computation of emission probabilities based on distributions described in Equations 2.13 and 2.14 is very time consuming. Often recognition systems use simplifications such as diagonal covariance matrices and diagonal covariance matrices with equal variances. A diagonal covariance matrix is defined as:

$$\Sigma_{s,m} = diag(\sigma_{s,m,1}^2, \ldots, \sigma_{s,m,D}^2) \tag{2.15}$$

and a Gaussian pdf is defined as:

$$N(x, \mu_{s,m}, \Sigma_{s,m}) = \frac{1}{\sqrt{(2\pi)^D}} \frac{1}{\prod_{d=1}^{D} \sigma_{s,m,d}} e^{-\frac{1}{2} \sum_{d=1}^{D} \frac{(x_d - \mu_{s,m,d})^2}{\sigma_{s,m,d}^2}} \tag{2.16}$$

where x_d is the d-th component of a D-dimensional feature vector x; $\mu_{s,m,d}$ is the d-th component of a D-dimensional mean vectors $\mu_{s,m}$.

In the diagonal covariance matrix all variances may be equal: $\sigma_{s,m,d}^2 = \sigma^2$, in this case the Gaussian pdf has the following form:

$$N(x, \mu_{s,m}, \sigma) = \frac{1}{\sqrt{(2\pi)^D}} \frac{1}{\sigma^D} e^{-\frac{1}{2\sigma^2} \sum_{d=1}^{D} (x_d - \mu_{s,m,d})^2} \tag{2.17}$$

The form shown in Equation 2.17 is used in VSR and will be considered in the following.

Modeling of initial state probabilities

The initial state probabilities in speech recognition are usually defined as shown in Equation 2.18:

$$\Pi_s = \begin{cases} 1, & s = 1 \\ 0, & \text{otherwise} \end{cases} \tag{2.18}$$

The first state has the initial state probability $\Pi_1 = 1$, and the initial probabilities of the other states are equal to zero. For the isolated word recognition task, the first (starting) state of the left-right HMM is a silence-state. It is supposed that before the speech signal there is a silence whose duration is not defined and can be infinitely long.

2.1.5 Computation of probabilities

The classifier in the speech recognizer has to find the probabilities $B_s(x)$ for each feature vector x and every HMM state s. $B_s(x)$ represents the probability of the observation x conditioned on state s. It is assumed that feature vectors have Gaussian distribution and all components of the feature vectors are decorrelated. From Equations 2.12 and 2.17, emission probabilities are computed as:

$$B_s(x) = p(x|s) = \sum_{m=1}^{M_s} C_{s,m} N(x, \mu_{s,m}, \sigma) \tag{2.19}$$

where s denotes a state index; $\mu_{s,m} = (\mu_{s,m,1}, \mu_{s,m,2}, \ldots, \mu_{s,m,D})$ denotes the m-th Gaussian mean vector within the state s; $C_{s,m}$ denotes a Gaussian weight; $N(x, \mu_{s,m}, \sigma)$ denotes a Gaussian distribution with a $D \times D$-dimensional diagonal covariance matrix $\Sigma = diag(\sigma^2, \ldots, \sigma^2)$.

The computation of emission probabilities requires a high number of computations such as:

$$\exp\left(-\frac{1}{2\sigma^2}\sum_{d=1}^{D}(x_d - \mu_{s,m,d})^2\right) \tag{2.20}$$

The computation of exponential functions and division by value $2\sigma^2$ are slow in processors used in embedded devices. For example, for a small vocabulary recognition task with a HMM set of 1200 Gaussians the computation of such expressions is performed 80 000 times per second. In a large vocabulary speech recognition with a HMM set of 20 000 Gaussians such computations are performed more than $1.3 \cdot 10^6$ times per second.

The probabilities may have very low values ($10^{-3} \ldots 10^{-10}$), the result of multiplications of such low values becomes too small in order to be represented as a floating-point value. In the following the single-precision 32 bit number in IEEE Standard for Binary Floating-Point Arithmetic are considered [IEEE 1985]. The floating-point numbers do not model real numbers well, even in simple cases such as representing the decimal fraction 0.1, which cannot be exactly represented in any binary floating-point format. For the computation of low probabilities the most serious problem may be the absorption ($1 + 1 \cdot 10^{-15} = 1$). In

embedded devices often a fixed-point arithmetic is used which has even worse precision than the floating-point arithmetic.

In order to solve these problems all probabilities are represented as negative log likelihoods. Taking logarithms of both parts of Equation 2.19 and multiplying by $-2\sigma^2$ the Equation 2.21 is obtained:

$$-2\sigma^2 \ln B_s(x) = -2\sigma^2 \ln p(x|s) \tag{2.21}$$

Negative log likelihoods are then calculated as:

$$-2\sigma^2 \ln B_s(x) = -2\sigma^2 \ln \left(\sum_{m=1}^{M_s} C_{s,m} \frac{1}{(\sigma\sqrt{2\pi})^D} \prod_{d=1}^{D} \exp\left(-\frac{(x_d - \mu_{s,m,d})^2}{2\sigma^2} \right) \right) \tag{2.22}$$

This equation is difficult to calculate, the following approximation is used:

$$\ln \left(\sum_{m=1}^{M_s} f(x) \right) \approx \ln \left(\max_m f(x) \right) = \max_m \left(\ln f(x) \right), \quad m = 1, \ldots, M_s \tag{2.23}$$

Then the negative log likelihoods are computed as:

$$-2\sigma^2 \ln B_s(x) \approx -2\sigma^2 \ln \left(\max_m \left(\frac{C_{s,m}}{(\sigma\sqrt{2\pi})^D} \prod_{d=1}^{D} \exp\left(-\frac{(x_d - \mu_{s,m,d})^2}{2\sigma^2} \right) \right) \right) \tag{2.24}$$

And finally:

$$-2\sigma^2 \ln B_s(x) = \min_m \left\{ -2\sigma^2 \ln \frac{C_{s,m}}{(\sigma\sqrt{2\pi})^D} + \sum_{d=1}^{D} (x_d - \mu_{s,m,d})^2 \right\} \tag{2.25}$$

$$-2\sigma^2 \ln B_s(x) = \min_m \left\{ -2\sigma^2 \ln C_{s,m} + \sum_{d=1}^{D} (x_d - \mu_{s,m,d})^2 + 2\sigma^2 D \ln \sigma\sqrt{2\pi} \right\} \tag{2.26}$$

The value $2\sigma^2 D \ln \sigma\sqrt{2\pi}$ is always a constant, it is omitted in further considerations. In such a way, for each state the minimum is computed:

$$\min_m \left\{ -2\sigma^2 \ln C_{s,m} + \sum_{d=1}^{D} (x_d - \mu_{s,m,d})^2 \right\}, \quad m = 1, \ldots, M_s \tag{2.27}$$

The distance measure between feature vector x and Gaussian with mean vector μ is a log

likelihood distance; it is calculated according to Equation 2.28 and is called emission score:

$$b_s(x) = -2\sigma^2 \ln B_s(x) = \min_m \left\{ c_{s,m} + \sum_{d=1}^{D} (x_d - \mu_{s,m,d})^2 + const \right\} \tag{2.28}$$

where $c_{s,m} = -2\sigma^2 \ln C_{s,m}$ is a Gaussian penalty and $b_s(x) = -2\sigma^2 \ln B_s(x)$ is an emission score; $m = 1, \ldots, M_s$.

The algorithm of log likelihood computation for state s is shown below.

Emission computation algorithm for state s

1. Set $bestscore = MAXPOSITIVE$ (highest positive value).
2. Set $score = c_{s,m}$ (Gaussian weight).
3. $score = score + (x_d - \mu_{s,m,d})^2$ (add partial distances for each dimension d).
4. If $score < bestscore$ then continue the log likelihood computation (go to step 3 for next dimension d), otherwise abort computations for this Gaussian m.
5. If $score < bestscore$ then $bestscore = score$ (update $bestscore$).
6. Repeat steps 2-5 M_s times for all Gaussians in state s.
7. The result of the score computation for state s is in variable $bestscore$.

In a baseline recognizer (see Section 3.3) the emission log-likelihoods are computed for every state s. If multiplication requires several cycles on a microcontroller, all multiplication operations in computation of distances $(x_d - \mu_{s,m,d})^2$ can be substituted by table look-ups. All values $(x_d - \mu_{s,m,d})^2$ are precalculated for a set of discrete values $(x_d - \mu_{s,m,d})$. In the case of VSR the values $(x_d - \mu_{s,m,d})$ are in the range of $-255 \leq (x_d - \mu_{s,m,d}) \leq 255$. Then the values $(x_d - \mu_{s,m,d})^2$ have to be computed only for 511 values and stored in the array only once for the whole recognition process. The precomputed distances can be stored in the flash memory during the design phase of the recognition system, or the distances can be computed and stored in the memory every time when the speech recognition algorithm is started. The appropriateness of the precalculation procedure is highly dependent on the microprocessor, in modern microcontrollers the integer multiplication may require less computing power than the memory look-up. Other fast log likelihood computation algorithms will be considered in Chapter 6.

In the recognition system the emission probabilities B_s, transition probabilities $A_{s,s'}$ and initial state probabilities Π_s are used as negative logarithms:

$$\begin{aligned} b_s(x) &= -2\sigma^2 \ln B_s(x) \\ a_{s,s'} &= -2\sigma^2 \ln A_{s,s'} \\ \pi_s &= -2\sigma^2 \ln \Pi_s \end{aligned} \tag{2.29}$$

The values a_s and π_s are stored in memory. For the computation of emission probabilities according to Equation 2.28 Gaussian penalties $c_{s,m}$ are precalculated and stored in memory.

2.1.6 Language modeling

Stochastic language models are used to increase the recognition rate in large vocabulary continuous speech recognition. Usually language models are constructed using word n-grams. The n-gram language model is based on the assumption that the probability of a word in a sentence $(w_1, \dots w_{N-1}, w_N)$ depends only on the $n-1$ previous words. In practice, only uni-, bi- and trigrams are used ($n = 1$, $n = 2$ and $n = 3$).

The language model may be built using class n-grams, where words are clustered into several classes [Witschel 2000, 1993; Ney and Essen 1991; Kneser and Ney 1991]. In the class-based bigrams the conditional probabilities $P(w_i|w_{i-1})$ are approximated as shown in Equation 2.30 (see [Ney et al 1989]):

$$P(w_i|w_{i-1}) = P(w_i|C_i)P(C_i|C_{i-1}) \tag{2.30}$$

where the word w_i belongs to the class C_i. The probabilities $P(w_i|C_i)$ are the relative frequencies of words within a given class. Each probability $P(C_i|C_{i-1})$ is estimated on a set of training sentences.

2.1.7 Search algorithm

In modern speech recognition systems a Viterbi decoding is mostly used for the global search task [Ney et al 1992]. Viterbi search [Rabiner and Juang 1993; Viterbi 1967] is a dynamic programming algorithm that follows the transitions within a network of HMM states and maintains the best possible path score at each state in every frame.

The principle of the one-pass Viterbi search is shown in Figure 2.4. One dimension represents the states in the network, and the other one is the time axis. From each state only three transitions are possible: to the state itself, to the next state and the skip transition (see Section 2.1.4). All valid paths start from the start state and end on the final state. The probability value conditioned on state s at time t represents the probability corresponding

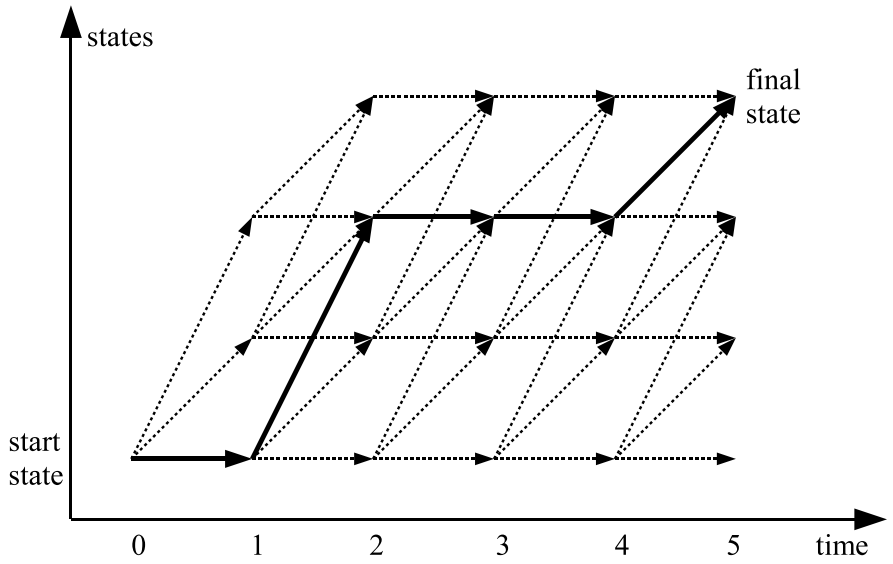

Figure 2.4: *Viterbi search diagram, the best search path is shown by bold arrows*

to the best state sequence leading from the initial state at time 0 to state s at time t. The search is started at time $t = 0$ with the path probability at the start state set to 1, and at all other states to 0.

The single best HMM state sequence $\{s_1, s_2, \ldots, s_T\}$ for the observation $\{x_1, x_2, \ldots, x_T\}$ can be found by obtaining the highest path probability $\alpha_t(i)$:

$$\alpha_t(i) = \max_{s_1, s_2, \ldots, s_{t-1}} p(s_1 s_2 \ldots s_{t-1} s_t = i, x_1 x_2 \ldots x_t | \lambda) \tag{2.31}$$

where $\alpha_t(i)$ is the highest path probability along a single path at time t, which accounts for the first t observations and ends in the state i. The best path probability can be computed recursively:

$$\alpha_{t+1}(j) = \left(\max_i \alpha_t(i) \cdot A_{i,j} \right) \cdot B_j(x_{t+1}) \tag{2.32}$$

where $A_{i,j}$ is the transition probability from state i to state j; $B_j(x_{t+1})$ is the emission probability in state j for observation x_{t+1} at time $t + 1$. In such a way, for every time frame each state has one best predecessor.

The track of the argument that maximizes $\alpha_{t+1}(j)$ for each i and j has to be stored, this is done via array $\psi_t(j)$. After the search is done, the back tracking is performed. The best search path is obtained with the help of the backtracking procedure by starting from the final state and following the best predecessor for each state until the start state is reached. The algorithm is listed below:

Viterbi search algorithm

1. Initialization

$$\alpha_1(i) = \Pi_i B_i(x_1) \qquad 1 \leq i \leq S$$
$$\psi_1(i) = 0$$

where Π_i is the initial state probability of state i.

2. Recursion

$$\alpha_t(j) = \max_{1 \leq i \leq S}[\alpha_{t-1}(i) A_{i,j}] \cdot B_j(x_t) \qquad 2 \leq t \leq T \quad 1 \leq j \leq S$$
$$\psi_t(j) = \arg\max_{1 \leq i \leq S}[\alpha_{t-1}(i) A_{i,j}] \qquad 2 \leq t \leq T \quad 1 \leq j \leq S$$

3. Termination

$$p^* = \max\nolimits_{1 \leq i \leq S}[\alpha_T(i)]$$
$$s_T^* = \arg\max\nolimits_{1 \leq i \leq S}[\alpha_T(i)]$$

4. Path (state sequence) backtracking

$$s_t^* = \psi_{t+1}(s_{t+1}^*), \quad t = T-1, T-2, \ldots 1$$

The implementation of HMM-based speech recognition is simplified using negative logarithms (log likelihoods). Thus, multiplications are substituted by additions of logarithms, see Section 2.1.5.

Negative logarithms of probabilities are computed during the preprocessing stage which is done once in offline mode. The following values are pre-computed and saved in memory:

- initial state scores $\pi_i = -2\sigma^2 \ln \Pi_i$;
- transition scores $a_{i,j} = -2\sigma^2 \ln A_{i,j}$;
- Gaussian weights $c_{j,m} = -2\sigma \ln C_{j,m}$;

where i and j are state indexes. The emission scores are computed shown in Equation 2.28. The Viterbi search algorithm with negative log likelihoods is shown below:

Viterbi search algorithm with negative log likelihoods

0. Preprocessing

$$\pi_i = -2\sigma \ln \Pi_i$$
$$c_{j,m} = -2\sigma \ln C_{j,m}$$
$$a_{i,j} = -2\sigma \ln A_{i,j}$$

1. Initialization

$$\alpha_1(i) = -2\sigma \ln \delta_1(i) = \pi_i + b_i(x_1) \qquad 1 \leq i \leq S$$
$$\psi_1(i) = 0$$

2. Recursion

$$\alpha_t(j) = -2\sigma \ln \alpha_t(j) = \min_{1 \leq i \leq S}[\alpha_{t-1}(i) + a_{i,j}] + b_j(x_t) \qquad 2 \leq t \leq T \quad 1 \leq j \leq S$$
$$\psi_t(j) = \arg \min_{1 \leq i \leq S}[\alpha_{t-1}(i) + a_{i,j}] \qquad 2 \leq t \leq T \quad 1 \leq j \leq S$$

3. Termination

$$P^* = \min_{1 \leq i \leq S}[\alpha_T(i)]$$
$$s_T^* = \arg \min_{1 \leq i \leq S}[\alpha_T(i)]$$

4. Path (state sequence) backtracking

$$s_t^* = \psi_{t+1}(s_{t+1}^*), \quad t = T-1, T-2, \ldots 1$$

A large vocabulary system consists of a large amount of HMM states and the needed computations cannot be performed in real time. In order to solve this problem, for each frame only the most likely states are considered [Ney et al 1992]. At time t the state with the highest path probability $P_{\max}(t)$ is found. Then in further computations at time $t+1$ only states with probabilities $P_i(t) > P_{\max}(t) \cdot B_{prun}$ are processed, where B_{prun} is a pruning threshold or beam width. The states within the beam are considered to be active, all other states are pruned. The beam search reduces the average computation cost of search in medium and large vocabulary systems.

The Viterbi search in continuous speech recognition is performed by the connection of the last HMM states of every word to the first HMM states of every word. When the language model is available, the connections are restricted by the allowed syntax, and the language model probabilities are used for the computation of the overall path probabilities. A one-pass dynamic programming algorithm [Ney 1984] may be used for continuous speech recognition, which is beyond the scope of this work.

2.2 Fundamentals of source coding

Source coding is a process of encoding information using fewer bits than a more obvious representation would use through use of specific encoding schemes [Data Compression – Wikipedia 2006]. The conversion of the quantized audio signal to WAV, MP3 or any of the familiar audio formats is a source coding process. Not only audio signals can be coded, but also images, video frames, etc. – anything containing information is a source, and compressing the stream of information is source coding.

There are two major types of source coding: lossless and lossy. Lossless source coding means that the source information symbols can be perfectly reconstructed without any error or loss. Lossless coding is desired as it brings no distortions, but the lossless requirement puts a serious constraints on how much the source can be compressed.

Lossy source coding allows certain amount of quality loss during the source coding procedure. In many cases lossy coding is preferred over lossless coding because of its strong compressing ability: the loss may be so minimal or somehow ignored by the information receiver (such as the ears when listening to music). For example, the audio format on audio compact disc is lossless and the MP3 audio format is lossy, but usually the human ear does not notice any difference at all when listening to MP3-coded music and an original, but the file size can be more than ten times smaller in the MP3 case.

However, the higher the compression ratio the higher the distortion: there is a trade-off between the source-rate and the distortion, larger source rate means smaller distortion and vice versa. In the following the principles of source coding are described in detail.

Let $S = \{s_1, s_2, \ldots, s_q\}$ be the set of q symbols that defines a source alphabet and the set of r symbols $Z = \{z_1, z_2, \ldots, z_r\}$ defines the code alphabet. Commonly, the binary code alphabet is used ($r = 2$, $Z = \{0, 1\}$). The mapping of each symbol s_i from S to a codeword $v_i = (z_{i1} z_{i2} ... z_{ik})$ of length k symbols is called the source encoding: $s_i \rightarrow v_i$. The source decoding is the reverse process of such mapping.

The n-th extension of a code which maps the symbols s_i into the codewords v_i is the code which maps the n-block sequence of source symbols $(s_{i1}, s_{i2}, \ldots, s_{in})$, the n-th extension of S, into the corresponding sequence of codewords $(v_{i1}, v_{i2}, \ldots, v_{in})$.

In order to use coding in practice, the code has to be uniquely decodable. A code is unique

decodable if, and only if, the n-th extension of the code is nonsingular (all the code words are distinct) for every finite n.

The code with shorter codewords on average is preferable for applications in embedded systems. One of the benchmarks that measures the average effect of the different codeword lengths is the average length of a code:

$$L = \sum_{i=1}^{n} P_i l_i \tag{2.33}$$

where L is the average length of the code; P_i is the probability of the source symbol s_i and l_i is the length of the codeword v_i.

The uniquely decodable code is called a compact code if its average length is less than or equal to the average length of all other uniquely decodable codes for the same source and code alphabet. The lower limit on how small the average length for a particular source and code alphabet can be obtained from the Shannon's noiseless coding theorem [Togneri 2002].

Shannon's noiseless coding theorem For each information source the average length L of a compact code is related to the entropy $H_r(S)$ by:

$$H_r(S) \leq L \leq H_r(S) + 1 \tag{2.34}$$

The entropy to the base r may be derived as shown in Equation 2.35:

$$H_r(S) = \sum_{i=1}^{q} P_i \log_r \frac{1}{P_i} = \frac{1}{\log_2 r} \sum_{i=1}^{q} P_i \log_2 \frac{1}{P_i} = \frac{H(S)}{\log_2 r} \tag{2.35}$$

where P_i is the probability of source symbol s_i; $H_r(S)$ is the entropy to base r; $H(S)$ is the entropy to base 2. The value of $H(S)$ is a number of bits that are required to code one source symbol.

The average length L_n of a compact code for the n-th extension of the information source is related to $H_r(S)$ by:

$$H_r(S) \leq \frac{L_n}{n} \leq H_r(S) + \frac{1}{n} \tag{2.36}$$

and thus:

$$\lim_{n \to \infty} \frac{L_n}{n} = H_r(S) \tag{2.37}$$

where $\frac{L_n}{n}$ represents the average number of code symbols per single symbol from S when coding the n-th extension of S.

The theorem implies that $\frac{L_n}{n}$ can be decreased as close to $H_r(S)$ as possible by increasing n. With increasing n the number of source symbols q^n grows exponentially and the coding

complexity increases in a similar way.

2.2.1 Theoretical aspects of scalar quantization

The scalar quantization (SQ) is a lossy data compression technique, its implementation in speech recognition in embedded devices has the following restrictions: the coding should bring the distortion as low as possible and the decoding algorithm should have low computational complexity and low memory requirements.

The scalar quantizer represents a data set $X = \{x_1, x_2, \ldots, x_n\}$ by the reproduction data set $\hat{X} = \{\hat{x}_1, \hat{x}_2, \ldots, \hat{x}_n\}$. Each reproduction value $\hat{x}_i$ comes from a finite set $V = \{v_1, v_2, \ldots, v_N\}$ which is called the quantizer codebook ($\hat{x}_i \in V$, $1 \leq i \leq n$). A reproduction value $\hat{x}_i = v_j$ is stored in memory as a codeword index j which is an integer $1 \leq j \leq N$. The code (index) occupies less memory than corresponding data value x_i, in such a way the set X is compressed.

In this work the scalar quantization is applied to code acoustic model parameters for speech recognition. The main goal is to achieve high compression rate with an insignificant loss of speech recognition accuracy. Unfortunately, it is hard to estimate the changes in the word error rate from the quantization distortion which is brought by the scalar quantization of acoustic model parameters. The decrease of the quantization distortion leads to decrease of the word error rate, that is why the main goal of the scalar quantization for speech recognition is to obtain lowest quantization distortion G:

$$G = \frac{1}{n} \sum_{i=1}^{n} g(x_i, \hat{x}_i) \tag{2.38}$$

where $g(x_i, \hat{x}_i)$ is a distance measure between value x_i and reproduction value $\hat{x}_i$. In the following the Euclidean distance $g(x_i, \hat{x}_i) = (x_i - \hat{x}_i)^2$ is considered. The distortion then can be written as:

$$G = \frac{1}{n} \sum_{i=1}^{n} (x_i - \hat{x}_i)^2 \tag{2.39}$$

An N-level scalar quantizer is defined by selecting the following items:

- an interval $[x_{\min}, x_{\max}]$ which should be large enough to contain all of the samples x_i to be quantized;
- a partition of $[x_{\min}, x_{\max}]$ into N subintervals $I_1, I_2, \ldots, I_N$;
- quantization levels $v_1, v_2, \ldots, v_N$ such that $v_j \in I_j \quad j = 1, 2, \ldots, N$.

The set $\{x_{\min}, x_{\max}, I_1, I_2, \ldots, I_N, v_1, v_2, \ldots, v_N\}$ uniquely defines the scalar quantizer. The input of the scalar quantizer is any real number from the interval $[x_{\min}, x_{\max}]$, the output is the index j of the quantization level v_j (code) assigned to an interval I_j.

Uniform quantizer

The uniform quantizer is the simplest quantizer which has equal subintervals I_j, the quantization levels v_j are the midpoints of the intervals I_j. This quantizer is designed using the algorithm shown below:

Uniform scalar quantizer algorithm

1. Choose an interval $[x_{\min}, x_{\max}]$ which contains all samples x_i from the data set, $1 \leq i \leq n$.
2. Split the interval into N subintervals I_j of equal lengths, $1 \leq j \leq N$.
3. Form a codebook V such that quantization level v_j is a midpoint of the subinterval I_j.
4. Quantize each sample x_i ($x_i \in I_j$) into the codebook entry v_j.

The uniform quantizer distortion satisfies the condition:

$$G \leq \left(\frac{x_{\max} - x_{\min}}{2N} \right)^2 \tag{2.40}$$

Lloyd-Max quantizer

The N-level optimal quantizer maps x_i into levels (quantization values) $\hat{x}_i \in V$ with minimum distortion G (see Equation 2.39). An optimal quantizer for the data set X should satisfy the Lloyd-Max conditions [Lloyd 1982]:

- Scalar quantizer should be the nearest neighbour quantizer; the quantized value $\hat{x}$ for any x must be a quantization level in V which is the closest to x:

$$|x - \hat{x}| = \min_{1 \leq j \leq N} |x - v_j| \quad \forall x \tag{2.41}$$

- Each quantization level v_j must be the average of all samples quantized to this level.

This quantizer is called a Lloyd-Max quantizer. The quantizer which is not a Lloyd-Max quantizer cannot be optimal, but there may be Lloyd-Max quantizers which are not optimal. The Lloyd's iterative algorithm is shown below, it designs a quantizer with low distortion.

Lloyd's quantizer design algorithm

1. Define $N+1$ threshold values y_i such that

$$x_{\min} = y_0 \leq y_1 \leq ... \leq y_{N-1} \leq y_N = x_{\max}$$

   with any initial choice (for example the uniform quantizer)

2. Define the quantization levels v_j

$$v_j = \frac{1}{N_j} \sum_{i=1}^{N_j} x_i \ : \qquad \hat{x}_i = v_j \quad \text{and} \quad j = 1, 2, \ldots, N$$

   where N_j is the number of vectors x_i that falls to the subinterval I_j. Subintervals I_j are defined as:

$$I_j = \begin{cases} [y_{j-1}, y_j), & j = 1, 2, \ldots, N-1 \\ [y_{N-1}, y_N], & j = N \end{cases}$$

3. If data sample x_i falls to subinterval I_j, then the sample is quantized into the average value v_j.

4. The new thresholds are formed according to

$$y_j = \frac{1}{2}(v_j + v_{j+1}), \qquad j = 1, 2, \ldots, N-1$$

5. Repeat steps 2, 3 and 4 recursively until the relative decrease of distortion G becomes less then a predefined threshold ε or after maximal number of iterations.

The design of an N-level scalar quantizer requires sufficient amount n of training data vectors. In practice, the algorithm convergence condition is defined by the rule of thumb:

$$10 \cdot N \leq n \tag{2.42}$$

2.2.2 Theoretical aspects of vector quantization

The vector quantization (VQ) is a data reduction technique that maps a vector set into the small set of discrete symbols (codes), these codes correspond to the reproduction vectors which are stored in a codebook.

The use of vector quantizers in lossy compression systems yields the following advantages: VQ distortion can be below scalar quantizer distortion at the same rate; VQ decoding is fast (simple table look-up); VQ is implementable in low rates.

On the other hand, VQ-based compression systems possess some disadvantages: VQ design can be hard; encoding in a VQ system can be slow.

Let V be the quantization codebook of size of N vectors for a D-dimension vector quantizer. The codebook V is given by:

$$V = \{v_1, v_2, \ldots, v_N\} \tag{2.43}$$

where $v_1, v_2, \ldots, v_N$ are D-dimensional codeword vectors.

The resulting D-dimensional VQ quantizes data sequence $X = \{x_1, x_2, \ldots, x_n\}$ of length n into reproduction data sequence $\hat{X} = \{\hat{x}_1, \hat{x}_2, \ldots, \hat{x}_n\}$. Data vectors x_i and reproduction vectors $\hat{x}_i$ are D-dimensional: $x_i = (x_{i,1}, x_{i,2}, \ldots, x_{i,D})$ and $\hat{x}_i = (\hat{x}_{i,1}, \hat{x}_{i,2}, \ldots, \hat{x}_{i,D})$. During encoding each data vector x_i is quantized into the closest vector $\hat{x}_i \in V$.

The main goal of the application of VQ to speech recognition is to code parameters with less bits and in the same time without significant loss of speech recognition accuracy. Like in case of SQ, the minimization of quantization distortion leads to the minimum of the word error rate.

The distortion G characterizes the quantization error of the whole VQ-based coding system on the given data:

$$G = \frac{1}{n} \sum_{i=1}^{n} g(x_i, \hat{x}_i) \tag{2.44}$$

where $g(x_i, \hat{x}_i)$ is the distance measure between data vector x_i and its reproduction vector $\hat{x}_i$; n is a number of data vectors.

One of the distance measures is the Euclidean distance which will be used in the following:

$$g(x_i, \hat{x}_i) = (x_i - \hat{x}_i)^2 = \sum_{d=1}^{D} (x_{i,d} - \hat{x}_{i,d})^2 \tag{2.45}$$

where d is a dimension index of vectors.

The set of all data vectors that quantized to one codebook vector v_j is called cluster $R(v_j)$. The vector quantizer assigns the reproduction vector $\hat{x}_i \in V$ to each data vector x_i. The j-th cluster $R(v_j)$ for the vector quantizer consists of all x_i for which $\hat{x}_i = v_j$. In such a way, the vector quantizer with the codebook of N vectors generates N clusters (see Figure 2.5).

Clustering is a memory saving coding technique: instead of the reproduction vectors only the codebook and the pointers to the corresponding codebook vectors are stored in memory. Each pointer occupies $\log_2 N$ bits for the codebook with N codewords. Therefore it is

only necessary to reserve $n \cdot \log_2 N$ bits to store pointers for n reproduction vectors (see Figure 2.6).

A VQ-based coding system attempts to find a vector quantizer yielding a desirable trade-off in the compression rate and the distortion G. In an optimal vector quantizer the distortion G in Equation 2.44 is minimized. The design of such a vector quantizer is an intractable computational problem, because of the great variety of potential codebooks that could be

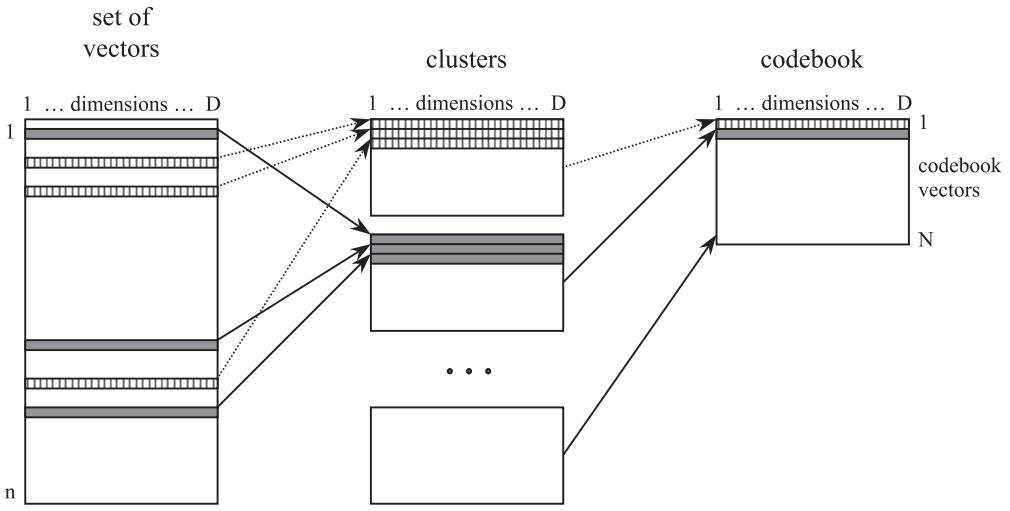

Figure 2.5: *Clustering algorithm: the set of n vectors is broken onto N clusters according to some clustering rule; each cluster is represented by a codebook vector; in the figure vectors and their respective codebook vector within one cluster are filled by the same pattern*

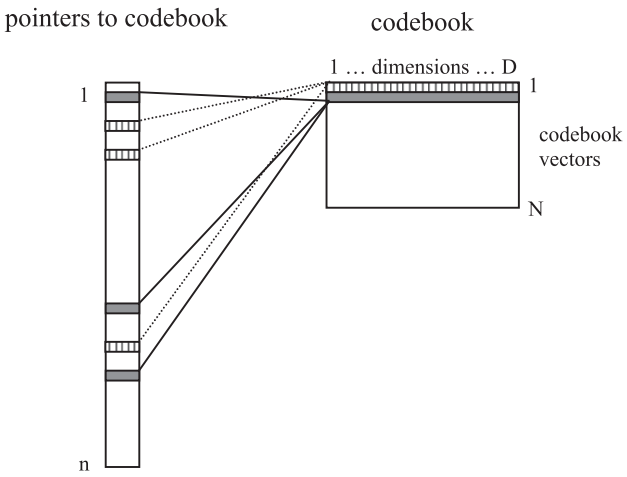

Figure 2.6: *Representation of the reproduction data after coding; during decoding the pointers (coded vectors) are substituted by their respective codebook vectors*

used. It is possible to make the design problem easier by attempting to find a locally optimal quantizer instead. This is what the k-means algorithm does.

The k-means algorithm is an iterative algorithm for the vector quantizer design. The algorithm starts with the quantization codebook of an arbitrary k-dimensional vector quantizer and generates a new codebook on each iteration:

---

k-means VQ algorithm

1. Initialization: choose any N vectors from the training data as the initial codebook.
2. Nearest-neighbour search: for each training vector find the closest code word in the current codebook and assign this vector to the corresponding cell (associated with the closest codebook vector).
3. Mean-values update: update codebook vector in each cell using the mean-value of the training vectors assigned to that cell.
4. Iteration: repeat steps 2 and 3 until the average distance falls below a preset threshold.

Although the above iterative procedure works well, it is advantageous to design an N-vector codebook in stages [Linde et al 1980]. Firstly, the 1-vector codebook is designed. Then using splitting technique on the codebook vectors initialize the search for a 2-vector codebook, and continue the splitting process until the desired N-vector codebook is obtained. This procedure is called the binary split algorithm:

Binary split VQ algorithm

1. Design a 1-vector codebook using k-means VQ algorithm; this is the reproduction vector of the entire set of training vectors.
2. Double the size of the codebook by splitting each current codebook C:

$$v_j^+ = v_j(1+\epsilon)$$
$$v_j^- = v_j(1-\epsilon)$$

   where ϵ is a splitting parameter (typically $0.01 \leq \epsilon \leq 0.05$), and j varies from 1 to the current codebook size N.
3. Use k-means iterative algorithm to get mean vectors for the split codebook.
4. Iterate steps 2 and 3 until a codebook of size N is designed.

During the iterative design of the codebook the mean vectors are updated in order to minimize the average distortion G. The updated mean vector $\bar{v}_j$ of the cluster of N_j data vectors assigned to the same codebook vector v_j is defined as shown below:

$$\bar{v}_j = \arg\min_{v_j} \frac{1}{N_j} \sum_{i=1}^{N_j} g(x_i, v_j) \tag{2.46}$$

where $g(x_i, v_j)$ is the distortion measure of the quantizer.

The solution of the mean vector problem is highly dependent on the choice of the distortion measure and can be obtained by solving the Equation 2.47:

$$\frac{\partial G}{\partial v} = 0 \tag{2.47}$$

For the Euclidean distance (L_2 norm) the mean vector is the mean of the vector set:

$$\bar{v}_j = \frac{1}{N_j} \sum_{i=1}^{N_j} x_i \tag{2.48}$$

After computation of $\bar{v}_j$ the mean vector is updated: $v_j = \bar{v}_j$.

Like in case of scalar quantization, an N-level vector quantizer requires sufficient amount n of training data vectors. In practice, the algorithm convergence condition is defined by the rule of thumb:

$$10 \cdot N \leq n \tag{2.49}$$

Chapter 3

State of the art and objectives of the research

3.1 Computation of emission probabilities

In the HMM based speech recognition system the emission probabilities are computed for every state. Then these probabilities are used in Viterbi search in order to find the most probable state sequence and the respective word or phrase.

The emission score $b_s(x)$ is computed for every HMM state s as shown in Equation 2.28 which is rewritten below:

$$b_s(x) = \min_m \left\{ c_{s,m} + \sum_{d=1}^{D} (x_d - \mu_{s,m,d})^2 + const \right\} \tag{3.1}$$

These computations have to be performed for every time frame for all Gaussians. In order to accelerate this time consuming process, several optimization techniques were developed. One of them is a parameter tying approach for HMMs. The tying (using the same value several times in a model) is often used to optimize the match between the number of parameters in the model and the limited amount of available acoustic training data. The objective of tying is to provide both robust and reliably estimated models. If enough training data is available, tying may lead to the loss of recognition performance. For example, semi-continuous HMMs and discrete density HMMs have poorer recognition performance than that of continuous density HMMs.

For speech recognition in embedded devices tying reduces memory consumption and computation time. Tied HMM parameters require less memory, the tied parameters or the results of their processing are usually stored in a look-up table or a codebook. In the following processing it is only necessary to retrieve the values without new computations. Without tying the same calculations are repeated several times, this results in a slower model evaluation.

Thus, the main goal of HMM parameter tying in speech recognition in embedded systems is to obtain such a tying structure that leads to an optimal trade-off between recognition quality and limited system resources.

The classification of tying levels is presented in [Takahashi and Sagayama 1995]. Four tying levels were defined:

- Tying on the model level is realized in most existing context-dependent HMMs. The left and the right contextual environments having the same effect on the center phoneme can share the same acoustic model.
- In the state level the states having similar feature distributions are tied across different models. This allows to generate context-dependent models with smaller number of HMM states. State tying is typically represented as full tying of states, several states may be identical. This leads to reduction of complexity but also to degradation of accuracy.
- In the distribution level similar Gaussian pdfs (with similar mean vectors and covariance matrices) are tied across different states. This approach is used in semi-continuous HMMs. HMMs with tied Gaussians have increased robustness and reduced computational complexity [Lee et al 2000]. On the other hand, tied mixtures HMMs (TMHMMs) introduce a high number of additional weights.
- Feature parameter tying level was proposed in [Takahashi and Sagayama 1995]. In feature parameter tying hidden Markov models (FPTHMMs) Gaussian mean values are merged into some representative mean values in each dimension by using the clustering technique. The clustered mean values are tied to represent the mean vectors of the distributions.

In [Takahashi and Sagayama 1995] it was experimentally shown that 16 representative values for feature parameter tying are enough for speech recognition without degradation of accuracy comparing to continuous density HMM. Each Gaussian mean vector component occupies only 4 bits as it can have only one of 16 different values. The FPTHMMs were converted from CDHMMs using clustering.

In sub-state tying (SST) the states are only partially tied [Gu and Rose 2000]. State emission probabilities are computed in two stages, each mixture of Gaussians is represented by the “mixture of (smaller) mixtures”. An intermediate level for tying is created, which is positioned between the Gaussian tying of TMHMM and whole state tying which ties the entire state mixture. The combined training and reduction (CTR) approach is combined with SST. The CTR procedure starts with a large universal codebook of Gaussian densities. The reduction of codebook and the mixing coefficient matrix sizes is combined with

parameter retraining. As reported in [Gu and Rose 2002], the considered approach gives better recognition results comparing to TMHMMs and CDHMMs with the same number of Gaussians. At the same time the total amount of memory is increased, this is a serious disadvantage for speech recognizers for embedded systems. Another disadvantage of this method is a complex data structure and a complex algorithm realization.

The further development of FPTHMMs is a stream distribution clustering HMM (SDCHMM) approach which was proposed in [Bocchieri and Mak 2001; Mak and Bocchieri 1998, 2001]. SDCHMMs allow 7 to 18-fold reduction in memory requirements and decrease of recognition time up to 30-60% without degradation of recognition accuracy.

In SDCHMM each HMM state is modeled by a mixture of Gaussian probability density functions. The set of Gaussians is divided into streams. In particular case of 3-dimensional (3-D) streams, the first stream contains the first three components (1st, 2nd and 3rd) of all Gaussian mean vectors, the second stream contains 4th, 5th and 6th components, and so on. All stream vectors within one stream are coded by means of vector quantization. A small number of model parameters leads to low memory consumption and savings in computations.

Possible values of stream vectors are limited in SDCHMMs by a codebook size, that is why the emission computation may be faster than in the recognizer based on CDHMMs. Firstly, all possible partial log likelihoods are computed for each stream for a current feature vector. Then for each Gaussian it is only necessary to sum all partial log likelihoods. A 67% reduction of computation time for emission probabilities was achieved.

A small number of model parameters allows the direct training of SDCHMMs without intermediate CDHMMs with significantly less training data. In [Mak and Bocchieri 1998, 2001] the direct training of SDCHMM with a priori knowledge of the stream distribution tying structure was investigated. Training uses the stream Gaussian tying structure (SGTS) which defines how the stream distributions are clustered. The described training process requires the information about pointers to codebook vectors for each stream. This information can be obtained from the conversion of already trained CDHMMs to SDCHMMs, or from speaker-independent SDCHMMs when speaker-specific SDCHMMs have to be trained. The training of SDCHMM with a priori knowledge of the SGTS requires less training data as the part of the information is encapsulated in SGTS. The disadvantage of such direct training of SDCHMM is a necessity of a priori known SGTS. If this information is not available then the intermediate CDHMM training is required. In the next step the CDHMMs are transformed into SDCHMMs and SGTS is generated. Finally SDCHMM direct training iterations are performed.

In [Komori et al 1995] another fast computation of emission probabilities for CDHMMs using rough and detail models is proposed. This approach requires two HMM sets: one

set contains rough HMMs, the second set contains the detailed HMMs. The proposed algorithm estimates state observation probabilities using rough HMMs and then reestimates most probable states using detailed HMMs. A reduction of computation time up to 70% is reported. The disadvantage of this method is the necessity to keep two HMM sets (rough and detailed) in computer memory.

In [Bocchieri 1993] a vector quantization for the fast computation of emission probabilities is described. In this approach Gaussian models are considered to be statistically accurate only if the input feature vector is near to the Gaussian mean vector. The Gaussian model provides only poor approximation of the emission probability when the feature vector falls on its distribution tail (outlier feature vector).

The traditional algorithm computes emission probabilities of all Gaussians. In the proposed method all Gaussians are clustered into neighbourhoods. For every input feature vector two subsets are defined. The first (small) subset consists of Gaussians whose emission probabilities must be exactly computed. The second complementary subset consists of Gaussians, such that the input vector falls on the tails of these Gaussians; the emission probabilities of these Gaussians are approximated either by table look-up or by a constant.

In [Ortmanns et al 1997a; Sixtus et al 2000] the speed-up of computation of emission probabilities is considered. The fast computation is based on the following approaches.

1. Parallel computation of vector distances using SIMD (single instruction, multiple data) instructions of modern microprocessors [Kanthak et al 2000]. This leads to a speed-up by more than a factor of two without any loss of recognition performance. This approach is processor-dependent.

2. The preselection VQ method is similar to the VQ method proposed in [Bocchieri 1993]. All Gaussians of all states are clustered. The emission computation is divided into two stages. In the first stage emission probabilities of cluster mean vectors are computed. Then emission probabilities are computed exactly for Gaussians in most closely placed clusters. Emission probabilities of other Gaussians which are placed far from the current feature vector are approximated by a constant.

The next fast algorithm of computation of emission probabilities was investigated by Nakagawa and Horibe [Nakagawa and Horibe 2001], they proposed the following three approaches.

1. The first approach is similar to VQ based computation of emission probabilities [Sixtus et al 2000], except that the emission probabilities of Gaussians in far placed clusters are approximated by the emission probabilities of cluster mean vectors.

2. The likelihood computation is pruned on the way whenever the emission probability becomes worse than a given threshold, and the emission probability is set to a constant value.

3. In the third approach time domain skipping for stable frames is employed. If the acoustic feature vector is similar to the feature vector from the previous time frame, the computation of emission probabilities is skipped. The emission probabilities from the past frame are reused.

A computation of emission probabilities using tree-structured pdfs can be considered as the extension of fast computation of emission probabilities based on VQ [Watanabe et al 1994]. The principle of the method is that higher probabilities are computed more precisely than lower probabilities. The method is based on a tree structure. The leaves of the tree are pdfs, the nodes are clusters with corresponding to the cluster pdf. Each cluster consists of clusters or pdfs. Firstly, the emission probabilities of the first level nodes are computed. Then for the most probable nodes the computation is continued. The pdfs in not selected nodes are approximated by cluster emission probabilities. The tree is designed using the k-means clustering algorithm.

In [Watanabe et al 1994] a 3-level tree was investigated, the first level consists of 16 clusters each of them connects to 16 further clusters in the second level. Totally all models contain 1500 pdfs. Trees with more than 3 levels are not suitable for embedded systems because of coding difficulties and increasing number of cluster pdfs. Furthermore, the recognition performance can degrade in case when each node "generates" less than 16 following clusters or nodes.

3.2 Search

In state of the art speech recognition systems the Viterbi search algorithm is widely used. This algorithm finds the most probable sequence of HMM states given a vocabulary and a sequence of acoustic feature vectors. The Viterbi search is very efficient for the recognition with small vocabularies. The computational complexity of the Viterbi search becomes too high with the increase of the vocabulary size. The recognition system may not be able to recognize the speech signal in real time. Several approaches were developed in order to solve this problem.

One of the improvements of the Viterbi search that reduces the search space and the computation time is the beam search. The main idea of this algorithm is removing (or pruning) the most improbable states from the consideration by the Viterbi decoder (see Section 2.1.7). The states with the accumulated path probability lower than some pruning threshold are

eliminated from further consideration. The pruning control can be implemented at the state level or at the model level [Suontausta et al 2000]. In the state level pruning, the decision of removing a state from the search space is done for each state independently of all the other states. In the model level pruning, all the states of a phoneme model are removed at the same time. The realization of the model level pruning is computationally cheaper than the realization of the state level pruning. The pruning threshold may be updated every time frame in order to keep the number of active phoneme models constant over the entire utterance.

The representation of a phonetic lexicon as a tree structure accelerates the recognition speed and saves the memory occupied by the lexicon [Ney et al 1992]. In a linear structure each word is associated with a separate pronunciation form. In large vocabulary speech recognition the linear lexicon is inefficient because the same phoneme sequences occur multiple times. In case of the tree-structured lexicon the words with the same initial phoneme sequences share these phonemes. The shared phonemes are processed only once per frame, thus less computations are required. During the search less memory is necessary to store periodically updated information associated with states as well as less memory is required to store the static lexicon structure.

The idea of the word stem based search is similar to the principle of a tree lexicon [Hauenstein 1993b]. In this approach each word is represented as a set of a word stem and a word ending. First N phonemes of the words form the word stems ($N = const$) and the rest of the word is the word-ending. In the structure the identical stems are merged, this makes the word stem based structure similar to the tree structure. The important advantage of word stems is the strictly regular structure: all word stems are of the same length, they can be processed very fast and efficiently in embedded systems.

In the last years a transducer approach receives more attention in speech processing [FSM–Internet site 2003; Ircing and Psutka 2003; Ljolje et al 1999; Mohri et al 2000; Mohri and Riley 2002; Mohri et al 2002; Rojc and Kacic 2001]. The weighted finite-state transducer in speech recognition is considered as a statically precompiled and optimized network based on the following knowledge sources: language model, phonetic lexicon and phonetic models.

The transducers approach has several advantages. The recognition algorithm is very simple and should only follow the transitions stored in tables. The language model is a part of the transducer, that is why transitions between words are processed by the same algorithm as transitions between states within words.

On the other hand, the precompiling steps usually take a high amount of memory for LVCSR tasks. Memory requirements in case of the fully precompiled transducer network with the language model is higher than in case of the tree structure. Full search space is smaller than in case of the tree structure, but it is still very large in order to be implemented as a static search space. The dynamic search space management is required for embedded

systems. The modification or the complete change of knowledge sources in fully precompiled transducer requires its re-compilation, in case of the tree structured lexicon any arbitrary language model may be used. See also [Kanthak et al 2002; Dolfing 2002] for detailed comparisons of the transducers approach and the tree structured lexicon.

The search unit processes every speech frame regardless of the content of the speech signal. The processing of the silence signal with low noise may require a high amount of memory and processing power. In some cases it can happen, that states in half of a lexicon structure are active (not pruned) when the silence signal is processed. The frame dropping approach may reduce the computational costs of the Viterbi search [Ahmed and Holmes 2004; Ramírez et al 2004; Surendran et al 2004; Marzinzik and Kollmeier 2002]. In this approach a voice activity detection unit classifies each frame into speech or non-speech. The speech frames are sent further to the Viterbi search unit, the non-speech frames are dropped. Thus, the non-speech parts of the signal are not processed.

3.3 Baseline speech recognizer for embedded systems

In this work all research and experiments were performed using the VSR Very Smart Recognizer ® version 3.00 (VSR). VSR is a Siemens platform independent recognizer which may be implemented in various embedded devices like mobile phones, smart devices and PDAs because of the modest memory requirements and low processing power performance. The recognizer has the architecture shown in Figure 2.1 (see also Section 2.1). The preprocessing and feature extraction are described in Section 2.1.3.

The baseline recognizer employs a linear lexicon structure. Each word in the vocabulary is modeled by a linear HMM. Digits are modeled by the whole word models, the amount of states in a word HMM is proportional to the mean lengths of the word. For a general recognition task word models are constructed as sequences of phonemes, each phoneme is modeled by 3 states, each segment consists of 1 HMM state. The silence model has only one state. Transitions are possible only into state itself and into the next state. In the following VSR is explored in isolated word recognition mode, in this case silence states are placed before and after each word.

The emission probabilities of the HMM states are modeled by the mixture of Gaussian pdfs. Each Gaussian is defined by its 24-dimensional mean vector and the weight. The covariance matrix is a diagonal matrix with equal elements on the diagonal, i.e. all Gaussians on all dimensions have the same variance, this single value is specified for a whole HMM set.

The acoustic models trained for speech recognition in embedded devices have reduced amount of parameters in order to allow recognition in real time. The phoneme based HMM sets considered in this work have 4000 or 1200 Gaussians, the HMM set for digits recognition have 1200 Gaussians. The state level tying is used in order to get robust models with such a low number of Gaussians. For comparison, state of the art speech recognizers used

on workstations or designed for offline recognition may have up to 300 000 Gaussians. The baseline recognizer uses a linear Viterbi beam search, the width of a beam is specified by a fixed score threshold.

To benchmark the computational performance of the recognition system being realized in an embedded device, an ARM RISC (Reduced Instruction Set Computer) microcontroller was chosen, which is widely used in embedded systems [Astrov et al 2003; ARM 2002]. For this task the ARM Instruction Simulator (ARMulator) supplied with the ARM Developer Suite v1.2 was employed.

ARM cores come in two basic flavors, Von Neumann and Harvard, depending on the memory access architecture. Von Neumann cores (e.g. ARM7TDMI family) use a single bus for both data and instruction accesses, while the Harvard cores (e.g. ARM9TDMI family) have a separate data and instruction bus, thus allowing simultaneous data accesses and instruction fetches.

Harvard cores are not normally employed in their raw state, but typically a cached variant with a Harvard cache architecture and a Von Neumann memory interface is used. However, benchmarking raw Harvard cores using an ideal zero wait-states memory model can be useful as an indication of the maximum achievable performance for a cached variant assuming 100% cache efficiency. The reference for the maximum performance was obtained by benchmarking on ARM9TDMI uncached core using the default ARMulator model of zero wait states 32-bit memory. Then the benchmarking on ARM920T (16 KB data and 16KB instruction cache) and on ARM940T (4KB data and 4KB instruction caches) was repeated, for a system configured with the processor clock rate of $f = 100$ MHz, the bus clock rate of $f_{bus} = 50$ MHz, and a 32-bit memory with 100 ns non-sequential access time and 20 ns sequential access time. The memory introduces 4 wait states for non-sequential R/W and 1 wait state for sequential R/W access.

The measurements on the workstation were performed for recognition tasks with large test sets (more than 1000 utterances), where measuring using ARMulator requires several days for each task. The relative improvements achieved on the embedded platform may be approximately evaluated using the results obtained on the workstation. In the experiments a two processor Pentium III-850 MHz workstation with 1 GB memory was used. The recognition task uses only one processor, a Linux operation system was installed on the workstation.

A typical low vocabulary command-and-control task may be realized using the baseline recognizer with the processing power requirements listed in Table 3.1. The values are shown for the isolated words recognition with a vocabulary of 30 words and a HMM set of 1200 Gaussians. The emission computations are thought to be performed on a DSP and not listed here, on a microcontroller they would run in $0.08 \ldots 0.12$ real time factor.

computation task	computational requirements real time factor		
	ARM9TDMI	ARM920T	ARM940T
Viterbi search	0.083	0.091	0.135
emission computation	0.262	0.289	0.303

Table 3.1: *Minimal computational requirements in real time factor for a baseline speech recognizer with a 30-word vocabulary*

parameter	4000 Gaussians	1200 Gaussians
Gaussians mean vectors	96 000	28 800
Gaussians weights	8 000	2 400
total	104 000	31 200

Table 3.2: *Memory requirements for baseline acoustic models*

vocabulary, words	495	1500	20 102	76 784
memory requirements (lexicon+search)	94 KB	301 KB	4.7 MB	16.2 MB
recognition speed, real time factor	0.12	0.26	2.50	-

Table 3.3: *Memory consumption and required recognition time for the baseline search algorithm*

The memory consumptions by HMM sets are shown in Table 3.2. A HMM set with 4000 Gaussians requires about 104 KB of memory, a HMM set with 1200 Gaussians requires less than 32 KB.

In Table 3.3 memory consumptions by the linear lexicon structures of different sizes and the required search spaces for the baseline recognizer are presented. The average recognition speed in real time factor is computed on the baseline workstation, the recognition with 20 102 words vocabulary is executed 2.5 times slower than real time. The baseline recognizer is unable to process vocabularies larger than 32 767 words, that is why no measurement results are available for the vocabulary of 76 784 words.

This system performs well only for low vocabulary isolated speech recognition tasks. For medium and large vocabulary isolated recognition tasks the emission computation and the search algorithms have to be accelerated in order to be performed in real time. For large vocabulary speech recognition the implementation of the recognizer in embedded devices requires the reduction of memory consumption by acoustic model parameters, lexicon structure and search parameters.

3.4 Objective of the research

The goal of this research is to developed highly efficient algorithms for doing real time isolated large vocabulary speech recognition on today's embedded platforms. Two algorithms in a speech recognizer are mostly resource consuming: the computation emission probabilities and the Viterbi search. These algorithms must be optimized in the baseline recognizer for implementation in embedded devices. The following goals should be achieved:

1. Memory consumption by HMM parameters has to be reduced by 50%. HMM parameters for speech recognition in baseline recognizer occupy more than 100 kilobytes of memory. The decrease of the memory consumption by a lexicon structure and parameters of acoustic models may be achieved by applying a coding approach. In such a way the optimal representation of the lexicon structure and acoustic models can be reached.

2. Increase of WER caused by tying and coding of parameters of acoustic models must be insignificant. A relative increase of WER less than 10% is acceptable, such increase is almost "invisible" for users. Human could hardly notice the difference between 5% and 6% of the WER.

3. A fast emission computation algorithm has to be developed, it should be 2 times faster than the baseline algorithm which is executed with 0.3 real time factor on the baseline embedded device. The algorithm has to be able process coded acoustic models directly, without decoding procedure.

4. A fast search has to be developed. The processing time for large vocabulary isolated word recognition has to be decreased without any loss of recognition quality. A recognition system with a vocabulary of 20 000 words has to be able to run in real time. The search algorithm has to be able process the optimized lexicon structure.

5. Data structures and algorithms must be optimized for architecture of modern microcontrollers used in embedded devices.

Chapter 4

Experimental setup

The main focus of this work is the speech recognition in embedded devices. Currently, embedded recognizers are used in mobile telephones and automotive environments. In the experiments the telephone and automobile spoken language databases with 8 kHz sampling frequency are used (see Table 4.1). Best recognition performance is achieved if the databases used for training of HMMs are collected in the same environment as the expected environment of the speech recognition application. For detailed description of databases see [SpeechDat-Web Site 2005; ELDA-Web Site 2005; ELRA-Web Site 2005].

In the "database" column the database names are listed. The databases MoTiV and SpeechDat Car [Draxler et al 1999] were recorded in vehicles. Several microphones were installed in a car. The samples were recorded over close-talk and hands-free microphones under different driving conditions. In the "short name" column the short notation of databases and languages are shown, these notations appear in description of experiments.

In Table 4.2 all used test sets are listed. The "test set" column contains the names of the test sets. In the "database" column the short notations of speech databases are shown according to Table 4.1. In the "vocabulary" column a type of speech (isolated or continuous) and a vocabulary size in words is shown. For German digits recognition tasks SieTill-c_d, SDI-1, SDI-2, SDII-c_d, SDII-mbl-c_d and SDCGE-is-c_d the vocabulary consists of 11

database	language	short name	recording environment
VoiceMail	German	VM-DE	fixed telephone network
SieTill	German	SieTill-DE	fixed telephone network
SpeechDat	German	SD-DE	fixed telephone network
SpeechDat II	German	SDII-DE	fixed and mobile telephone network
SpeechDat Car	German	SDC-DE	car
MoTiV (CSDC2)	German	MoTiV-DE	car
SpeechDat II	Spanish	SDII-ES	fixed telephone network

Table 4.1: *Language databases used in experiments*

test set	database	vocabulary	speakers	sentences	words	length
VM62	VoiceMail-DE	isolated, 62	261	—	13600	8h 10m
SieTill-c_d	SieTill-DE	continuous, 11	356	13116	43092	11h 44m
SDI-1	SD-DE	continuous, 11	87	226	1776	25m
SDI-2	SD-DE	continuous, 11	73	173	1396	21m
SDII-c_d	SDII-DE	continuous, 11	488	1649	14862	3h 22m
SDII-mbl-c_d	SDII-DE	continuous, 11	185	527	5282	1h 11m
SDII-mbl-apl	SDII-DE	isolated, 83	1469	—	4391	4h 0m
SDCGE-is-c_d	SDC-DE	continuous, 11	37	877	4871	48m
MoTiV	MoTiV-DE	isolated, 26	230	—	2600	58m
AppW	SDII-ES	isolated, 32	500	—	1414	1h 21m
Spell	SDII-ES	continuous, 30	497	1427	9783	2h 30m
Digits	SDII-ES	isolated, 10	492	—	493	26m
Cities	SDII-DE	isolated	1387	—	1811	1h 41m
CarKit	MoTiV-DE	isolated	172	—	325	8m

Table 4.2: *Description of test sets*

words (German "two" is represented as "zwei" and "zwo"). The Spanish digits lexicon consists of 10 words for the task Digits. The lexicon for spelling in Spanish consists of 30 spelling letters. In test sets VM62, SDII-mbl-apl, MoTiV and AppW vocabularies are application words (command-and-control task). Tasks Cities and CarKit are tested with different vocabulary sizes: 495, 1500, 20 102 and 76 784 words.

For the recognition tests already trained HMMs were used (see Table 4.3), the training procedure is described in [Bauer 2001]. In "HMM ID" the names of HMM sets are shown as they appear in experiments.

The "models" column shows the type of HMM sets. For general recognition tasks phoneme based HMM sets are used. Such HMMs are called "generalists" because they may be used for general recognition tasks. The "generalist" HMM sets allow the recognition of words that have no training material: words must have only phonetic transcription in order to be recognized, the vocabulary size can be increased without retraining.

HMM ID	language	models	context	segments	Gaussians
TRAIN_S	German	phoneme-based	nctx	118	4000
TRAIN_U	German	phoneme-based	nctx	118	4000
TRAIN_BA	German	phoneme-based	ctx	608	1200
TRAIN76	German	whole word	-	217	1257
TRAIN_V	German	whole word	-	238	1199
TRAIN_Q	Spanish	phoneme-based	nctx	85	4000

Table 4.3: *Properties of HMM sets used in experiments*

The second type of HMMs is a whole word models (“specialists”), these HMM sets are trained for one special task with some low-size vocabulary, for example, digits. The vocabulary is fixed and cannot be extended without new training. For example, TRAIN_V HMM set was trained to recognize German digits only, TRAIN76 HMM set was trained to recognize German digits and the “end” word (“Ende”).

“Nctx” in the “context” column means that phonemes are context-independent. “Ctx” means that phonemes are context dependent, they have the influence of the previous phoneme (left context) and the next phoneme (right context). Each phoneme HMM is modeled by three segments, and each segment is modeled by a sum of several Gaussians. A context-dependent HMM set has more different phonemes than a context-independent HMM set. That is why the context-dependent HMM set has more segments and fewer Gaussian pdfs per segment than the context-independent HMM set with the same total number of Gaussians per set.

The total number of Gaussians is the main measure of the HMM set size. With the increase of the HMM set size the recognition performance increases, but the memory requirements becomes higher. HMM set size may reach several hundred thousands of Gaussians, but in speech recognizers in embedded system the number of Gaussians is usually limited by several thousands.

Chapter 5

Reduction of memory consumption of HMM parameters

This chapter is dedicated to the problem of memory saving coding of HMM parameters. The investigated coding algorithms are based on scalar and vector quantization approaches which were explained in Sections 2.2.1 and 2.2.2.

5.1 Properties of HMM parameters

In the following the properties of HMM parameters are investigated. Each HMM state is modeled by a sum of Gaussian distributions (see Equation 2.28). Gaussian m of state s is represented in memory as a set of values:

$$(c_{s,m}, \quad \mu_{s,m,1}, \mu_{s,m,2}, \ldots, \mu_{s,m,D}) \tag{5.1}$$

where $c_{s,m}$ is a Gaussian weight (penalty); $\mu_{s,m,d}$ is a component of dimension d of the Gaussian mean vector $\mu_{s,m}$; d is a dimension index of the Gaussian mean vector $\mu_{s,m}$, $1 \leq d \leq D$.

In VSR each Gaussian distribution has the following properties:

- $c_{s,m} = 0$ if the state s is modeled by only one Gaussian; $c_{s,m} > 0$, otherwise.
- Gaussian penalties are coded as 16-bit integers, usually $0 \leq c_{s,m} \leq 4000$.
- Each Gaussian mean vector component $\mu_{s,m,d}$ is a signed 8-bit integer value from -127 to 127.

In the following considerations a set of all D-dimensional Gaussian mean vectors $\mu_{s,m}$ from a HMM set will be often considered as a set of vectors without making a difference to which

state belong the mean vectors. In order to simplify the considerations all mean vectors are reindexed and the new notation is introduced:

$$\mu_{(i)} = (\mu_{(i),1}, \mu_{(i),2}, \ldots, \mu_{(i),D})$$

where $\mu_{(i)}$ is a reindexed Gaussian mean vector with index i, $\mu_{(i),d}$ is a d-th component of the mean vector $\mu_{(i)}$. The new index is put into brackets in order to distinguish two notations: $\mu_{s,m}$ and $\mu_{(i)}$. As the result of reindexing, the set of mean vectors $\{\mu_{s,m}\}$ is represented as $\{\mu_{(i)}\}$ with totally n vectors.

In the following the set $\{\mu_{(i)}\}$ is divided into D subsets. Each subset $\{\mu_{(i),d}\}$ consists of n Gaussian mean-vector components from dimension d, $\quad (1 \leq d \leq D)$.

The distribution of the scalars from dimension d can be approximated by normal distribution which is evaluated by the mean value $\bar{\mu}_d$ and variance σ_d^2, these values are estimated for each dimension d.

In [Bronstein and Semendjajew 1989] the computation of values $\bar{\mu}_d$ and σ_d^2 is described for the normal distribution:

$$N(\mu_{(i),d}) = \frac{1}{(\sigma_d\sqrt{2\pi})^D} \cdot \exp\left(-\frac{(\mu_{(i),d} - \bar{\mu}_d)^2}{2\sigma_d^2}\right) \tag{5.2}$$

The value $\bar{\mu}_d$ is estimated as shown in Equation 5.3:

$$\bar{\mu}_d = \frac{1}{n}\sum_{i=1}^{n} \mu_{(i),d} \tag{5.3}$$

The variance σ_d is estimated by:

$$\sigma_{d,1}^2 = \frac{1}{n-1}\sum_{i=1}^{n}(\mu_{(i),d} - \bar{\mu}_d)^2 \tag{5.4}$$

or through average error η_d (see Equation 5.6):

$$\eta_d = \frac{\sum_{i=1}^{n}(\mu_{i,d} - \bar{\mu}_d)^2}{\sqrt{n(n-1)}} \tag{5.5}$$

$$\sigma_{d,2}^2 = \frac{\pi}{2}\eta_d^2 \tag{5.6}$$

The distribution of μ_d is close to normal distribution if the difference between the values $\sigma_{d,1}^2$ and $\sigma_{d,2}^2$ obtained by Equations 5.4 and 5.6 is insignificant.

The entropy of Gaussian mean vector components is considered in the following. As it was mentioned before, each mean vector component $\mu_{(i),d}$ is an integer between -127 and 127,

i.e. $\mu_{(i),d}$ is coded with the source alphabet S of $q = 255$ source symbols:

$$s_1 = -127;\ s_2 = -126;\ \ldots s_{254} = 126;\ s_{255} = 127 \tag{5.7}$$

For each dimension d the subsets $\{\mu_{(1),d}, \mu_{(2),d}, \ldots, \mu_{(n),d}\}$ are defined. For each of these subsets the probabilities $P_{j,d} = P(\mu_{(i),d} = s_j)$ of source symbols s_j are computed as a number of events $\mu_{(i),d} = s_j$ divided by amount n of components in $\{\mu_{(1),d}, \mu_{(2),d}, \ldots, \mu_{(n),d}\}$. The entropy $H_{r,d}(S)$ is computed as shown in Equation 5.8:

$$H_{r;d}(S) = \sum_{j=1}^{q} P_{j,d} \log_r \frac{1}{P_{j,d}} \tag{5.8}$$

where q denotes the base of source alphabet $S = \{s_1, s_2, \ldots, s_q\}$; $P_{j,d}$ is the probability of the source symbol s_j in subset $\{\mu_{(1),d}, \mu_{(2),d}, \ldots, \mu_{(n),d}\}$; r denotes the base of the code alphabet. If $r = 2$ then $H_{2,d}(S)$ is a number of bits that are required to code one component of a Gaussian mean vector.

The properties of Gaussian mean values are explored on the example of TRAIN_S HMM set. Estimated mean values $\bar{\mu}_d$ of Gaussian mean vectors, variances $\sigma^2_{d,1}$ and $\sigma^2_{d,2}$ (obtained by two different ways) and entropies $H_{2,d}(S)$ for dimension d are shown in Table 5.1. The values estimated using the data from all of dimensions are shown in the last row of the table.

The distributions of Gaussian mean vector components μ_d for different dimensions are similar, as the σ^2_d values for various dimensions d are close to each other. This property is explained by using LDA for generation of feature vectors. The first dimension is most im-

d	$\bar{\mu}_d$	$\sigma^2_{d,1}$	$\sigma^2_{d,2}$	$H_{2,d}(S)$	d	$\bar{\mu}_d$	$\sigma^2_{d,1}$	$\sigma^2_{d,2}$	$H_{2,d}(S)$
1	3.4588	21.2715	22.7885	6.2716	13	0.8275	11.0441	10.8342	5.4931
2	-0.2462	19.5294	19.4272	6.2708	14	-0.2672	14.7016	14.1537	5.8820
3	3.2207	22.1543	22.4815	6.4835	15	-0.6830	11.7169	11.6118	5.5823
4	-2.3350	17.6524	17.7364	6.1618	16	-0.8590	11.7732	11.3780	5.5712
5	-0.4435	18.3374	18.3807	6.2023	17	-0.3902	8.7989	8.6228	5.1676
6	-1.4283	16.9659	17.0502	6.1105	18	-0.5025	11.1355	10.8894	5.5029
7	0.2435	15.6088	15.5777	5.9853	19	0.0823	11.0213	10.8640	5.4932
8	-0.6985	15.0703	15.0625	5.9247	20	0.1850	11.2290	10.7185	5.4962
9	-0.8405	14.3935	14.2672	5.8730	21	-0.6597	9.2380	8.9918	5.2279
10	-1.2928	13.5822	13.4359	5.7895	22	-0.1415	8.2123	7.7983	5.0470
11	-0.1713	12.0195	11.4809	5.5925	23	0.2258	10.7793	9.5925	5.3029
12	0.3678	11.2029	11.1415	5.5142	24	-0.1375	10.4194	10.2977	5.4126
ALL	-0.1375	14.2419	13.5697	5.8570					

Table 5.1: *Characteristics of the Gaussian mean vectors in German TRAIN_S HMM set: the entropy $H_{2,d}(S)$, the mean value $\bar{\mu}_d$, the variances $\sigma^2_{d,1}$ and $\sigma^2_{d,2}$ (computed using two different approaches) are estimated for each dimension d*

portant, and it has the highest entropy and the highest variance [Hauenstein and Marschall 1995; Bauer 2001, 2004].

All mean vector components are coded in VSR with 8 bits. The results in $H_{2,d}(S)$ columns show, that mean vector components may be lossless coded with 6 or 7 bits. The results from Table 5.1 are obtained for German TRAIN_S HMM set. The results for other HMM sets and for other languages are similar. Since the characteristics of all dimensions are similar, it could be possible to apply the same coding technique with similar parameters or codebooks in order to store the mean values compactly.

5.2 Lossless coding of HMM parameters

Many efficient lossless compression algorithms [DataCompression-Web Site 2002] were developed: from classical algorithms such as Huffman coding [Huffman 1952; Ohm 1995], arithmetic coding [Witten et al 1987; Pennebaker et al 1988], Lempel-Ziv algorithm [Nelson 1989; Ziv and Lempel 1977; Welch 1984], simpler byte pair encoding (BPE) [Shibata et al 2000; Manber 1997] and run-length encoding (RLE) [Maniscalco 2001] to very complex algorithms used in commercial compression software. The common problem of all compression algorithms is a high processing power consumption for decompression algorithm. The RLE and BPE algorithms are very simple but they are inefficient for the HMM parameters compression task.

Different lossless compression algorithms were tested. A test data consists of 4000 Gaussian mean values from German TRAIN_S HMM set. Mean value components were coded as one byte integers. The HMM set of 4000 24-dimensional mean vectors results to a 96000-byte binary file. Table 5.2 demonstrates the results of compression tests.

The first column shows the names of the compression programs. In the second column the lengths in bytes of compressed files are shown. The third column shows the reduction of

compression algorithm	memory consumption, bytes	reduction, %
no compression	96000	baseline
Shannon's limit	70283	26.8
gzip for UNIX	72129	24.9
ice for DOS	72176	24.8
zip for UNIX	72249	24.7
lha for DOS	72712	24.3
arj for DOS	73693	23.2
rar for Windows	74133	22.8
zoo for DOS	91114	5.1

Table 5.2: *Comparison of different lossless coding algorithms on a HMM parameters compression task*

memory consumption of HMM set files.

In the experiments most of the tested compression programs achieve memory reduction of about 25% which is close to the maximum memory reduction estimation from the Shannon's noiseless coding theorem (26.8%), see Section 2.2. This limit was not reached because the coded file includes also some extra information, e.g. file format, compressed file name, etc. The 25% of reduction of memory consumption is still not enough to reach the objective of 50% memory reduction.

Even in case of very efficient compression, the decoding can hardly be implemented as a fast "on the fly" decompression algorithm because of its very high computational complexity. The decoding of random entry is hard without complete decompression or decompression from beginning of the file. The complete decompression of the HMM set and storing in RAM before the recognition process does not make sense for ASR in embedded devices because it would require memory for compressed HMMs as well as for decompressed HMMs.

5.3 Memory reduction for Gaussian mean vectors

5.3.1 Streams approach for HMM

To reduce memory consumption by HMM parameters the lossy coding based on VQ can be applied, this is done in case of semi-continuous HMMs where Gaussian mean vectors are quantized. The recognition accuracy of a recognition system based on semi-continuous HMM approach is lower than of a system based on CDHMMs. To improve the accuracy the VQ is applied to smaller units obtained by the streaming technique.

The set of Gaussian mean vectors in a HMM set is represented as shown below:

$$\left\| \begin{array}{cccc} \mu_{(1),1} & \mu_{(1),2} & \cdots & \mu_{(1),D} \\ \mu_{(2),1} & \mu_{(2),2} & \cdots & \mu_{(2),D} \\ \cdots & \cdots & \cdots & \cdots \\ \mu_{(n),1} & \mu_{(n),2} & \cdots & \mu_{(n),D} \end{array} \right\| \tag{5.9}$$

where n is a number of all Gaussians in all states, D is a dimensionality of Gaussian mean vectors, $\mu_{(i),k}$ is the k-th component of the i-th Gaussian mean vector.

This set of Gaussian mean vectors is divided into K subsets which are called "streams". The first stream is a subset of the first D_1 components of all n Gaussian mean vectors, the second stream is a subset of the next D_2 components of all n Gaussian mean vectors, and so on. An example of the definition of three streams of equal dimensionality is shown below:

$$\left\| \begin{array}{ccc} \mu_{(1),1} & \cdots & \mu_{(1),D_k} \\ \mu_{(2),1} & \cdots & \mu_{(2),D_k} \\ \cdots & \cdots & \cdots \\ \mu_{(n),1} & \cdots & \mu_{(n),D_k} \end{array} \right\|, \left\| \begin{array}{ccc} \mu_{(1),D_k+1} & \cdots & \mu_{(1),2D_k} \\ \mu_{(2),D_k+1} & \cdots & \mu_{(2),2D_k} \\ \cdots & \cdots & \cdots \\ \mu_{(n),D_k+1} & \cdots & \mu_{(n),2D_k} \end{array} \right\|, \left\| \begin{array}{ccc} \mu_{(1),2D_k+1} & \cdots & \mu_{(1),D} \\ \mu_{(2),2D_k+1} & \cdots & \mu_{(2),D} \\ \cdots & \cdots & \cdots \\ \mu_{(n),2D_k+1} & \cdots & \mu_{(n),D} \end{array} \right\| \tag{5.10}$$

where $K = 3$ is the number of streams and $D_k = D/K = D/3$ is the dimensionality of each stream.

In general case, the streams could be of different dimensionalities. In the following the special case is considered when all of the streams are of the same size. Thus, K streams of the same dimensionality $D_k = D/K$ are defined so that the first stream consists of first D_k dimensions, the second stream consists of the second D_k dimensions, and so on. After that the observation probabilities (see Equation 2.12) are calculated as shown in Equation 5.11.

$$B_s(x) = \sum_{m=1}^{M_s} C_{s,m} \prod_{k=1}^{K} N(x_k, \mu_{s,m,k}, \sigma) \tag{5.11}$$

where $N(x_k, \mu_{s,m,k}, \sigma)$ is a Gaussian stream probability from stream k with the diagonal covariance matrix of dimension D_k where all elements on the main diagonal are equal to σ^2; $x_k = (x_1, \ldots, x_{D_k})$ is a stream feature vector from stream k; $\mu_{s,m,k}$ is a Gaussian stream mean vector m of state s from stream k.

The graphical representation of streams is shown in Figure 5.1 where Gaussian mean vectors and their respective stream vectors are filled by the same pattern.

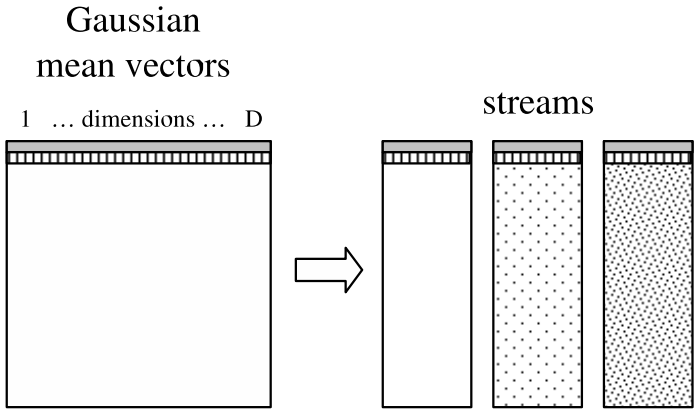

Figure 5.1: *Division of Gaussian mean vectors set into three streams*

5.3.2 Coding approach for SDCHMM

Now the clustering (see Section 2.2.2) and streaming techniques can be combined together and applied to HMM-based speech recognition. The set of stream vectors of the same dimensionality is obtained from the Gaussian mean vectors. Then the clustering is applied to each stream, and the codebooks for each stream are generated. All stream vectors are substituted by codes (pointers to codewords in the codebook). This combined approach reduces the memory consumption because codes require less bits to be stored than the stream vectors do.

The memory consumption in bits for the source CDHMM is shown in Equation 5.12:

$$mem = n \cdot D \cdot size \tag{5.12}$$

where n is a number of Gaussians; D is a dimensionality of Gaussians; $size$ is a size of numerical representation of a Gaussian component in bits.

The memory requirement in bits for the coded SDCHMM is defined as the sum of the memory for the code vectors $(K \cdot n \cdot \log_2 N)$ and the memory for the codebooks $(K \cdot N \cdot size)$:

$$mem' = K \cdot n \cdot \log_2 N + K \cdot N \cdot size \tag{5.13}$$

where K is a number of streams, all stream have the same dimensionality $D_k = D/K$; n is a number of Gaussians in the HMM set; N is a number of codebook vectors in each codebook; $\log_2 N$ is the number of bits that are necessary to code a codebook vector; $size$ is a size of numerical representation of a Gaussian mean vector component in bits.

5.3.3 Shared codebook approach for SDCHMM

In the considered SDCHMMs the stream structure is very simple (see Figure 5.2) and could be easy implemented in embedded systems. The fact that the streams are of the same size suggests the new approach with only one shared codebook for all streams [Astrov 2002; Varga et al 2002; Astrov et al 2003]. The shared codebook approach requires that the stream vectors satisfy the following two conditions: the streams have to be of the same dimensionality and the distributions of the stream vectors in all streams have to be similar. The similarity of the distributions of the stream vectors was shown in Section 5.1.

The SDCHMMs with shared codebook is obtained as it is shown in Figure 5.3. Firstly, Gaussian mean vectors are divided into K streams of equal dimensionality D_k. The merged stream is composed from all of the stream vectors of all streams. This merged stream is clustered, and the codebook is created. Then each stream vector is substituted by a pointer to the shared codebook.

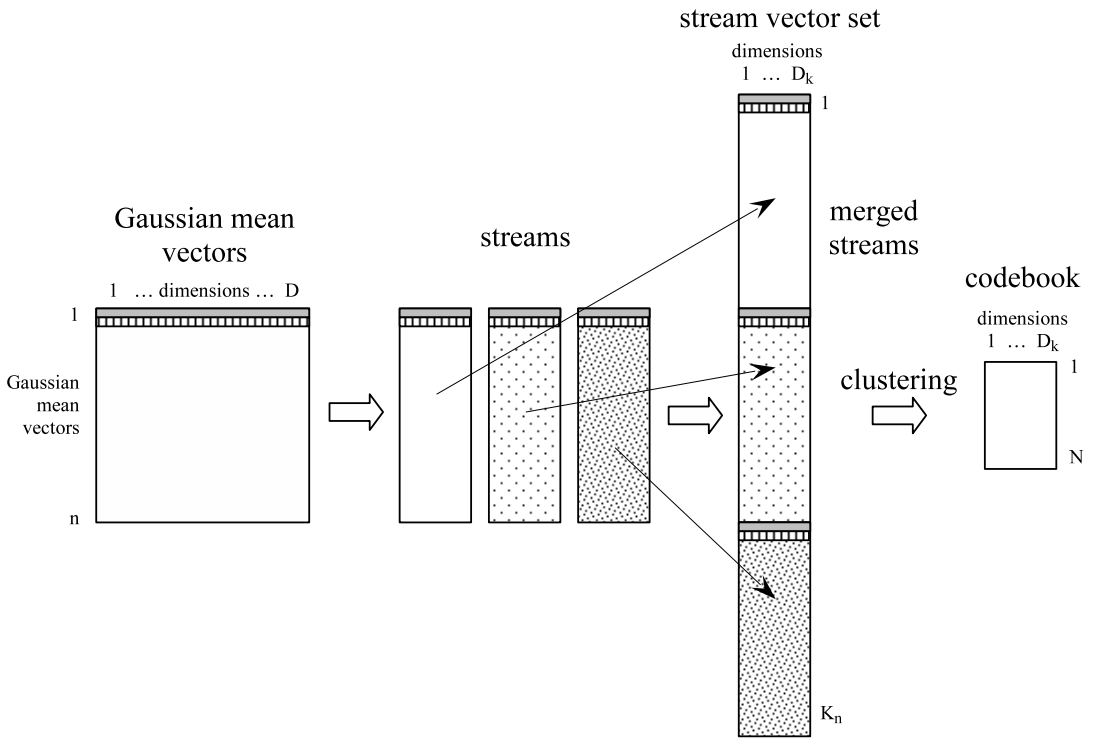

Figure 5.2: *Generation of shared codebook for SDCHMM set in case of three streams*

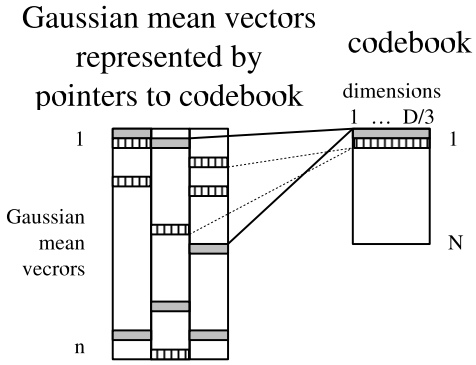

Figure 5.3: *Representation of a SDCHMM set with shared codebook in case of three streams*

5.4 Memory reduction for Gaussian weights

Besides the mean vectors, the Gaussian weights can also be compressed. In common HMMs considered in this work, the Gaussian weights $c_{s,m}$ satisfy the condition $0 \leq c_{s,m} \leq 4000$. Only 12 bits ($\log_2 4000 \simeq 12$) are needed to code the information for one weight. The

memory consumption in bytes for Gaussian weights is then estimated by Equation 5.14:

$$mem'_{weights} = \frac{12}{8}n = 1.5n \tag{5.14}$$

where n is a number of Gaussians in all states.

The task is to code weights by 8-bit integers, as such coding is easy to implement in embedded speech recognition systems. This problem can be solved using several coding schemes considered in the following.

"Square-root" coding

The recognizer calculates Gaussian scores (log likelihoods) according to Equation 2.28. Using the fact that the differences $(x_d - \mu_{s,m,d})^2$ are in the power of two, the following heuristic was investigated. To obtain 50% memory reduction it is necessary to code 16-bit values of $c_{s,m}$ with 8 bit integers: 8 bits are enough to code integer value $\hat{c}_{s,m} = int\left[\sqrt{c_{s,m}}\right]$. Thus, only one integer multiplication is required to decode the Gaussian weight. A Gaussian weight code occupies only 1 byte instead of 2 bytes. The required memory for all Gaussian weights in bytes is then:

$$mem'_{weights} = n \tag{5.15}$$

The memory reduction ratio in this case is exactly 50% for any HMM set:

$$ratio_{weights} = \frac{mem'_{weights}}{mem_{weights}} = \frac{n}{2n} = 0.5 \tag{5.16}$$

Scalar quantization

In the second approach the scalar quantization (see Section 2.2.1) is employed for coding of Gaussian weights. A codebook of $N = 256$ 2-byte values is used, thus, every Gaussian weight code occupies only 1 byte.

With the codebook of N codewords the required memory in bytes is shown in Equation 5.17:

$$mem'_{weights} = n + 2N \tag{5.17}$$

where $2N$ is the extra memory in bytes for the codebook.

The memory reduction ratio in this case is:

$$ratio_{weights} = \frac{mem'_{weights}}{mem_{weights}} = \frac{n + 2N}{2n} = 0.5 + \frac{N}{n} \tag{5.18}$$

For a typical case, $N = 256$, $n = 4000$, $ratio_{weights} = 0.564$.

Weight difference coding

In the third approach the differences between two Gaussian weights are coded. The sequence of Gaussian weights $c_{s,1}, c_{s,2}, \ldots, c_{s,m}$ within one state s could be coded recursively, as it is shown in Equation 5.19:

$$c_{s,i+1} = c_{s,i} + \alpha_s \cdot \Delta c_{s,i} \tag{5.19}$$

The next weight is calculated from the previous by adding the displacement $\alpha_s \cdot \Delta c_{s,i}$. Values $c_{s,i}$ are coded with 16 bits and $\Delta c_{s,i}$ with 8 bits. The integer multiplier α_s is needed because 8 bits could not be enough to code the difference $c_{s,i} - c_{s,i-1}$. To code each state s of HMM it is necessary to define instead of set of m Gaussian weights $\{c_{s,1}, c_{s,2}, c_{s,3}, \ldots, c_{s,m}\}$ the following set: $\{c_{s,1}, \alpha_s, \Delta c_{s,1}, \Delta c_{s,2}, \ldots, \Delta c_{s,m-1}\}$. Gaussian weights $c_{s,i}$ were 16-bit integers before the coding. In the new coded set only $c_{s,1}$ is a 16-bit integer, values α_s and differences $\Delta c_{s,i}$ are 8-bit integers. The maximum rounding error of Gaussian weights in current state is $\alpha_s/2$, and when $\alpha_s = 1$ the rounding error is equal to zero.

This approach is especially efficient when the differences $\Delta c_{s,i}$ are always positive or always negative, i.e. Gaussians within one prototype are sorted in ascending or descending order of their weights. As it will be shown in Section 6.3 the ascending order decreases also emission computation time.

After coding, the first weight code in each state occupies 3 bytes (2 bytes for $c_{s,1}$ and one byte for α_s). All other weight codes $\Delta c_{s,i}$ in this state occupy only 1 byte. The required memory in bytes is then:

$$mem'_{weights} = n + 2S \tag{5.20}$$

where S is a number of states.

The memory reduction ratio in this case is:

$$ratio_{weights} = \frac{mem'_{weights}}{mem_{weights}} = \frac{n + 2S}{2n} = 0.5 + \frac{S}{n} \tag{5.21}$$

For a typical case, $S = 118$, $n = 4000$, $ratio_{weights} = 0.530$.

Modified weight difference coding

It is possible to decrease the memory consumption further by selecting α_s equal for every state:

$$\alpha_s = \quad \alpha \qquad (s = 1, 2, \ldots, S) \tag{5.22}$$

Then it is necessary to store α only once for HMMs set. The memory consumption in bytes is then:

$$mem'_{weights} = n + S + 1 \tag{5.23}$$

where $n - S$ bytes are required to code the differences $\Delta c_{s,i}$; $2S$ bytes are required for $c_{s,0}$ components; and one byte is required for α. The memory reduction ratio in this case is:

$$ratio_{weights} = \frac{mem'_{weights}}{mem_{weights}} = \frac{n + S + 1}{2n} = 0.5 + \frac{S + 1}{2n} \tag{5.24}$$

For a typical case, $S = 118$, $n = 4000$, $ratio_{weights} = 0.515$.

The recognition error rates may be slightly higher in this modification than in the weight difference coding with different α_s.

5.5 Experiments

5.5.1 SDCHMM with 1-D streams

For 1-D streams a codebook with 16 different vectors should be used in order to reach 50% memory reduction. Each 8-bit component of Gaussian mean vectors is coded with 4 bits, i. e. represented by a 4-bit pointer.

Each 24-dimensional Gaussian mean vector is divided into 24 streams. The first stream consists of the first components of mean vectors, the second stream consists of only second components, etc. After that the streams are merged leading to one large array of stream vectors. Scalar quantization is applied and 16 codebook vectors are defined.

The distribution of the stream vectors was investigated in the following way (see also Section 5.1): a discrete function F was defined which is the number of counts of the values x of the stream vectors μ. Since the stream vectors are 8-bit integers the value x is in the range $-128 \leq x \leq 127$.

In Figure 5.4 it can be observed that $F(x) = 0$ for $-128 \leq x \leq -77$ and $73 \leq x \leq 127$. This is due to the properties of the acoustic models. For the coding therefore only the values $\mu_i = x$ with $F(x) > 0$ are considered. In this interval the set of 16 codebook vectors has to be found.

Let x_{min} and x_{max} denote the limits of an interval in a way that:

$$\begin{cases} F(x_{min}) > 0, \\ F(x) = 0, \quad x < x_{min} \end{cases} \tag{5.25}$$

$$\begin{cases} F(x_{max}) > 0, \\ F(x_{max}) = 0, \quad x > x_{max} \end{cases} \tag{5.26}$$

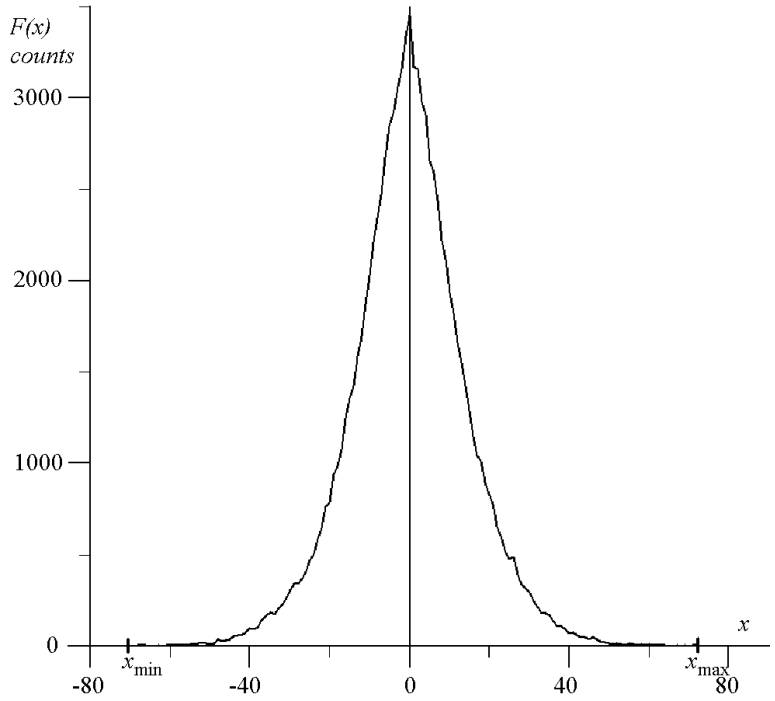

Figure 5.4: *Occurrence diagram of Gaussian mean vectors set plotted for one dimension*

In particular case shown in Figure 5.4 $x_{min} = -76$, $x_{max} = 72$. The interval $[x_{min} \dots x_{max}]$ is split into N sub-intervals:

$$\begin{gathered} I_j = [y_{j-1} ... y_j], \qquad j = 1, \dots, N \\ x_{min} = y_0 \leq y_1 \leq ... \leq y_{N-1} \leq y_N = x_{max} \end{gathered} \tag{5.27}$$

For each cluster one codebook vector has to be found. In order to obtain clusters and corresponding codebook vectors for the 1-byte streams the following approaches were investigated.

Uniform SQ: The interval $[x_{min} \dots x_{max}]$ was split into N intervals I_j of the same size:

$$I_j = \begin{cases} [y_{j-1}, y_j), & j = 1, 2, \dots, N-1 \\ [y_{N-1}, y_N], & j = N \end{cases} \tag{5.28}$$

$$y_j - y_{j-1} = \frac{x_{max} - x_{min}}{N} = const \tag{5.29}$$

Each stream vector from cluster I_j is mapped into a codebook vector v_j:

$$v_j = int \left[\frac{y_{j-1} + y_j}{2} \right] \tag{5.30}$$

In order to represent more frequent values with low distortion two scalar quantizers are developed. Intervals I_j were defined by the Lloyd-Max SQ. Two algorithms of the definition of codebook vectors are considered.

Lloyd-Max SQ, approach A (Lloyd-a): each codebook vector v_j is chosen to be the mean values of the interval I_j (see Equation 5.31). This is most close to the standard Lloyd-Max quantizer described in Section 2.2.1.

$$v_j = int\left[\frac{y_{j-1} + y_j}{2}\right] \tag{5.31}$$

Lloyd-Max SQ, approach B (Lloyd-b): v_j is a centroid of the interval I_j, it is defined such that amounts of stream vectors in interval $[y_{j-1}, v_j)$ and $[v_j, y_j,)$ are equal.

Tests were made with these three coding approaches "uniform", "Lloyd-a" and "Lloyd-b". HMM parameters are coded using SDCHMM approach with 1-D streams and the corresponding codebook. The codebook has 16 entries, thus, each stream vector is coded with 4 bits.

The memory reduction approach was tested on the following data (see Chapter 4 for detailed description):

- speech databases: SD-DE, SDII-DE, SieTill-DE, MoTiV-DE;
- test sets: SDI-1, SDI-2, SDII-mbl-apl, SDII-mbl-c_d, MoTiV, SieTill-c_d;
- HMM sets: TRAIN_S, TRAIN_U, TRAIN_V and TRAIN76.

The tasks were defined with these databases and HMMs, the results of recognition experiments are shown in Table 5.3. In the first column the task numbers are listed for further reference. In the next column combinations of test sets and HMM sets are shown; tasks 6 to 10, which were done on the MoTiV database, were performed with and without the spectral subtraction noise reduction from feature extraction module ("with NR" and "without NR"). Each task was tested without any coding and with the three coding approaches "uniform", "Lloyd-a" and "Lloyd-b". The memory consumption in bytes for each coding scheme is represented in the "memory" column. Word error rates are shown in the last five columns. The insertion, deletion and substitution rates are listed in columns "ins", "del" and "subst" respectively; WER is computed as a sum of all these error rates. The relative increase of WER is shown in the last column, a positive value denotes that the WER was increased in comparison to the recognition with the baseline HMM set, a negative value denotes the decrease of WER.

task	test set and HMM	coding scheme	memory [bytes]	word error rate [%]				increase of WER, [%]
				subst	del	ins	WER	
1	SDI-1 TRAIN_V	no coding	28776	1.7	0.3	1.0	3.0	
		uniform	14404	1.8	0.2	1.3	3.3	7.4
		Lloyd-a	14404	2.2	2.5	0.3	5.0	64.8
		Lloyd-b	14404	2.0	0.3	1.4	3.7	20.4
2	SDI-2 TRAIN_V	no coding	28776	1.4	0.5	0.4	2.4	
		uniform	14404	1.4	0.5	0.8	2.7	15.2
		Lloyd-a	14404	1.8	1.6	0.2	3.7	54.5
		Lloyd-b	14404	1.4	0.4	0.7	2.6	5.3
3	SDII-mbl-apl TRAIN_S	no coding	96000	6.3	0	0	6.3	
		uniform	48016	7.4	0	0	7.4	17.4
		Lloyd-a	48016	7.5	0	0	7.5	19.2
		Lloyd-b	48016	7.0	0	0	7.0	10.9
4	SDII-mbl-apl TRAIN_U	no coding	96000	4.8	0	0	4.8	
		uniform	48016	5.4	0	0	5.4	13.8
		Lloyd-a	48016	6.1	0	0	6.1	27.1
		Lloyd-b	48016	4.9	0	0	4.9	3.3
5	SDII-mbl-c_d TRAIN_V	no coding	28776	2.1	2.0	0.9	4.9	
		uniform	14404	2.3	1.3	1.5	5.1	4.2
		Lloyd-a	14404	3.4	9.0	0.2	12.7	156.7
		Lloyd-b	14404	2.3	1.5	1.4	5.1	4.2
6	MoTiV TRAIN_S without NR	no coding	96000	36.5	0	0	36.5	
		uniform	48016	35.9	0	0	35.9	-1.7
		Lloyd-a	48016	50.3	0	0	50.3	37.7
		Lloyd-b	48016	34.8	0	0	34.8	-4.7
7	MoTiV TRAIN_S with NR	no coding	96000	21.1	0		21.1	
		uniform	48016	21.3	0	0	21.3	0.7
		Lloyd-a	48016	31.8	0	0	31.8	50.8
		Lloyd-b	48016	20.1	0	0	20.1	-4.9
8	MoTiV TRAIN_U without NR	no coding	96000	37.2	0	0	37.2	
		uniform	48016	37.0	0	0	37.0	-0.6
		Lloyd-a	48016	52.3	0	0	52.3	40.7
		Lloyd-b	48016	35.8	0	0	35.8	-3.8
9	MoTiV TRAIN_U with NR	no coding	96000	20.8	0	0	20.8	
		uniform	48016	21.4	0	0	21.4	2.6
		Lloyd-a	48016	32.9	0	0	32.9	57.9
		Lloyd-b	48016	20.2	0	0	20.2	-3.1
10	SieTill-c_d TRAIN_V	no coding	28776	1.6	1.7	0.3	3.5	
		uniform	14404	1.7	1.3	0.4	3.4	-4.3
		Lloyd-a	14404	3.6	8.7	0.1	12.4	249.2
		Lloyd-b	14404	2.2	1.7	0.5	4.4	23.7
11	SDII-c_d TRAIN76	no coding	30168	1.2	0.1	0	1.3	
		uniform	15100	1.5	0	0	1.6	20.1
		Lloyd-a	15100	3.6	1.0	0	4.6	248.3
		Lloyd-b	15100	1.3	0	0	1.3	0
12	SDII-mbl-c_d TRAIN76	no coding	30168	2.1	2.0	0.9	4.9	
		uniform	15100	2.3	1.3	1.5	5.1	4.2
		Lloyd-a	15100	4.5	11.7	0.3	16.5	233.7
		Lloyd-b	15100	3.3	2.0	1.5	6.7	36.0

Table 5.3: *Recognition results with the SDCHMM with 1-D streams*

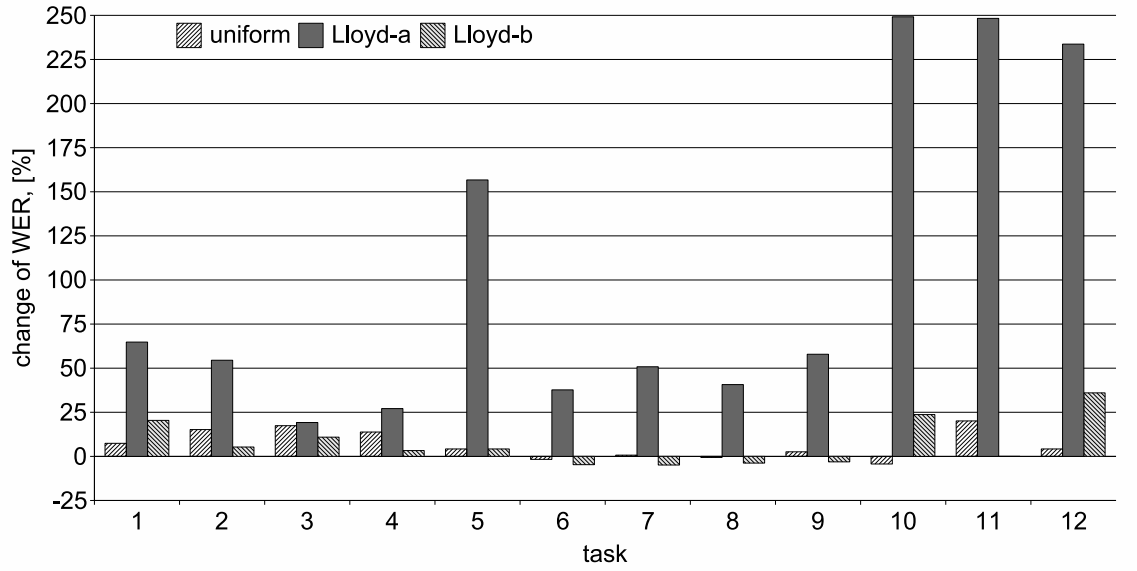

Figure 5.5: *Relative change of WER of coding approaches on 12 different tasks*

The results of the tests are also shown in Figure 5.5. 50% reduction of memory consumption by Gaussian mean vectors was reached on every task with every investigated coding approach.

The "Lloyd-a" coding always leads to highest increase in word error rates. "Lloyd-b" and "uniform" coding schemes lead to good results on most of the tasks. The "Lloyd-b" approach has relative increase of WER more than 10% on tasks 1, 3, 10 and 12, on tasks 6, 7, 8 and 9 the recognition accuracy is increased.

The "uniform" coding on tasks 1, 10 and 12 leads to a significantly better result than "Lloyd-b" coding, on tasks 2, 3, 4 and 11 it does not meet the target of a relative increase in word error rates of less than 10%.

As it has been observed in tasks 6, 7, 8 and 9 the HMMs coded by "Lloyd-b" algorithm lead to lower WERs than the baseline HMMs. The TRAIN_U and TRAIN_S HMMs are trained using telephone speech databases, but the recognition was performed on MoTiV database recorded in car environment. Thus the coding may bring some low improvement in case of mismatch of training and test task environment.

None of the 1-D stream coding approaches reached the goal of relative increase of WER less than 10% on all tasks. The increase of the codebook size to 32 or 64 codewords will improve the recognition performance, but the memory reduction will be less than 50%. The multidimensional streams may solve this problem, they will be considered in the following.

5.5.2 SDCHMM with multidimensional streams

In the following the memory reduction approaches described in Section 5.3.1 were experimentally investigated. Speech recognition tests were made for German and Spanish languages, see Chapter 4 for detailed descriptions of tasks and HMM sets. The results of experiments are shown in Table 5.4. German TRAIN_U HMM set was tested with SDII-mbl-apl and MoTiV tasks, "NR" in MoTiV NR task denotes that the noise reduction in preprocessing was used. AppW, Spell and Digits tasks were tested with Spanish TRAIN_Q HMM set. SDI-1, SDI-2, SDII-mbl-c_d and SieTill-c_d tasks were tested with German TRAIN_V HMM set.

German TRAIN_U HMM

stream dim	codebook type	memory		WER [%]					
		bytes	%	SDII-mbl-apl		MoTiV		MoTiV NR	
	baseline	96000	100%	4.8	rel%	37.2	rel%	20.8	rel%
4	shared	25024	26.1%	6.2	+30.1	34.8	-6.5	20.7	-0.4
4	independent	30114	31.4%	6.0	+18.6	35.8	-3.8	20.8	+0.2
3	shared	32768	34.1%	4.9	+1.9	37.3	+0.4	21.1	+1.5
3	independent	38144	39.7%	5.5	+13.8	37.6	+1.2	21.2	+1.8
2	shared	48512	50.5%	4.9	+2.9	36.2	-2.6	20.7	-0.4
2	independent	54144	56.4%	4.7	-1.4	37.4	+0.5	20.8	0
1	shared	48016	50.0%	4.9	+3.3	35.8	-3.8	20.2	-3.1

Spanish TRAIN_Q HMM

stream dim	codebook type	memory		WER [%]					
		bytes	%	AppW		Spell		Digits	
	baseline	96000	100%	0.8	rel%	31.7	rel%	1.6	rel%
4	shared	25024	26.1%	0.9	+18.2	34.2	+8.0	2.0	+25.0
4	independent	30144	31.4%	1.6	+100	35.1	+10.7	1.4	-12.5
3	shared	32768	34.1%	0.9	+18.2	33.5	+5.6	1.6	0
3	independent	38144	39.7%	1.0	+27.3	32.7	+3.1	1.6	0
2	shared	48512	50.5%	0.7	-9.1	32.8	+1.1	1.8	+12.5
2	independent	51444	56.4%	0.7	-9.1	31.7	0	1.8	+12.5
1	shared	48016	50.0%	1.1	+36.4	32.0	+0.9	1.4	-12.5

German TRAIN_V HMM

stream dim	codebook type	memory		WER [%]							
		bytes	%	SDI-1		SDI-2		SDII-mbl-c_d		SieTill-c_d	
	baseline	28776	100%	3.0	rel%	2.4	rel%	4.9	rel%	3.8	rel%
4	shared	8218	28.6%	3.2	+5.6	2.4	0	5.5	+11.5	4.0	+6.4
4	independent	13378	46.4%	3.1	+3.7	2.3	-3.0	5.0	+2.3	3.9	+2.4
3	shared	10360	36.0%	3.1	+1.9	2.5	+6.1	5.4	+9.2	4.0	+4.8
3	independent	15736	54.7%	3.1	+3.7	2.4	0	5.0	+2.3	3.9	+2.8
2	shared	14900	51.8%	2.9	-1.9	2.1	-12.1	4.9	-0.8	3.8	+1.2
2	independent	20532	73.2%	3.0	0	2.4	0	5.0	+2.3	3.8	-0.2
1	shared	14404	50.1%	3.7	+20.4	2.6	+5.3	5.1	+4.2	4.4	+23.7

Table 5.4: *Test results with multidimensional streams*

In the first column the stream dimensionalities are shown. Preliminary tests have shown that streams of dimension more than 4 with a codebook size of 256 vectors lead to high degradation of recognition rate. In the second column the types of codebook (shared or independent for each stream) are shown. The codebooks have 256 vectors for 2-D, 3-D and 4-D streams, and 16 vectors for 1-D streams with "Lloyd-b" coding scheme.

In the third column the required memory to store Gaussians mean vectors with codebooks is shown. The last columns show WERs for different recognition tasks. Values for the baseline are obtained by doing the recognition with continuous density HMMs. The memory size is shown in bytes and in percents relative to the baseline. For the SDCHMMs the WERs are shown as absolute values and relative differences to the baseline WER.

As can be observed in Table 5.4, the recognition results of SDCHMMs with independent and shared codebooks are almost similar. Independent codebooks require more memory as shared codebooks. The smaller the HMM the more memory (relatively) is required for separate codebooks.

In case of TRAIN_V HMM set the memory consumption of SDCHMMs with 2-D streams and independent codebooks is 41% higher than that of SDCHMMs with 2-D streams and the shared codebook. The SDCHMMs with 2-D streams and the shared codebook occupies less memory and has better recognition performance than the SDCHMMs with 3-D streams and independent codebooks.

In all SDCHMMs the codebooks have 256 vectors because this leads to 1-byte indexes. The indexes are thus easy to handle. For multidimensional streams and constant codebook size (256 vectors) the quantization error increases with the increase of dimensionality, and thus the higher increase of WER can be expected. In the results shown in Table 5.4 the relative increase of WERs in 2-D streams SDCHMMs with shared codebook is less than 12.5%. 3-D streams approach leads to maximum 18.2% increase of WER. 4-D streams approach leads to 30.1% increase of WER.

The increase of WER of feature parameter tying HMMs (1-D stream SDCHMMs) with 16 codebook vectors is higher than the WER increase of 2-D streams SDCHMMs, as the precision of scalar quantization is less than that of vector quantization. The MoTiV, MoTiV NR, Spell and Digits tasks are exceptions. MoTiV and MoTiV NR tasks are "mismatch" tasks with different training and test conditions, thus higher distortions of acoustic models may lead to a better match to the test data. The improvement of the recognition accuracy occurs mostly on tasks with high baseline WERs.

The AppW and Digits tasks for Spanish TRAIN_Q HMM set have high relative increase of WER, but these results are not relevant as in AppW task only 11 of 1414 words were recognized wrongly, in Digits task only 8 of 483 words were recognized wrongly. The wrong recognition of only one extra word in Digits task leads to increase of WER up to 12.5% relatively.

5.5.3 Reduction of memory consumption by Gaussian weights

The approaches described in Section 5.4 are tested experimentally. In Table 5.5 the test results for 4 different coding techniques for Gaussian weights are shown: SQ denotes scalar quantization approach with codebook of 256 codewords; SQRT denotes "square root" coding approach, "deviation-1" denotes the coding technique of weight differences; "deviation-2" denotes the modified weight differences coding approach. The tests were performed using TRAIN_V, TRAIN_U and TRAIN_S HMM sets (see Chapter 4 for detailed description).

In the first column the combinations of HMM sets and test sets are shown. The experiments with the MoTiV task were performed with and without the noise reduction from feature extraction module ("MoTiV NR" and "MoTiV"). The "weights memory" column shows the memory consumption in bytes by Gaussian weights (including codebook, if necessary) and the relative memory consumption, the baseline is the HMM set without any coding.

Word error rates are shown in the last five columns. The insertion, deletion and substitution rates are listed in columns "ins", "del" and "subst", respectively; WER is computed as a sum of all these error rates. The relative increase of WER is shown in the last column, a positive value denotes that WER was increased in comparison to the recognition with the baseline HMM set (without coding), a negative value denotes the decrease of WER.

The coding of Gaussian penalties leads to distortion of penalty values. On some tasks distortion leads to better recognition rate, but this is not the property of the coding. This effect occurs due to the mismatch in train and test conditions.

In Table 5.5 it is observed that WER of HMMs with coding of weight differences ("deviation-1" and "deviation-2" coding) is almost the same as the baseline WER (no coding). This means that coded and then decoded Gaussian weights are very close to Gaussian weights of the baseline HMMs.

In some cases the weight difference coding schemes may increase memory consumption. Such case is the HMM set where each state is modeled by one Gaussian. Then the number of states S is equal to the number of Gaussians n. Equation 5.21 gives $ratio_{weights} = 1.5$ and Equation 5.24 gives $ratio_{weights} \simeq 1$. In small context-dependent HMM sets for embedded devices one state at the average may be modeled by two Gaussians. In this case it is advisable to use other coding schemes.

The SQ approach codes weights with errors, that leads to higher difference between WER of the coded HMM set and of the baseline HMM set. The SQRT coding scheme is recommended for the implementation in embedded systems because the decoding procedure is very simple (only one integer multiplication per Gaussian weight) and no extra memory is required for a codebook.

coding scheme	coding scheme	weights memory		error rates				
		bytes	relative %	subst	del	ins	WER	relative
	no coding	2398	100	1.7	0.3	1.0	3.0	
TRAIN_V	SQ	1711	71.4	1.7	0.3	1.0	3.0	0
	SQRT	1199	50.0	1.7	0.3	1.0	3.0	0
SDI-1	deviation-1	1675	69.8	1.7	0.3	1.0	3.0	0
	deviation-2	1438	60.0	1.7	0.3	1.0	3.0	0
	no coding	2398	100	1.4	0.5	0.4	2.4	
TRAIN_V	SQ	1711	71.4	1.4	0.5	0.4	2.3	-3.0
	SQRT	1199	50.0	1.4	0.5	0.4	2.4	0
SDI-2	deviation-1	1675	69.8	1.4	0.5	0.4	2.4	0
	deviation-2	1438	60.0	1.4	0.5	0.4	2.3	-3.0
	no coding	2398	100	2.1	2.0	0.9	4.9	
TRAIN_V	SQ	1711	71.4	2.1	2.0	0.9	5.0	+0.4
	SQRT	1199	50.0	2.1	2.0	0.9	4.9	0
SDII-mbl-c_d	deviation-1	1675	69.8	2.1	2.0	0.9	5.0	+0.4
	deviation-2	1438	60.0	2.1	2.0	0.9	5.0	+0.4
	no coding	2398	100	10.2	1.9	0.3	12.4	
TRAIN_V	SQ	1711	71.4	10.2	1.9	0.3	12.4	0
	SQRT	1199	50.0	10.2	1.9	0.3	12.4	-0.1
SieTill-c_d	deviation-1	1675	69.8	10.2	1.9	0.3	12.4	0
	deviation-2	1438	60.0	10.2	1.9	0.3	12.4	0
	no coding	8000	100	4.8	0.0	0.0	4.8	
TRAIN_U	SQ	4512	56.4	4.9	0.0	0.0	4.9	+1.9
	SQRT	4000	50.0	4.9	0.0	0.0	4.9	+1.9
SDII-mbl-apl	deviation-1	4236	53.0	4.8	0.0	0.0	4.8	0
	deviation-2	4119	51.5	4.8	0.0	0.0	4.8	0
	no coding	8000	100	6.3	0.0	0.0	6.3	
TRAIN_S	SQ	4512	56.4	7.2	0.0	0.0	7.2	+14.9
	SQRT	4000	50.0	6.4	0.0	0.0	6.4	+1.8
SDII-mbl-apl	deviation-1	4236	53.0	6.3	0.0	0.0	6.3	0
	deviation-2	4119	51.5	6.3	0.0	0.0	6.3	+0.4
	no coding	8000	100	37.2	0.0	0.0	37.2	
TRAIN_U	SQ	4512	56.4	33.1	0.0	0.0	33.1	-11.1
	SQRT	4000	50.0	36.5	0.0	0.0	36.5	-2.0
MoTiV	deviation-1	4236	53.0	37.2	0.0	0.0	37.2	+0.1
	deviation-2	4119	51.5	37.1	0.0	0.0	37.1	-0.2
	no coding	8000	100	20.8	0.0	0.0	20.8	
TRAIN_U	SQ	4512	56.4	19.1	0.0	0.0	19.1	-8.5
	SQRT	4000	50.0	20.7	0.0	0.0	20.7	-0.9
MoTiV NR	deviation-1	4236	53.0	20.8	0.0	0.0	20.8	0
	deviation-2	4119	51.5	20.9	0.0	0.0	20.9	+0.2
	no coding	8000	100	36.5	0.0	0.0	36.5	
TRAIN_S	SQ	4512	56.4	31.8	0.0	0.0	37.8	-12.9
	SQRT	4000	50.0	35.9	0.0	0.0	35.9	-1.6
MoTiV	deviation-1	4118	53.0	36.5	0.0	0.0	36.5	+0.1
	deviation-2	4119	51.5	36.5	0.0	0.0	36.5	0
	no coding	8000	100	21.1	0.0	0.0	21.1	
TRAIN_S	SQ	4512	56.4	18.7	0.0	0.0	18.7	-11.3
	SQRT	4000	50.0	21.0	0.0	0.0	21.0	-0.7
MoTiV NR	deviation-1	4236	53.0	21.1	0.0	0.0	21.1	0
	deviation-2	4119	51.5	21.1	0.0	0.0	21.1	0

Table 5.5: *Memory reduction for Gaussian weights: tests results for German TRAIN_U and TRAIN_S HMM sets*

5.6 Conclusion

Table 5.6 shows the relative change in WER and the required memory in bytes to store parameters of the HMM sets with 4000 Gaussians (TRAIN_Q, TRAIN_S and TRAIN_U). The HMMs were represented as SDCHMMs with one shared codebook. The 1-D SDCHMMs use the "Lloyd-b" coding scheme to obtain the codebook with 16 codewords, the 2-D and 3-D schemes use VQ with shared codebook with 256 code vectors. The Gaussian weights were coded with the SQRT coding scheme. In case of 1-D or 2-D streams about 52 kilobytes of memory is required. In case of 3-D streams about 37 kilobytes are necessary.

The WER change is shown for HMM sets with different mean vector coding schemes. The results obtained for TRAIN_Q, TRAIN_S and TRAIN_U HMM sets are supplemented with WERs obtained using smaller HMM sets of 1200 Gaussians (TRAIN_V, TRAIN_76). The coding of Gaussian weights leads to very low changes in the recognition accuracy, the change of WER induced by coding of weights is not considered in the results shown in this table. The 1-D streams approach results to relative increase of WER up to 36%, this coding scheme does not satisfy the objectives of the research listed in Section 3.4. The 2-D streams lead to 2.9% increase of WER, the 12.5% increase was only observed in case of Digits task with very low baseline WER (false recognition of one word leads to 12.5% relative increase of WER). The 3-D approach lead to 9.2% relative increase of WER, the 18.2% increase was only observed in case of AppW task with very low baseline WER.

parameter	baseline	1-D streams	2-D streams	3-D streams
Gaussians mean vectors	96000	48000	48000	32000
Gaussians weights	8000	4000	4000	4000
codebook for mean vectors	-	16	512	768
total memory	104000	52016	52512	36768
relative WER change, %		-4.9...+36.0	-12.1...+2.9	0...+9.2

Table 5.6: *Comparison of memory requirements in bytes and WERs for CDHMMs (baseline) and SDCHMMs with different stream sizes*

In this chapter the memory saving coding of HMM parameters was investigated. The reduction of memory consumption by HMM parameters was achieved by using 2-D and 3-D streams with shared codebook approach for Gaussian mean vectors and "SQRT" coding of Gaussian weights. The memory consumption by coded HMM set is three times lower than by the baseline HMM set. The WERs for coded HMM set are increased up to 10% relatively, which meets the objectives of the research (see Section 3.4).

Chapter 6

Fast emission computation approaches

In a small vocabulary (30-100 words) speech recognizer the emission computation procedure consumes most of processing power. In this chapter the problem of the fast emission computation is considered in detail. The emission computation algorithm was described in Section 2.1.5. Based on vector quantization and SDCHMM structure, several fast emission computation algorithms are explored. The acceleration of the emission computation is based on the following principles:

1. The SDCHMM approach described in Chapter 5.3.1 is robust and has reduced memory requirements. The streams approach accelerates the emission computation by reusing of computed results stored in memory.

2. The approximation and incomplete computation of emission probabilities are combined and the new method is developed. This approach is considered in Section 6.2. The approximation leads to the increase of WER. The task is to design such fast computation approach that the relative WER increase is less than 10%. The combination of the fast emission computation and SDCHMM structure is explored.

3. The special order of HMM parameters in memory explored in this work can lead to the acceleration of emission probabilities. Such particular data structure does not reduce information about HMM parameters and does not reduce recognition accuracy. Moreover, the reorganization in a special way brings more information about data structure and this feature is used by memory reduction for Gaussian weights. This approach is considered in Section 6.3.

6.1 Fast emission computation for SDCHMM

In SDCHMM the observation probability of the feature vector x in the state s is calculated according to Equation 5.11 which is rewritten below:

$$B_s(x) = \sum_{m=1}^{M_s} C_{s,m} \prod_{k=1}^{K} N(x_k, \mu_{s,m,k}, \sigma) \tag{6.1}$$

The negative log likelihood (score) is computed in SDCHMM as shown in Equation 6.2 (see Equations 2.19 and 2.28 for emission computations in CDHMM).

$$b_s(x) = \min_m \left\{ c_{s,m} + \sum_{k=1}^{K} \left(\sum_{d_k=1}^{D_k} (x_{k,d_k} - \mu_{s,m,k,d_k})^2 \right) \right\} + const \tag{6.2}$$

where d_k is the index parameter for stream k; D_k is the dimensionality of stream k; μ_{s,m,k,d_k} is a d_k-th component of the stream mean vector $\mu_{s,m,k}$.

In SDCHMMs Gaussian mean vectors are coded by the limited amount of stream vectors from the codebook. Taking in account this feature, all stream Gaussian log likelihoods can be precomputed once for current frame and their values can be stored in a look-up tables. In [Aiyer et al 2000] the distance between feature stream vector and Gaussian stream vector (Equation 6.3) is called "atom":

$$G_{s,m,k} = \sum_{d_k=1}^{D_k} (x_{k,d_k} - \mu_{s,m,k,d_k})^2 \tag{6.3}$$

In the stream k the stream mean vectors $\{\mu_{s,m,k,1}, \ldots, \mu_{s,m,k,d_k}\}$ are coded using a codebook with N codewords, that is why stream vectors are limited by the N possible values, i.e. for a current stream feature vector $\{x_{k,1}, \ldots, x_{k,d_k}\}$ exist only N atoms. For K streams and N atoms per stream, the atoms fill the $K \times N$ table $\mathbf{G}$. This table in [Aiyer et al 2000] is called "atom table". The component $G_{s,m,k}$ is defined as

$$G_{s,m,k} = \sum_{d_k}^{D_k} (x_{k,d_k} - \nu_{k,i,d_k})^2, \quad (i = 1, \ldots, N, \quad k = 1, \ldots, K) \tag{6.4}$$

where stream mean vector $\mu_{s,m,k}$ is represented by a codebook vector $\nu_{k,i}$; ν_{k,i,d_k} is a d_k-th component from the stream k of the i-th codebook vector.

The negative log likelihood is computed as shown in Equation 6.5.

$$b_s(x) = \min_m \left\{ c_{s,m} + \sum_{k=1}^{K} G_{s,m,k} \right\} + const \tag{6.5}$$

The acceleration of the emission computation using 3-D streams approach is experimentally explored on the baseline embedded system with 100 MHz processor, see Section 3.3. In the benchmark a 5 seconds long utterance and a vocabulary of 30 words were used [As-

trov et al 2003]. Table 6.1 shows minimum computational requirement expressed in real time factor for emission probability calculation and Viterbi search. The results are shown for recognition using CDHMMs (baseline) and SDCHMMs with 3-D streams. The feature extraction is not considered here, it is assumed to be implemented on DSP. The dependency of the processor load factor on the system configuration, e.g. cache size and memory access times, are clearly visible. ARM9TDMI with zero-wait states memory is reference for maximum performance. ARM920T core has higher cache size than the ARM940T core, that is why the performance of ARM920T core is better.

In the experiments CDHMM parameters are compressed using a 3-D stream based coding. The HMM acoustic models take up only 12 kilobytes of flash memory storage. On the ARM920T microcontroller the computation of the emission probabilities and the Viterbi search with a vocabulary of 30 words runs with 0.17 real time factor.

computation task	computational requirements, RTF		
	ARM9TDMI	ARM920T	ARM940T
Viterbi search	0.083	0.091	0.135
emission CDHMM (baseline)	0.262	0.289	0.303
emission SDCHMM 3-D	0.077	0.082	0.127

Table 6.1: *Minimal computational requirements in real time factor for a speech recognition with a 30-word vocabulary*

6.2 Fast emission computation using vector quantization

The computation of log likelihoods for a given state requires processing of all Gaussians within this state. Some of Gaussian probabilities may be approximated by the pre-computed values or constants without loss of recognition accuracy, these probabilities does not require exact computation. The log likelihoods are calculated precisely for a small portion of Gaussians (about 5-25%) with mean vectors that are placed close to current feature vector. For other Gaussians which are placed far away from the feature vector the log likelihoods are approximated [Bocchieri 1993; Haeb-Umbach and Ney 1991]. The method can be considered as the combination of two HMM sets: one rough HMM set and one precise HMM set.

Such approximation is done using the clustering approach which was described in Section 2.2.2. All Gaussian mean values in all states are broken into several clusters. For each cluster their mean vectors are obtained.

Firstly, the log likelihoods for all cluster mean vectors are computed. Then the developed algorithm has to define Gaussians which log likelihoods have to be calculated precisely and Gaussians which log likelihoods have to be approximated. This can be done in several ways:

1. The log likelihood has to be computed exactly if the log likelihood of the cluster mean value is less then the predefined threshold. This algorithm is very simple and can be easily realized. The preliminary experiments have shown that for this algorithm the threshold has to be high: in average more then a half of all Gaussian log likelihoods have to be calculated precisely in order to have an acceptable WER. For a feature vector x_1 with a low absolute value $|x_1|$ the threshold has to be low in order to compute the log likelihoods fast. On the other hand for a feature vector x_2 with a high absolute value $|x_2|$ the threshold has to be high such that log likelihoods for more then one cluster will be computed exactly. A possible decision could be a varying threshold that depends on the current feature vector.

2. The log likelihood has to be computed exactly for a small portion of clusters. This approach requires a special type of sorting procedure, that finds cluster mean vectors placed close to the current feature. This sorting is performed after or during the log likelihood computation for cluster mean vectors.

The possible ways of log likelihood approximation are listed below:

1. Log likelihoods can be approximated by experimentally obtained constant value for the current HMM set.

2. The more accurate approach is the approximation of log likelihoods by cluster mean value log likelihood. This approach requires that all of the computed cluster mean vector log likelihoods are stored in a memory array.

6.3 Optimization of data placement in memory

In emission computation algorithm shown in Section 2.1.5 the calculation times for different emissions are not equal. For a given state s and given feature vector the emission computation time depends on the following factors:

- the Gaussian weights within state s;
- the feature vector and the Gaussian mean vectors;
- the “previous history”: the value of ”best score” variable. The smaller is the value of ”best score”, the less calculations will be performed. The earlier the best Gaussian is found within the HMM state, the less computations are performed because more log likelihoods computations will be abandoned (see Section 2.1.5 for detail description).

The order of Gaussians within one state defines how fast the ”best score” will be found but it does not influence the recognition performance. The computational time is reduced if most often selected for ”best score” Gaussians are placed on first positions within each state. This problem cannot be solved for all HMMs and test sets, a simple method is used to accelerate the emission computation. The order of Gaussians within one state is set such that firstly the scores are computed for Gaussians with lowest weights.

Three different structures of HMM parameter placement in memory were tested. The first HMM set (baseline) was obtained after training, no sorting was made. In the second HMM set, Gaussians are sorted in ascending order, i.e. firstly Gaussian log likelihoods with lowest weights are computed. In the third HMM set, Gaussians are sorted in descending order of their weights, scores are computed firstly for Gaussians with highest weights. The SDII-mbl-apl test set was recognized with TRAIN_U and TRAIN_S HMM sets.

The recognition was performed on the workstation, see Section 3.3 for system description. During the recognition of SDII-mbl-apl task the scores for 66 034 495 Gaussians were computed for TRAIN_U HMM set. The recognition with TRAIN_S required computations of 87 163 180 Gaussian scores.

HMM set	sorting order	distances		recognition time	
		absolute	relative	absolute	relative
TRAIN_U	no sorting	28 345 512 556		23m21.230s	
	ascending	21 192 341 694	-25.2%	20m38.270s	-11.6%
	descending	28 952 635 379	+2.1%	23m43.650s	+1.6%
TRAIN_S	no sorting	32 551 236 063		32m46.900s	
	ascending	28 047 106 902	-13.8%	30m31.960s	-6.9%
	descending	36 234 739 559	+11.3%	33m31.520s	+2.3%

Table 6.2: *Differences in the recognition times for different data organizations within HMM sets with SDII-mbl-apl task*

In Table 6.2 the test results are shown. The column “distances” shows how many distances such as $(x_d - \mu_{s,m,d})^2$ were computed. The column “recognition time” shows the user execution time in minutes and seconds obtained by using UNIX “time” shell command, the recognition time was measured for feature extraction, emission computation and the search together.

The relative change of the number of distances and execution time shows the change relatively to the “no sorting” baseline HMM. Negative values in Table 6.2 mean that the recognition time and the number of computed distances were reduced.

As expected, the lowest recognition time was for the HMM sets with Gaussians sorted in ascending order of their weights within one state. The highest recognition time was measured for the HMM sets with Gaussians sorted in descending order of their weights within one state.

6.4 Combined methods

Three methods for reduction of the computation time (VQ-based, SDCHMMs, reorganization of data) could be combined together. In order to explore the effect of the combination of these techniques the VQ-SDCHMM combination was explored in detail. The reorganization of data leads to no changes in the recognition accuracy that is why this technique was not experimentally explored in detail.

In the following the VQ-based fast emission computation procedure was combined with the stream structure of SDCHMM. Firstly, the VQ approach was applied to the HMM set, the clusters were defined and their cluster mean vectors were built. Then the streaming procedure was applied to the Gaussian mean vectors and also to the cluster mean vectors. Thus the memory consumption was reduced, this algorithm is shown below:

Fast emission computation algorithm based on VQ and SDCHMM combination

1. Clustering: obtain cluster mean vectors
2. Streaming: apply streaming procedure to the set of Gaussians and to cluster mean vectors
3. Codebook generation: create a shared codebook using as training data Gaussian mean vectors and mean values of stream vectors
4. Coding: build SDCHMMs, code Gaussian mean vectors and cluster mean vectors

---

The TRAIN_U HMM set was tested on the workstation with SDII-mbl-apl task. The test set has 4391 utterances consisting of 941 289 frames, this is about 4 hours of speech. During the experiments the following parameters were changed:

1. Stream dimension: 2-D and 3-D streams were tested because these coding schemes do not heavily affect on recognition rate.
2. Number of clusters: the set of Gaussians was broken onto 64 or 256 clusters.
3. The number of clusters which log likelihoods were computed exactly was changed. The relative number of clusters was in the range from 0 to 100%.

For each experiment WERs were obtained. The execution time was measured using emission computation program which computes only emission probabilities.

The results for TRAIN_U HMM set with 4000 Gaussians are shown in Figure 6.1. The curves have notations $CLUSTERS/DIM$ where $CLUSTERS$ is the number of clusters for VQ-based fast emission computation, DIM is the dimensionality of streams. For example, "64/3" means that for fast emission computation 64 clusters were used and the stream dimension is 3 (3-D streams).

In Figure 6.1a the computation time for the combination of the VQ-based fast emission computation and SDCHMM is shown. The horizontal line "baseline" denotes the time necessary for emission computation of HMMs without streams and VQ approaches. Curves "baseline 2-D" and "baseline 3-D" show the time necessary for emission computation without VQ-based fast emission computation using 2-D and 3-D streams approaches respectively.

The axis "clusters with exactly computed log-likelihoods" (emission approximation factor) denotes for how many clusters the emissions were computed exactly. "25%" means that the emissions were computed for 25% of the clusters, for example, for 16 of 64 clusters or for 64 of 256.

The value "100%" means that the log likelihoods for all Gaussian mean vectors were computed exactly. In this case the VQ-based fast emission computation with stream approach needs more time than the computation without VQ approach, as extra time is required for computation of cluster mean vectors log likelihoods.

Fast emission computation with 64 clusters requires less time than with 256 clusters since for the same emission approximation factor less cluster mean vector likelihoods have to be computed.

Figure 6.1b shows WER over the emission approximation factor. The vertical axis is the WER in percents and the horizontal axis is the emission approximation factor. The horizontal line "baseline" shows WER of HMMs without streams and VQ approaches. The "baseline 2-D" and "baseline 3-D" denote WERs obtained by the recognition without VQ-based fast emission computation approach using 2-D and 3-D streams respectively. It can be observed that WERs are close to baseline WER in case of emission approximation factor greater than 20%. For emission approximation factors less than 10% WER increase gets very high. In practice, emission approximation factor 10%...20% should be used: higher values lead to increase of computation costs and lower values lead to degradation of recognition performance.

Figure 6.1c demonstrates main characteristics of VQ based fast emission computation. The vertical axis denotes WER in percents, the horizontal axis shows computation time in seconds. The "baseline" point denotes the WER/recognition time ratio without streams and VQ approaches. The "baseline 2-D" and "baseline 3-D" points denote the ratios of recognition without VQ-based fast emission computation using 2-D and 3-D streams respectively.

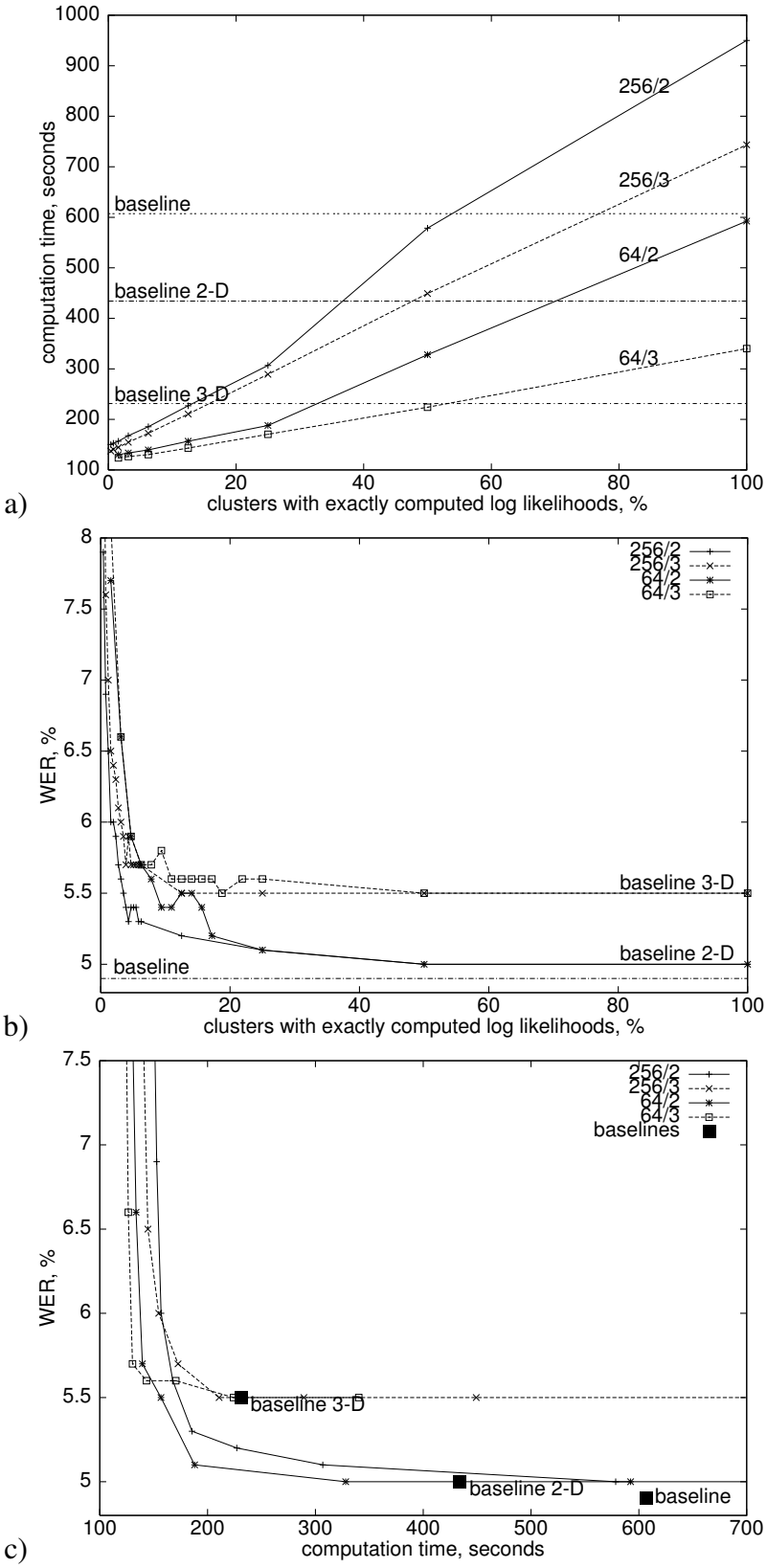

Figure 6.1: *Combination of clustering and streaming techniques for TRAIN_U HMM set: a) dependence of the execution time on the number of precisely computed clusters; b) dependence of WER on the number of precisely computed clusters; c) WER/execution time characteristics*

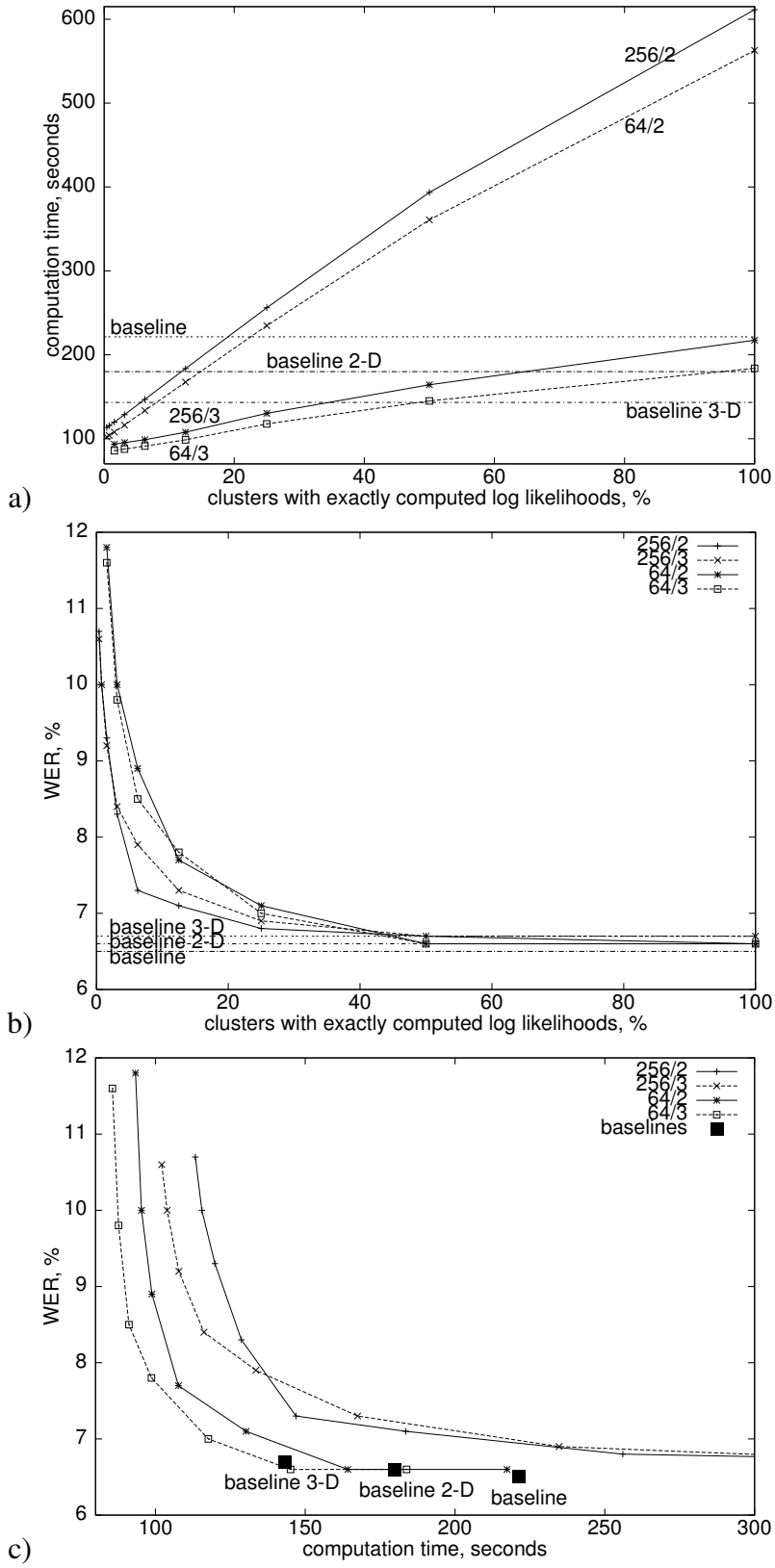

Figure 6.2: *Combination of clustering and streaming techniques for TRAIN_BA HMM set: a) dependence of the execution time on the number of precisely computed clusters; b) dependence of WER on the number of precisely computed clusters; c) WER/execution time characteristics*

In Figure 6.1c the curves “64/3” and “256/3” grow if the computation time decreases, the curves do not become less than “baseline 3-D” (3-D without VQ approach). This means that VQ approach being applied to 3-D stream SDCHMM cannot decrease the recognition time without degradation of WER. 2-D streams with VQ approaches (curves “64/2” and “256/2”) show better recognition performance than 3-D streams with VQ (curves “64/3” and “256/3”) and 3-D without VQ approach (“baseline 3-D”). 2-D streams with VQ approach leads to better recognition performance and recognition time, yet the 2-D approach leads to higher memory consumption than the 3-D approach.

Figure 6.2 show the results for SDII-mbl-apl task with context-dependent TRAIN_BA HMM set with 1200 Gaussians.

As it is shown in Figure 6.2a, the emission computation time baselines are small. For 2-D stream coding VQ-based fast emission computation (curves “256/2” and “64/2”) require more time than without VQ (“baseline 2-D”) if the emission approximation factor is higher than 15%. For 3-D stream coding VQ-based fast emission computation (curves “256/3” and “64/3”) require more time than without VQ (“baseline 3-D”) if the emission approximation factor is higher than 50%.

In Figure 6.2b the dependence of WER on emission approximation factor is shown. WER of VQ-based fast emission computation with 2-D and 3-D coding are higher than baselines if the emission approximation factor is less than 50%.

And finally, in Figure 6.2c the main characteristics of VQ-based fast emission computation is shown. All of the VQ approaches gives the same or even worse results as without VQ (baselines). In the case of 1200 Gaussians HMM the VQ-based fast emission computation is not efficient.

6.5 Conclusion

The experiments have shown that 2-D and 3-D stream coding are of high efficiency: they lead to the reduction of emission computation time up to 66% in comparison to the baseline system, which meets the objectives of the research (see Section 3.4).

3-D stream coding of HMMs with 1200 Gaussians is most appropriate for current applications in mobile telephones because this approach requires very low processing power and the memory consumption is about 10 kilobytes for 1200 Gaussians.

The reorganization of Gaussians (sorting) within one HMM state allows to decrease the computation time without any loss of recognition accuracy. This method is useful only for HMM sets that model each HMM state with several Gaussians. The approach does not lead to significant increase of computation time when HMM states are modeled by 1 or 2 Gaussians. The reordering is recommended for HMM recognizers as it does not degrade the recognition accuracy.

VQ-based emission computation is inefficient for HMMs with 1200 Gaussians: the reduction of computation costs leads to high WER increases. VQ-based fast emission computation for HMM is considered as a promising approach for larger HMM sets, for example with 4000, 6000, 8000 or 16 000 Gaussians. For larger HMM sets 2-D stream with VQ-based fast emission computation could have better WER/recognition time performance than HMM sets with 3-D stream (see Figure 6.1c). The VQ-based fast emission computation approach with streaming will have better performance with multilingual HMMs since they have more Gaussians than HMMs for only one language. The implementation of speech recognizer with HMM sets of 1200 Gaussians and 3-D streams coding is possible on contemporary mobile phones.

Most of experiments were performed using a workstation, these experiments allow to estimate the behavior of an embedded system. The workstation has high amount of working memory (1 GB) and cache (256 KB), thus the emission computation code and the HMM set may be completely cached. In such a way this system may be considered as an ideal case of an embedded device. The relative computational improvements on the ARM9TDMI microcontroller may be estimated from the relative results obtained on the workstation. In several years the performance of embedded devices will be similar to the performance of the used workstation that is why these experiments allow to forecast the performance of embedded devices in the nearest future.

Taking into account the architecture of a current mobile phone platform that contains one DSP with a fast access to only a comparatively small memory and a relatively slow microcontroller with a fast access to a bigger memory, it would make sense to share the decoding between these two processors according to their specifications. One solution would be to perform on DSP temporary distance precalculations where all possible distances between a feature vector and the codebook vectors are computed. For that DSP has access to the codebook of SDCHMM. DSP computes and saves in memory all possible stream log likelihoods for every feature vector stream components. These stream log likelihoods are then used by the microcontroller, which does not have to calculate stream log likelihoods anymore, but only needs to retrieve them from memory. Once the stream log likelihoods are not needed they are discarded from memory.

Chapter 7

Memory saving fast search algorithm

In the small vocabulary speech recognition the most memory consuming data block is occupied by acoustic models, the most computationally complex process is the emission computation. The large vocabulary speech recognition requires more memory and processing power for the search algorithm than in case of small vocabularies.

The baseline VSR recognizer was designed for small and medium vocabulary sizes. The goal of the research is to develop such search algorithm that is able to perform large vocabulary (20 000 words and more) isolated words search in real time on a baseline embedded system (100 MHz ARM processor). The large vocabulary isolated words recognition in VSR with vocabulary of 20 000 words and linear search requires 4.7 MB memory for search space and performs on the workstation 2.5 times slower than real time.

One of the approaches that reduces memory for vocabulary is a tree search. The tree search with its processing of equal word parts of different words only once per frame decreases also the required processing power for large vocabulary recognition tasks [Haeb-Umbach and Ney 1991; Ortmanns et al 1997b].

In this chapter the problems of the reduction of computation time and memory savings in the search algorithms for large vocabulary speech recognition are explored. First, the theoretical aspects of search are considered. Then a tree and a word stem based lexicon structures are described. Based on these structures the new modification of word stem based search is proposed.

The implementation of the word stem based tree search for large vocabulary speaker independent isolated word recognition for embedded systems is described in details. This fast search algorithm combines the effectiveness of the tree structure for large vocabularies and the fast Viterbi search within the regular structures of word stems. The algorithm is proved to be very fast for workstation and embedded platform realizations. In order to decrease the processing power, the word stem based tree search is combined with a frame dropping approach. The recognition speed was increased by a factor of 5 without frame dropping and by a factor of 10 with frame dropping in comparison to linear Viterbi search for iso-

lated word recognition task with a vocabulary of 20 000 words. Thus, the large vocabulary isolated word recognition becomes possible for embedded systems.

7.1 Frame dropping approach

A speech signal consists of speech parts and parts during which the speaker is silent. A method to further reduce the computational complexity of the search algorithm is to only send speech parts to the recognizer and drop the non-speech parts. The recognizer processes every 15 ms a frame of 32 ms of the speech signal. A voice activity detection (VAD) classifies each frame into speech or non-speech [Astrov and Andrassy 2003].

The VAD employed here consists of a multilayer perceptron neural network. The neural network has three layers: input, hidden and output layer. As input 12 cepstral coefficients plus one energy value are taken. The current frame as well as the three past frames and the three future frames are considered leading to 91 input values altogether. As output the network has one node which was trained to represent the non-speech probability of the current frame. A threshold of 0.5 was chosen for this output node to classify into speech and non-speech.

Furthermore a hang before of 2 frames and a hangover of 7 frames were applied. Like that seven frames of the signal are sent to the recognizer after the VAD detects the beginning of a non-speech section. The two frames before the VAD detect the beginning of a speech section are likewise sent to the recognizer. Thus a clipping of unvoiced speech parts at the beginning and end of an utterance should be avoided. This configuration leads to no degradation of the recognition performance in the experiments shown in this chapter.

7.2 Theoretical aspects of the tree search

In large vocabulary recognition task many words begin with the same initial phoneme sequences. Therefore the pronunciation lexicon is arranged as a tree [Ney et al 1998; Ney and Ortmanns 1999]. Each node of the tree stands for a phoneme such that a node sequence from the tree root to a tree leaf represents a word of the vocabulary. The leaves mark the end of a word, and some of them may be located in the tree interior, since some words form the beginning part of another word.

The general idea of a lexicon tree search is that the words starting with identical phonemes are processed together. A lexicon tree is shown in Figure 7.1. Three phonemes of words “einem” and “einer” are identical (“ai”, “n”, “e”), these phonemes build the subset of words “Ei”, “ein” and “eine”. Such processing of phonemes which happens only once for several words reduces number of computations. For identical phoneme groups the acoustical scores are computed only once per search iteration. In such a way the search space for German

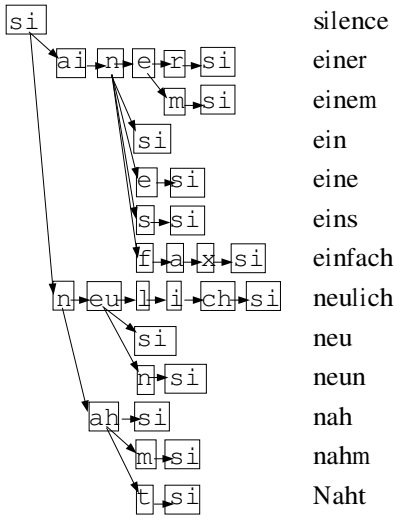

Figure 7.1: *Structure of a phoneme-based lexicon tree*

language with lexicon of 12 000 words can be reduces by 2.6 times [Haeb-Umbach and Ney 1991; Ortmanns et al 1997b]. During the search, the first several states of each word are active most of the time. Thus, using the tree structure allows to reduce the number of computations by 5-6 times [Haeb-Umbach and Ney 1991; Ney et al 1992] for a large vocabulary (12 000 words).

7.3 Modified word stem based tree search

The software implementation of the tree search which is described in Section 7.2 could be realized as a graph that branched after each phoneme with the following alternatives [Hauenstein 1993a,b]:

- is the end of a word reached?
- is there several alternatives for branching?
- is the the end of a short word reached in the branching position?

For the commonly used tree search the tree is built up by a structure of linked lists. When the branching point is reached, several memory access operations and address computations are required. In case of large vocabularies these operations lead to “wait states” when the data is not cached. In embedded systems the cache memory is several tens of kilobytes, the cache size is not sufficient to accelerate the memory access. In [Deligne et al 2001] the minimized graph search only leads to a modest increase in decoding speed, as the RISC

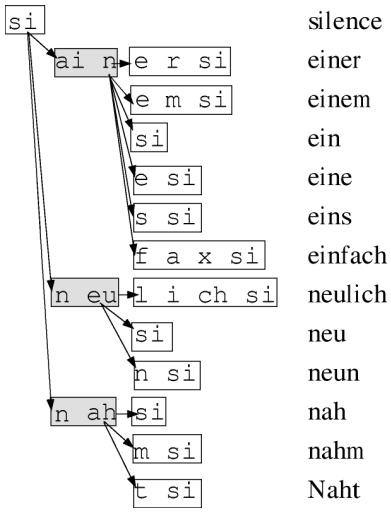

Figure 7.2: *Structure of a word stem based lexicon tree (stem length is limited to 2 phonemes)*

hardware which uses a deeply pipelined ALU cannot function at high efficiency because of the irregularities in minimized graph structure.

A solution for this problem is a word stem based lexicon structure which was proposed in [Hauenstein 1993a,b]. The structure of word stem based tree is shown in Figure 7.2. Instead of using a lexicon, that is fully branched after each phoneme (tree-based lexicon), an approach of fixed length word stems is chosen. Each word consists of a fixed length word stem and a word-ending. In Figure 7.2 a word stem size is set to 2 phonemes. All words with shared word stems are treated in common as long as the active states are within the word stem. After reaching the end of the stem the search algorithm branches into all word endings regardless whether they have more phonemes in common or not. Despite of having a third phoneme in common the words “einer” and “einem” are treated as different word-endings. This leads to a very simple and regular algorithm.

In the following the new organization of a lexicon is proposed. The structure uses the combination of the tree approach with the idea of the word stems. Such lexicon representations has advantages of the tree (higher savings of memory and computational costs) and advantages of the word stems (compact placement of states within linear blocks in memory, fast processing of regular structures).

In the new structure word-endings and word stems are considered as linear units of various length and treated by one algorithm. A word stem now denotes a set of sequentially placed states, the processing of such regular linear structure is fast. The data within a word stem is placed compactly in the memory, this leads to a better cache use.

The new fast tree search algorithm codes the tree with word stems as presented in Figure 7.3. The word stems (shown as rectangles) are built on a state level, in Figure 7.3 they are

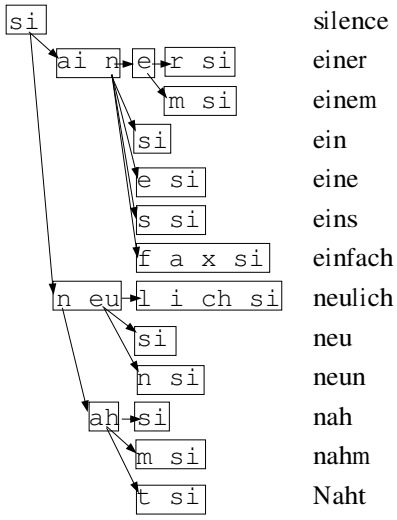

Figure 7.3: Modified word stem based tree structure

shown on a phoneme level for a simple representation and understanding. In this structure the branching is only possible at the end of a word stem. The end of a word stem is not always the end of a word.

The search starts at the tree top of the structure, the initial probability of the first state in the top stem is set to 1. The Viterbi search iteration is performed for each feature vector. Within the word stem the search is executed similar to the linear search algorithm. When the state probability is less than the pruning threshold, this state is pruned. When the state probability becomes greater or equal to pruning threshold it is processed in the following Viterbi iterations (the state is activated). During the search the iterations are performed only for active word stems. The word stem is pruned when it has no active states. In the following the pruning threshold defines the maximum difference between current path log likelihood and the best path log likelihood. The pruning thresholds 2000 and 800 are considered: the value 2000 is used to get very accurate results and the value 800 is used for faster computations with low degradation of WER.

After reaching the end of the stem the search branches into the successor word stems, they are activated. The search path reaches the word end when the word stem has no successors (it is the last stem of the word) and its last state is active. The word hypothesis is chosen as the recognition result when it has highest path probability during some predefined minimum stable time.

7.4 Experiments

In this section the recognition accuracy and computation performance of the search algorithms are shown. The memory requirements and processing power of the word stem based

search algorithms and the baseline linear search algorithm are compared. The algorithms were tested on Cities and CarKit isolated word recognition tasks. The recognition experiments with TRAIN_BA HMMs were performed for different vocabulary sizes: 495, 1500, 20 102 (20k) and 76 784 (77k) words. The results of the experiments were obtained on a Pentium III 850 MHz workstation running Linux in order to explore the recognition accuracy and to compare the performance with the baseline recognizer. The additional tests on an ARM platform were made in order to estimate the required system resources for embedded systems.

Estimation of recognition accuracy

In the following the word error rate (WER) using the n-best search is investigated. The search algorithm proposes up to n best hypotheses. This type of the recognition could be used in SMS dictation in mobile phones or in speech controlled navigation systems, e. g. where user after the speech input may precisely select the result from the proposed list via touch-screen or keyboard.

The recognition accuracy is the same for the baseline recognizer with linear lexicon and with new word stem based tree structure. The usage of frame dropping does not reduce the recognition accuracy.

N-best	WER [%]			
	495	1500	20k	77k
Cities, pruning=800				
n=1	13.3	18.6	42.3	59.2
n=5	4.5	6.8	19.5	32.0
n=20	3.5	4.5	9.8	18.2
CarKit, pruning=2000				
n=1	23.9	32.0	55.6	70.2
n=5	7.3	10.9	33.0	45.5
n=20	4.0	7.1	18.5	28.9
CarKit, pruning=800				
n=1	25.2	34.0	58.0	70.2
n=5	14.3	17.2	38.9	50.8
n=20	12.0	15.9	27.8	38.8

Table 7.1: *WER for different vocabulary sizes*

In Table 7.1 the recognition results are the same for the baseline and modified word stem based tree search algorithms. WER is shown for different vocabulary sizes (495, 1500, 20k and 77k). WER increases with increasing vocabulary sizes. Cities task was tested with pruning threshold 800, in the case of CarKit test set two pruning thresholds (2000 and 800) were tested. The threshold 2000 leads to better recognition results but requires more processing power.

The n-best recognition rate is shown for $n = 1$, $n = 5$ and $n = 20$ hypotheses. The n-best search with $n = 20$ hypotheses is used to estimate highest possible recognition accuracy. Selecting $n = 5$ will give the results close to the real applications, $n = 1$ provides the results when no correction or selection from n-best list is possible.

In the experiments TRAIN_BA HMMs set for embedded systems with only 1200 Gaussians is used. With TRAIN_BA HMM set the recognition accuracy without n-best lists ($n = 1$) is unacceptable for real applications. With $n = 5$ the recognition with 495 and 1500 words vocabularies may be applied in practice in low noise conditions (task Cities). Large vocabulary recognition tasks (20k and 77k) would require more precise modeling with higher number of Gaussians.

Reduction of memory requirements

The lexicon structure and the search space use compact coding of parameters in such a way that one 32-bit word contains two or three variables in order to reduce the memory usage. Such placement is advantageous for embedded devices as less memory is required. The arrays are placed compactly and the cache memory is used more efficiently.

In Table 7.2 the memory requirements are shown for different vocabulary sizes: 495, 1500, 20k and 77k words. The shown amount of memory includes memory for lexicon structure and the full search space for isolated word recognition (without pruning). The word stem based tree search algorithm requires 1.8-2.4 times less memory than the baseline search with a linear vocabulary structure.

As can be observed, a medium vocabulary (495 and 1500 words) isolated words recognizer require less than 160 KB of memory for the search and could thus be implemented in most contemporary mobile phones. A large vocabulary isolated word recognition require less than 7 MB of memory for the search and could be realized in embedded devices (for example, PDAs or car navigation systems).

algorithm	memory requirements			
	495	1500	20k	77k
linear search (baseline)	94 KB	301 KB	4.7 MB	16.2 MB
word stem based tree search	53 KB	157 KB	2.1 MB	6.8 MB

Table 7.2: *Memory requirements for the search with different vocabulary sizes*

Reduction of computation costs on workstation

In this set of experiments the performance of the search algorithms on a workstation (Pentium III, 850 MHz) is estimated for Cities task with pruning threshold 800. First, the lexicon structure (linear or word stem based tree) and feature vectors are read into the memory.

algorithm	recognition time (RTF)			
	495	1500	20k	77k
baseline	0.12	0.26	2.50	-
baseline-FD	0.15	0.20	1.02	-
word stem based tree search	0.07	0.09	0.47	1.04
word stem based tree search-FD	0.03	0.05	0.21	0.46

Table 7.3: *Recognition time per utterance (real time factor) for different vocabulary sizes*

Then the algorithm computes log likelihoods and performs the search for each utterance. This test was made with and without frame dropping. The execution time was measured for the whole recognition process (feature extraction, emission computation and search) using Unix ”time” shell command (user CPU time). These measurements were repeated 5 times and the mean values were obtained, the results are represented as real time factors and shown in Table 7.3. The baseline linear search algorithm uses signed 16-bit integer indexes and able to process vocabularies with 32 768 words or less, that is why the recognition time for the baseline algorithm could not be measured for 77k vocabulary.

The developed fast search algorithms are faster than the baseline recognition algorithm in all tests. The frame dropping increases the recognition speed (see lines “baseline-FD” and “word stem based tree search-FD”). For the baseline algorithm with a 495-word vocabulary the frame dropping increases the recognition time because of the yet unoptimized frame dropping algorithm realization. As shown in Table 7.3 the frame dropping in general accelerates the search procedure by 2-2.5 times.

Estimation of computation costs on embedded devices

The performance of the fast search algorithm is explored on three ARM RISC processor cores: ARM9TDMI, ARM920T and ARM940T (see Section 3.3). The first experiment is performed with TRAIN_BA HMM set, the pruning threshold is set to 800. The feature extraction and emission computation times are not considered: the program reads from files the vocabulary structure and the precomputed log likelihoods (emission scores) for one utterance from Cities task with 255 frames (3.825 s including pauses before and after the speech part). Then the number of core clocks only for the tree search procedure is counted. From the number of core clocks the real time factor is computed, see Table 7.4.

Without frame dropping only recognition with 495 and 1500 word vocabularies is possible in real time on the reference embedded system. With vocabulary of 20k words the real time recognition becomes possible only with frame dropping. For the real time recognition with 77k words vocabulary the system running 2 times faster than the reference platform is required (both processor clock frequency and memory access rates have to be accelerated).

The search on ARM920T processor runs faster than on the ARM940T because of the higher amount of cache memory, but both processors require more time than ARM9TDMI core

vocabulary size	RTF		
	ARM9TDMI	ARM920T	ARM940T
pruning=800, without frame dropping			
495	0.168	0.245	0.308
1500	0.399	0.718	0.796
20k	2.734	6.003	5.825
77k	5.867	15.539	15.486
pruning=800, with frame dropping			
495	0.03.	0.054	0.070
1500	0.063	0.149	0.166
20k	0.448	0.981	0.955
77k	0.822	2.169	2.173

Table 7.4: *Recognition speed measured in real time factor for one utterance from Cities task*

with an ideal zero wait-states memory model. ARM9TDMI may perform search with frame dropping and 77k vocabulary in real time, this shows that the memory access time and cache memory size are the most critical parameters for the large vocabulary isolated word recognition on embedded devices.

The dynamically consumed processing power for ARM9TDMI core is shown in the following experiment. For each frame of the utterance the required amount of core clocks were computed for isolated word recognition with fast search algorithm. The results are shown in Figure 7.4 for the vocabulary size of 20k words. The dynamically consumed processing

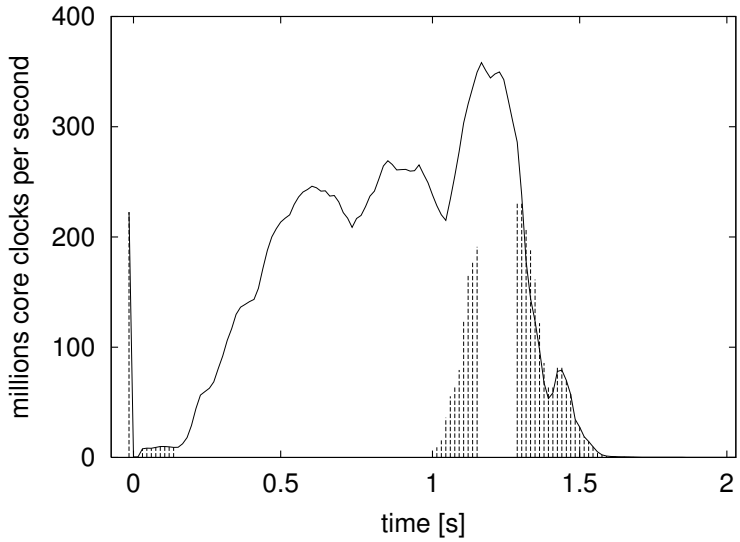

Figure 7.4: *Dynamically consumed processing power for one utterance in case of 20k vocabulary size. The firm line shows the required processing power for the search without frame dropping. The impulses shows the required processing power for the search with frame dropping*

power without frame dropping is drawn with the line, the results with frame dropping are drawn with impulses.

The processing of the first frame includes the initialization of the search space, that is why the impulses in the beginning of the graphs are observed. In the test utterance that shown in Figure 7.4, the frame dropping algorithm decided, that the utterance had no speech signal in a short interval between 1 s and 1.5 s. During this period nothing was sent to the search algorithm, and search iterations were not performed. This explains the gap in the graph between 1 s and 1.5 s.

Since the recognizer has a frame buffer, each frame need not be processed in real time. The search in recognition of this utterance without frame dropping is performed with 0.715 real time factor, with frame dropping the processing speed is accelerated to 0.117 real time factor. As observed, the frame dropping reduces the required processing power.

The next experiment shows the needed processing power for search in embedded systems using ten utterances from CarKit test set, TRAIN_BA HMM set, pruning thresholds 800 and 2000, no frame dropping. In this test the search time was measured (feature extraction and emission computation were not considered). In Table 7.5 the required processing power in real time factor is shown.

The required processing power depends on the utterance: if the utterance is noisy and could not be well recognized or has many hypotheses, then the search path is wide and the processor load is high. In Table 7.5 the processing power for misrecognized utterances is shown in **bold** font style. As observed, the misrecognized utterances require more processing power than utterances recognized correctly.

As can be seen from the table, the recognition with 20k word vocabulary may be performed in real time, some delay in the obtaining of the recognition results may be observed (for real time factors greater than 1). The more accurate recognition with the pruning score 2000 cannot be executed with acceptable delays with vocabularies greater than 10k words.

vocab-ulary / pruning	required processing power for utterance real time factor										
	1	2	3	4	5	6	7	8	9	10	avg.
495/800	0.03	0.03	0.02	0.03	**0.07**	0.05	**0.05**	0.06	0.05	**0.07**	0.05
495/2000	0.09	0.10	0.06	0.09	**0.26**	0.17	0.16	0.13	0.16	**0.18**	0.15
1500/800	0.07	0.08	0.03	0.07	**0.19**	0.13	**0.13**	0.17	**0.13**	**0.18**	0.12
1500/2000	0.28	0.29	0.19	0.27	**0.84**	0.47	0.52	0.42	**0.50**	**0.57**	0.45
20k/800	0.50	0.50	0.28	0.53	**1.85**	0.98	**1.23**	1.52	**1.02**	**1.48**	0.99
20k/2000	3.60	3.29	2.08	3.40	**13.27**	6.26	8.09	6.49	**7.20**	**7.66**	6.13
77k/800	0.96	**0.96**	0.68	1.05	**4.11**	**2.01**	**2.66**	3.28	**2.02**	**3.07**	2.08
77k/2000	9.16	**7.47**	5.20	8.64	**39.41**	**16.53**	24.07	19.17	**20.72**	**22.27**	17.26

Table 7.5: *Search performance for embedded system with ARM920T core and CarKit test set*

7.5 Conclusion

The new modification of the word stem based search for large vocabulary isolated word recognition is developed. The considered algorithm uses less memory and processes faster than the baseline search algorithm with linear lexicon structures. In combination with frame dropping the new algorithm allows the realization of medium and large vocabulary isolated word recognition in embedded systems.

The medium vocabulary speech recognition is possible on most of modern embedded systems. The isolated word recognition with vocabulary of 1500 words requires 160 KB of memory for the search. The recognition may be processed with 0.17 real time factor for the search with frame dropping and with 0.13 real time factor for emission computation on an ARM processor. The n-best recognition of city names with $n = 5$ has 6.8% word error rate.

The real time isolated speech recognition may require the whole processing power of embedded system. The large vocabulary isolated word recognition (20k words) requires 2.1 MB of memory to store the lexicon structure and the search space. The large vocabulary recognition task may be processed in real time with frame dropping on the reference ARM embedded device. The n-best recognition of city names with $N = 5$ has 19.5% word error rate. The recognition accuracy have to be improved by employing more detailed HMM sets with higher number of Gaussians, e. g. 7500.

The increase of pruning thresholds will reduce error rates, but it will increase the required computation costs. New microcontrollers with higher computation power will be able to perform large vocabulary isolated speech recognition in real time more accurate. The most critical parameters of the embedded system are the memory access time and the cache size.

Chapter 8

Discussion and future work

8.1 Main achievements

Currently small and medium vocabulary isolated word recognition with very low consumption of system resources and very large vocabulary isolated word recognition (more than 50 000 words) are the main challenges in speech recognition for embedded devices. Voice control of navigation systems in vehicles, name dialing and isolated dictation of short messages in embedded devices becomes very attractive. A requirement of a high amount of system resources in embedded systems is a serious problem of speech recognizers. In this thesis several approaches were investigated in order to find a trade-off between the accuracy of the speech recognition in noisy environment, memory and processing power consumption.

The baseline speech recognizer (see Section 3.3) considered as the reference recognizer in this work has the following features. It uses continuous density HMMs with 1200 or 4000 Gaussians. Each entry in the lexicon is represented as a linear sequence of HMM states.

The baseline embedded system has the ARM920T microcontroller with 100 MHz core clock rate and 50 MHz bus rate, the sequential memory access time is 20 ns and non-sequential access time is 100 ns. The system has 16 KB instructions cache and 16 KB data cache. An HMM set with 4000 Gaussians requires 104 KB of memory, an HMM set with 1200 Gaussians requires about 32 KB. The 30-word recognition task with German TRAIN_BA HMM set (1200 Gaussians) can run on the baseline system in real time, but it consumes about half of the processor resources: the feature extraction runs with 0.08-0.12 real time factor, the emission computation and Viterbi search run with 0.289 and 0.091 real time factors.

The large vocabulary isolated word recognition task with 20 000 words lexicon cannot be executed in real time on such embedded system. The recognition runs 2.5 times slower than real time even on the workstation with a Pentium-III 850 MHz processor. The lexicon

structure and the search space are the main memory consumers in the large vocabulary isolated word recognition system, they require together 4.7 MB.

The main objective of the research was the development of the algorithms that allow speech recognition to be executed in real time on the reference embedded system. The goals are listed below, see also Section 3.4 for details. The memory consumption by HMM parameters has to be reduced by 50%, the emission computation has to be accelerated by the factor of 2, the relative increase of WER must be less than 10%. For the large vocabulary isolated words recognition task with 20 000 words lexicon the search has to be accelerated, the recognition has to be performed in real time, the memory consumption by the lexicon structure has to be reduced by 50%. The achievements are listed below.

For a typical HMM set of 1200 Gaussians the 3-D streams approach accelerates the emission computation 3.5 times. On the reference embedded device the emission computation is performed with real time factor 0.082 instead of 0.289 for the baseline recognizer. The memory requirements for acoustic models is reduced by the factor of 3, an HMM set with 4000 Gaussians requires now only 37 KB of memory instead of 104 KB, an HMM set with 1200 Gaussians requires about 10 KB instead of 29 KB. The distortions introduced by coding algorithms are very low and the relative increase of WER is less than 9.2%.

The reduction of the memory consumption for HMM parameters by 66% have been achieved by coding of Gaussian mean vectors and weights. Memory demands for Gaussian mean vectors were reduced by applying the SDCHMM approach. For further improvements instead of several independent codebooks only one shared codebook was used. The recognition rates of SDCHMMs with independent and shared codebooks are similar.

- 2-D stream coding with the shared codebook has 50% compression rate with only 2.9% relative decrease of the recognition accuracy.
- 3-D stream coding with the shared codebook has 66% compression rate with less than 9.2% relative decrease of the recognition accuracy.

Thus, 2-D and 3-D streams have a good trade-off between the recognition accuracy and the memory consumption, these schemes are most appropriate for the coding of HMM parameters in speech recognition systems in embedded devices.

The streams approach accelerates the computation of emission probabilities. With the 2-D and 3-D streams approaches the computation is performed 2 and 3 times faster, respectively. The combination of streams approach with VQ-based computation of emission probabilities may increase the computation speed further. This approach is advantageous for SDCHMMs with 2-D streams, in this case they outperform SDCHMMs with 3-D streams without VQ in emission computation speed and acoustic modeling accuracy.

Weights of Gaussian pdfs were coded using new coding approaches:

- The standard scalar quantization technique was investigated. The reached compression rate is slightly more than 50% for any size of HMM sets: coded weights occupy only 50% of memory in comparison to the weights from the baseline HMM set, extra 512 bytes are necessary to store a codebook. This coding has the highest distortion in all considered coding schemes.

- The recursive weight difference coding technique encodes Gaussian weights very precisely, the coding distortion is very low, in many cases it is possible to encode weights without any loss of information. This coding scheme may achieve a 50% compression rate in case of large context-independent HMM sets (more than 4000 Gaussians) where each HMM state is modeled by at least 10 Gaussians. The compression rate decreases when HMM states are modeled by one or two Gaussians, in this case the coding approach is inefficient.

- The square root coding stores integer parts of the square roots of the Gaussian weights. This scheme is most advantageous for speech recognition in embedded devices. The algorithm is very simple and fast, the decoding of each Gaussian weight requires only one integer multiplication. The algorithm has the lowest compression rate, the memory consumption of Gaussian weights is reduced by exactly 50% for any size of HMM sets.

In experiments the change of WER is minimal for all coding schemes, the recognition results are similar for all weight coding schemes. Thus, it is advantageous to encode Gaussian weights by the square root coding technique because it has lower computational cost and better compression rate than other coding schemes.

The search procedure for large vocabulary isolated word recognition (20 102 words) was accelerated, it is executed now with $0,981$ real time factor. The acceleration was achieved by using a new modified word stem based tree structure and a frame dropping algorithm. Lexicon structure and search space occupy now 2.1 MB instead of 4.7 MB for the baseline system. Thus, large vocabulary isolated word recognition becomes possible in embedded devices.

The isolated word recognition task with 76 784 words vocabulary is able to run in real time on systems with at least 2 times higher performance than the baseline embedded system. Current embedded devices with more powerful processors, faster memory access and higher cache size may perform such recognition in real time.

8.2 Future work

The prediction known as Moore's Law [Moore 1965] states that transistor density on integrated circuits doubles about every two years. Thus, in 5 years embedded devices will have approximately 10 times more resources than they have now (50-500 MHz processors and 4-256 MB of working memory).

With an increasing amount of system resources other, more complicated speech recognition tasks may be realized. The large vocabulary continuous speech recognition in embedded devices will set new goals. The continuous search requires at least 10 times more memory and computational power because of processing several paths in parallel. New algorithms have to be developed to realize large vocabulary continuous speech recognition in real time.

The continuous speech recognition will require implementation of language models, bigrams or trigrams. For large vocabularies the language models will require several megabytes of memory in order to store their parameters, efficient memory compression algorithms for language models have to be developed. More accurate tri-grams will complicate the search procedure, new approaches for implementation of language models in the search will be needed.

Current HMM sets designed for recognizers on embedded devices have poor recognition accuracy for large vocabulary recognition tasks. The improvement may be achieved by increasing the number of Gaussians in the HMM set, but this will increase memory consumption by HMM parameters and slow down the computation of emission probabilities, thus, new solutions will be required.

Other input channels available in embedded devices may be employed in parallel to spoken speech in order to improve recognition accuracy. The lip-reading algorithm will require additional features extracted from the sequence of images obtained by a digital camera in a modern mobile phone.

The speech recognition in embedded devices became possible not long ago, it has a high potential of further improvements, implementation fields and research activities. The development is driven by the increasing power of computers and embedded devices and also telephone and Internet applications using voice services. In the future a man-machine voice communication will become customary in everyday situations and will simplify human life.

Appendix A

Nomenclature

$\mathbf{A}$	LDA matrix
$A_{s,s'}$	transition probability from state s to state s'
$a_{s,s'}$	transition penalty from state s to state s'
B_s	emission probability of state s
b_s	emission penalty of state s
C_i	word class in class-based language model
$C_{s,m}$	weight of emission probability of state s and pdf m
$c_{s,m}$	weight penalty of state s and Gaussian m
D	dimensionality of feature vector; dimensionality of data vector
D_k	dimensionality of stream k
d	dimension index
$d(x, \mu)$	distance between feature vector x and Gaussian mean vector μ
G	quantizer distortion
$G_{s,m,k}$	atom, distance between feature stream vector and Gaussian stream vector
$g(x, \hat{x})$	distance measure between value x and reproduction value $\hat{x}$
$H_r(S)$	entropy
H	meta-atom
I_n	quantization subinterval
i	index variable
j	index variable
K	number of streams
k	stream index
L	average length of a code
l_i	length of a codeword X_i
m	mixture index
N	number of recognized words; number of entries in a codebook

$N(x, \mu, \Sigma)$	Gaussian pdf
n	frame number; number of vectors in data set; number of Gaussians in HMM set
q	base of source alphabet
r	base of code alphabet
S	number of states; source alphabet
s	state index
T	number of frames
t	time; frame number
V	codebook; quantizer codebook
v	vector; codebook vector (codeword); quantization level
W	sequence of recognized words
w	recognized word
$w(n)$	Hamming window
X	sequence of feature vectors or data; code word; data set
x	data value; data vector; feature vector
$[x_{\min}, x_{\max}]$	quantization interval
$\hat{X}$	reproduction data set
$\hat{x}$	reproduction value; reproduction vector
y	interval border in scalar quantization (threshold)
α	displacement multiplication coefficient
$\alpha_t(i)$	best path score at time t that ends in the state i
$\lambda(\pi, a, b)$	hidden Markov model
μ	Gaussian mean vector
η	average error
π	initial state probability vector
Σ	covariance matrix
σ	variance
ψ	back tracking array for Viterbi search

Appendix B

List of abbreviations

ALU	arithmetic logical unit
ASR	automatic speech recognition
BPE	byte pair encoding
CPU	central processing unit
DTW	dynamic time warping
DSP	digital signal processor
FFT	fast Fourier transform
GSM	Global System for Mobile Communications
HMM	hidden Markov model
CDHMM	continuous density HMM
FPTHMM	feature parameter tying HMM
SDCHMM	stream (subspace) distribution clustering HMM
TMHMM	tied mixtures HMM
Hz	hertz
kHz	kilohertz
MHz	megahertz
LDA	linear discriminative analysis
LVCSR	large vocabulary continuous speech recognition
LVISR	large vocabulary isolated speech recognition
OOV	out-of-vocabulary
PDA	personal digital assistant
pdf	probability density function
RISC	reduced instruction set computer
RLE	run-length encoding
RTF	real time factor
SGTS	stream (subspace) Gaussian tying structure
SIMD	single instruction, multiple data

SMS	short message service
SQ	scalar quantization
SST	sub-state tying
VAD	voice activity detection
VQ	vector quantization
WER	word error rate
μC	microcontroller

Bibliography

[Ahmed and Holmes 2004] Ahmed, Beena ; Holmes, W. H.: A Voice Activity Detector Using the Chi-Square Test. In: *Proc. IEEE Int. Conf. on Acoustics, Speech, and Signal Processing (ICASSP)* vol. I, 2004, pp. 625–628

[Aiyer et al 2000] Aiyer, A. ; Gales, M.J.F. ; Picheny, M.A.: Rapid likelihood calculation of subspace clustered Gaussian components. In: *Proc. IEEE Int. Conf. on Acoustics, Speech, and Signal Processing (ICASSP)* vol. III, 2000, pp. 1519–1523

[ARM 2002] ARM: *Benchmarking with ARMulator, Application Note 93*. March 2002. – ARM DAI 0093A, ©ARM Ltd.

[Astrov 2002] Astrov, Sergey: Memory space reduction for Hidden Markov Models in low-resource speech recognition systems. In: *Proc. Int. Conf. on Spoken Language Processing (ICSLP)*, 2002, pp. 1585–1588

[Astrov and Andrassy 2003] Astrov, Sergey ; Andrassy, Bernt: Large Vocabulary Speaker Independent Isolated Word Recognition for Embedded Systems. In: *Proc. European Conference on Speech Communication and Technology (Eurospeech)*, 2003

[Astrov et al 2003] Astrov, Sergey ; Bauer, Josef G. ; Stan, Sorel: High Performance Speaker and Vocabulary Independent ASR Technology for Mobile Phones. In: *Proc. IEEE Int. Conf. on Acoustics, Speech, and Signal Processing (ICASSP)* vol. II, 2003, pp. 281–284

[Bauer 1997] Bauer, Josef G.: Enhanced Control and Estimation of Parameters for a Telephone Based Isolated Digit Recognizer. In: *Proc. IEEE Int. Conf. on Acoustics, Speech, and Signal Processing (ICASSP)*, 1997, pp. 1531–1534

[Bauer 2001] Bauer, Josef G.: *Diskriminative Methoden zur automatischen Spracherkennung für Telefon–Anwendungen*, Lehrstuhl für Mensch–Maschine–Kommunikation der Technischen Universität München, Dissertation, 2001

[Bauer 2004] Bauer, Josef G.: *Personal Communication*. 2004

[Beaugeant and Scalart 2001] Beaugeant, Christophe ; Scalart, Pascal: Speech Enhancement Using a Minimum Least Square Amplitude Estimator. In: *7th Int. Workshop on Acoustic Echo and Noise Control*, 2001

[Bocchieri 1993] Bocchieri, Enrico: Vector Quantization for the Efficient Computation of Continuous Density Likelihoods. In: *Proc. IEEE Int. Conf. on Acoustics, Speech, and Signal Processing (ICASSP)* vol. II, 1993, pp. 692–695

[Bocchieri and Mak 2001] Bocchieri, Enrico ; Mak, Brian Kan-Wing: Subspace Distribution Clustering Hidden Markov Model. In: *IEEE Transactions on Speech and Audio Processing* 9 (2001), Nr. 3, pp. 264–275

[Bronstein and Semendjajew 1989] Bronstein, I.N. ; Semendjajew, K.A.: *Taschenbuch der Mathematik*. Thun und Frankfurt/Main : Verlag Harri Deutsch, 1989

[Data Compression – Wikipedia 2006] *Data Compression, Wikipedia*. 2006. – http://en.wikipedia.org/wiki/Data_compression

[DataCompression-Web Site 2002] Nelson, Mark: *DataCompression.info*. 2002. – http://datacompression.info

[Deligne et al 2001] Deligne, Sabine ; Eide, Ellen ; Gopinath, Ramesh ; Kanevsky, Dimitri ; Maison, Benoit ; Olsen, Peder ; Printz, Harry ; Sedivy, Jan: Low-Resource Speech Recognition of 500-Word Vocabularies. In: *Proc. European Conference on Speech Communication and Technology (Eurospeech)* vol. III, 2001, pp. 1833–1836

[Dolfing 2002] Dolfing, Hans J.: A Comparison of Prefix Tree and Finite-State Transducer Search Space Modelings for Large-Vocabulary Speech Recognition. In: *Proc. Int. Conf. on Spoken Language Processing (ICSLP)*, 2002, pp. 1305–1308

[Draxler et al 1999] Draxler, Christoph ; Grudszus, Robert ; Euler, Stephan ; Bengler, Klaus: First Experiences of the German SpeechDat-Car Database Collection in Mobile Environments. In: *Proc. European Conference on Speech Communication and Technology (Eurospeech)* vol. 2, 1999, pp. 919–922

[Duda and Hart 1973] Duda, Richard O. ; Hart, Peter E.: *Pattern Classification and Scene Analysis*. New York, Chichester, Brisbane, Toronto, Singapore : John Wiley & Sons, 1973

[ELDA-Web Site 2005] *ELDA Web Site*. 2005. – http://www.elda.org

[ELRA-Web Site 2005] *ELRA Web Site*. 2005. – http://www.elra.info

[FSM–Internet site 2003] *General-Purpose Finite-State Machine Software Tools*. 2003. – http://www.research.att.com/sw/tools/fsm

[Furui 1986] Furui, Sadaoki: Speaker-Independent Isolated Word Recognition Using Dynamic Features of Speech Spectrum. In: *IEEE Transactions on Acoustics, Speech and Signal Processing* 34 (1986), Nr. 1, pp. 52–59

[Gu and Rose 2000] Gu, Liang ; Rose, Kenneth: Sub-State Tying in Tied Mixture Hidden Markov Models. In: *Proc. IEEE Int. Conf. on Acoustics, Speech, and Signal Processing (ICASSP)* vol. II, 2000, pp. 1013–1016

[Gu and Rose 2002] Gu, Liang ; Rose, Kenneth: Substate Tying With Combined Parameter Training and Reduction in Tied-Mixture HMM Design. In: *IEEE-T-SAA* 10 (2002), Nr. 3, pp. 137–145

[Haeb-Umbach et al 1993] Haeb-Umbach, R. ; Geller, D. ; Ney, H.: Improvements in Connected Digit Recognition using Linear Discriminant Analysis and Mixture Densities. In: *Proc. IEEE Int. Conf. on Acoustics, Speech, and Signal Processing (ICASSP)* vol. II, 1993, pp. 239–242

[Haeb-Umbach and Ney 1991] Haeb-Umbach, Reinhold ; Ney, Hermann: A Look-Ahead Search Technique for Large Vocabulary Continuous Speech Recognition. In: *Proc. European Conference on Speech Communication and Technology (Eurospeech)* vol. II, 1991, pp. 495–498

[Hauenstein 1993a] Hauenstein, Alfred: Architecture of a 10000 Word Real Time Speech Recognizer. In: *Proc. European Conference on Speech Communication and Technology (Eurospeech)* vol. 3, 1993, pp. 1829–1832

[Hauenstein 1993b] Hauenstein, Alfred: *Optimierung von Algorithmen und Entwurf eines Prozessors für die automatische Spracherkennung*, Lehrstuhl für Integrierte Schaltungen der Technischen Universität München, Dissertation, 1993

[Hauenstein and Marschall 1995] Hauenstein, Alfred ; Marschall, Erwin: Methods for Improved Speech Recognition over Telephone Lines. In: *Proc. IEEE Int. Conf. on Acoustics, Speech, and Signal Processing (ICASSP)* vol. 1, 1995, pp. 425–428

[Huffman 1952] Huffman, D. A.: A method for the construction of maximum redundancy codes. In: *Proc. IRE* vol. 40, 1952, pp. 1098–1101

[IEEE 1985] : *IEEE Standard 754-1985. IEEE Standard for Binary Floating-Point Arithmetic*. 1985

[Ircing and Psutka 2003] Ircing, Pavel ; Psutka, Josef: Fitting Class-Based Language Models into Weighted Finite-State Transducer Framework. In: *Proc. European Conference on Speech Communication and Technology (Eurospeech)*, 2003, pp. 1873–1876

[Kanthak et al 2000] Kanthak, S. ; Schütz, K. ; Ney, Hermann: Using SIMD Instructions for Fast Likelihood Calculation in LVCSR. In: *Proc. IEEE Int. Conf. on Acoustics, Speech, and Signal Processing (ICASSP)*, 2000, pp. 1531–1534

[Kanthak et al 2002] Kanthak, Stephan ; Ney, Hermann ; Riley, Mihael ; Mohri, Mehryar: A Comparison of Two LVCSR Search Optimization Techniques. In: *Proc. Int. Conf. on Spoken Language Processing (ICSLP)*, 2002, pp. 1309–1312

[Kneser and Ney 1991] Kneser, Reinhard ; Ney, Hermann: Forming Word Classes by Statistical Clustering for Statistical Language Modelling. In: *First Quantitative Linguistics Conference, QUALICO*, 1991, pp. 221–226

[Köhler 2000] Köhler, Joachim: *Erstellung einer statistisch modellierten multilingualen Lautbibliothek für die Spracherkennung*. Aachen : Shaker Verlag, 2000

[Komori et al 1995] Komori, Yasuhiro ; Yamada, Masayuki ; Tamamoto, Hiroki ; Ohora, Yasunori: An Efficient Output Probability Computation for Continuous HMM Using Rough and Detail Models. In: *Proc. European Conference on Speech Communication and Technology (Eurospeech)* vol. II, 1995, pp. 1087–1090

[Lee et al 2000] Lee, Akinobu ; Kawahara, Tatsuya ; Takeda, Kazuya ; Shikano, Kiyohiro: A New Phonetic Tied-Mixture Model for Efficient Decoding. In: *Proc. IEEE Int. Conf. on Acoustics, Speech, and Signal Processing (ICASSP)* vol. III, 2000, pp. 1269–1272

[Linde et al 1980] Linde, Y. ; Buzo, A. ; Gray, R. M.: An algorithm for vector quantization design. 28 (1980), Nr. 1, pp. 84–95

[Ljolje et al 1999] Ljolje, Andrej ; Riley, Michael D. ; Hindle, Donald M.: The AT&T Large Vocabulary Conversational Speech Recognition System. In: *Proc. European Conference on Speech Communication and Technology (Eurospeech)*, 1999, pp. 807–810

[Lloyd 1982] Lloyd, Stuart P.: Least Squares Quantization in PCM. In: *IEEE Transactions on Information Theory* vol. IT-28, March 1982, pp. 129–137

[Mak and Bocchieri 1998] Mak, Brian ; Bocchieri, Enrico: Training of Subspace Distribution Clustering Hidden Markov Model. In: *Proc. IEEE Int. Conf. on Acoustics, Speech, and Signal Processing (ICASSP)* vol. II, 1998, pp. 673–676

[Mak and Bocchieri 2001] Mak, Brian Kan-Wing ; Bocchieri, Enrico: Direct Training of Subspace Distribution Clustering Hidden Markov Model. In: *IEEE Transactions on Speech and Audio Processing* 9 (2001), Nr. 4, pp. 378–387

[Manber 1997] Manber, Udi: A Text Compression Scheme that Allows Fast Searching Directly in the Compressed File. In: *ACM Transactions on Information Systems* vol. 15, April 1997, pp. 124–136

[Maniscalco 2001] Maniscalco, Michael A.: A Second Modified Run Length Encoding Scheme for Blocksort Transformed Data. (2001). – http://www.geocities.com/m99datacompression/papers/rle2.html

[Martin 1994] Martin, Rainer: Spectral Subtraction Based on Minimum Statistics. In: *EUSIPCO*, 1994, pp. 1182–1185

[Marzinzik and Kollmeier 2002] Marzinzik, M. ; Kollmeier, B.: Speech pause detection for noise spectrum estimation by tracking power envelope dynamics. In: *IEEE Transactions on Speech and Audio Processing* 10 (2002), Nr. 2, pp. 109–118

[Mohri et al 2000] Mohri, Mehryar ; Pereira, Fernando ; Riley, Michael: Weighted Finite-State Transducers in Speech Recognition. In: *ISCA ITRW Automatic Speech Recognition: Challenges for the Millenium*, 2000, pp. 97–106

[Mohri et al 2002] Mohri, Mehryar ; Pereira, Fernando ; Riley, Michael: Weighted Finite-State Transducers in Speech Recognition. In: *Computer Speech and Language* 16 (2002), Nr. 1, pp. 69–88

[Mohri and Riley 2002] Mohri, Mehryar ; Riley, Michael: An Efficient Algorithm for the N-Best-String Problem. In: *Proc. Int. Conf. on Spoken Language Processing (ICSLP)*, 2002, pp. 1313–1316

[Moore 1965] Moore, Gordon E.: Cramming more components onto integrated circuits. 38 (1965), Nr. 8, pp. 109–118

[Nakagawa and Horibe 2001] Nakagawa, Seiichi ; Horibe, Yukihisa: A Fast Calculation Method in LVCSRs by Time-Skipping and Clustering of Probability Density Distributions. In: *Proc. European Conference on Speech Communication and Technology (Eurospeech)* vol. II, 2001, pp. 855–858

[Nelson 1989] Nelson, Mark: LZW Data Compression. In: *Dr. Dobb's Journal* (1989), October

[Ney et al 1992] Ney, H. ; Haeb-Umbach, R. ; Tran, B.-H. ; Oerder, M.: Improvements in Beam Search for 10000–Word Continuous Speech Recognition. In: *Proc. IEEE Int. Conf. on Acoustics, Speech, and Signal Processing (ICASSP)* vol. I, 1992, pp. 9–12

[Ney 1984] Ney, Hermann: The Use of a One–Stage Dynamic Programming Algorithm for Connected Word Recognition. In: *IEEE Trans. on Acoustics, Speech, and Signal Processing* 32 (1984), Nr. 2, pp. 263–271

[Ney and Essen 1991] Ney, Hermann ; Essen, Ute: On Smoothing Techniques for Bigram-Based Natural Language Modelling. In: *Proc. IEEE Int. Conf. on Acoustics, Speech, and Signal Processing (ICASSP)*, 1991, pp. 825–828

[Ney et al 1989] Ney, Hermann ; Mergel, Dieter ; Noll, Andreas ; Paeseler, Annedore: Continuous Speech Recognition Using a Stochastic Language Model. In: *ICASSP* (1989), pp. 719–722

[Ney and Ortmanns 1999] Ney, Hermann ; Ortmanns, Stefan: Dynamic Programming Search for Continuous Speech Recognition. In: *IEEE Signal Processing Magazine* 16 (1999), Nr. 5, pp. 64–83

[Ney and Ortmanns 2000] Ney, Hermann ; Ortmanns, Stefan: Progress in Dynamic Programming Search for LVCSR. In: *In Proceedings of the IEEE* 88 (2000), Nr. 8

[Ney et al 1998] Ney, Hermann ; Welling, Lutz ; Ortmanns, Stefan ; Beulen, Klaus ; Wessel, Frank: The RWTH Large Vocabulary Continuous Speech Recognition System. In: *Proc. IEEE Int. Conf. on Acoustics, Speech, and Signal Processing (ICASSP)* vol. III, 1998, pp. 853–856

[Ohm 1995] Ohm, Jens-Rainer: *Digitale Bildcodierung: Repräsentation, Kompression, und Übertragung von Bildsignalen*. Berlin and Heidelberg : Springer Verlag, 1995

[Ortmanns et al 1997a] Ortmanns, S. ; Ney, H. ; Firzlaff, T.: Fast Likelihood Computation Methods for Continuous Mixture Densities in Large Vocabulary Speech Recognition. In: *Proc. European Conference on Speech Communication and Technology (Eurospeech)* vol. 4, 1997, pp. 143–146

[Ortmanns et al 1997b] Ortmanns, Stefan ; Eiden, Andreas ; Ney, Hermann ; Coenen, Norbert: Look-Ahead Techniques for Fast Beam Search. In: *Proc. IEEE Int. Conf. on Acoustics, Speech, and Signal Processing (ICASSP)*, 1997, pp. 1783–11786

[Pennebaker et al 1988] Pennebaker, W. B. ; Mitchell, J. L. ; Langdon, G. G. ; Arps, R. B.: An overview of the basic principles of the Q-coder adaptive binary arithmetic coder. In: *IBM J. Res. Develop.* 32 (1988), pp. 717–752

[Rabiner and Juang 1993] Rabiner, L. ; Juang, B.H.: *Fundamentals of Speech Recognition*. Prentice Hall, Englewood Cliffs, NJ, 1993 (Signal Processing Series)

[Ramírez et al 2004] Ramírez, J. ; Segura, J. C. ; Benítez, C. ; Torre, A. de la ; Rubio, A.: A New Voice Activity Detector Using Subband Order-Statistics Filters for Robust Speech Recognition. In: *Proc. IEEE Int. Conf. on Acoustics, Speech, and Signal Processing (ICASSP)* vol. I, 2004, pp. 849–852

[Rojc and Kacic 2001] Rojc, Matej ; Kacic, Zdravko: Representation of Large Lexica Using Finite-State Transducers for the Multilingual Text-to-Speech Synthesis Systems. In: *Proc. European Conference on Speech Communication and Technology (Eurospeech)*, 2001, pp. 2251–2254

[Shibata et al 2000] Shibata, Yusuke ; Kida, Takuya ; Fukamachi, Shuichi ; Takeda, Masayuki ; Shinohara, Ayumi ; Shinohara, Takeshi ; Arikawa, Setsuo: Speeding Up Pattern Matching by Text Compression. In: *CIAC 2000*, 2000, pp. 306–315

[Sixtus et al 2000] Sixtus, Achim ; Molau, Sirko ; Kanthak, Stephan ; Schlüter, Ralf ; Ney, Herman: Recent improvements of the RWTH large vocabulary speech recognition system on spontaneous speech. In: *Proc. IEEE Int. Conf. on Acoustics, Speech, and Signal Processing (ICASSP)* vol. III, 2000, pp. 1671–1674

[SpeechDat-Web Site 2005] *SpeechDat Web Site*. 2005. – http://www.speechdat.org

[Suontausta et al 2000] Suontausta, Janne ; Häkkinen, Juha ; Viikki, Olli: Fast Decoding in Large Vocabulary Name Dialing. In: *Proc. IEEE Int. Conf. on Acoustics, Speech, and Signal Processing (ICASSP)* vol. III, 2000, pp. 1535–1538

[Surendran et al 2004] Surendran, Arun C. ; Sukittanon, Somsak ; Platt, John: Logistic Discriminative Speech Detectors Using Posterior SNR. In: *Proc. IEEE Int. Conf. on Acoustics, Speech, and Signal Processing (ICASSP)* vol. V, 2004, pp. 625–628

[Takahashi and Sagayama 1995] Takahashi, Satoshi ; Sagayama, Shigeki: Four–Level Tied–Structure for Efficient Representation of Acoustic Modeling. In: *Proc. IEEE Int. Conf. on Acoustics, Speech, and Signal Processing (ICASSP)* vol. I, 1995, pp. 520–523

[Togneri 2002] Togneri, Roberto: *Lecture Notes: Information Theory and Coding*. 2002. – http://www.ee.uwa.edu.au/ roberto/teach/itc314/

[Varga et al 2002] Varga, Imre ; Aalburg, Stefanie ; Andrassy, Bernt ; Astrov, Sergey ; Bauer, Josef G. ; Beaugeant, Christophe ; Geißler, Christian ; Höge, Harald: ASR in Mobile Phones — an Industrial Approach. In: *IEEE Transactions on Speech and Audio Processing* 10 (2002), Nr. 8, pp. 562–569

[Viterbi 1967] Viterbi, A.: Error Bounds for Convolutional Codes and an Asymptotically Optimum Decoding Algorithm. In: *IEEE Trans. on Information Theory* 13 (1967), pp. 260–269

[Watanabe et al 1994] Watanabe, Takao ; Shinoda, Koichi ; Takagi, Keizaburo ; Yamada, Eiko: Speech Recognition Using Tree-Structured Probability Density Function. In: *Proc. Int. Conf. on Spoken Language Processing (ICSLP)* vol. I, 1994, pp. 223–226

[Welch 1984] Welch, Terry: A Technique for High-Performance Data Compression. In: *Computer* (1984), June

[Witschel 1993] Witschel, Petra: Constructing Linguistic Oriented Language Models for Large Vocabulary Speech Recognition. In: *Proc. European Conference on Speech Communication and Technology (Eurospeech)*, 1993, pp. 1199–1202

[Witschel 2000] Witschel, Petra: *Optimierte stochastiche Sprachmodellierung auf linguistischen Klassen*, Fakultät für Mathematik und Informatik der Ludwig-Maximilllians-Universität München, Dissertation, 2000

[Witten et al 1987] Witten, I. H. ; Neal, R. M. ; Cleary, J. G.: Arithmetic coding for data compression. In: *Comm. ACM* 30 (1987), pp. 520–540

[Ziv and Lempel 1977] Ziv, J. ; Lempel, A.: A Universal Algorithm for Sequential Data Compression. In: *IEEE-T-IT* (1977), May