Czech LVCSR and language modelling

Ondřej Glembek

December 21, 2005

Abstract

This article presents an approach to an automatic indexing of lectures being given by a native Czech speaker. The recorded digital audio file is the input for the trained system, whose "knowledge" of the speaker—in the form of the previous trainer—and the content of the lectures—in the form of written lectures—is supposed to ensure proper vocabulary word recognition and the audio-file indexing.

1 Introduction

With expansion and increasing accessibility of technological improvements and inventions, every field of science is affected and gives high potential to further evolution. Voice processing (with its high computation system requirements) is directly dependent on hardware innovations, while other fields of science would profit of having a voice processing tool by hand. There is a list of applications which desire high-quality transparent voice processing features, such as speech recognition, speaker identification, automatic indexing, etc. Their requirements are posed on the type of use as well as on specific parameter specifications. These dependencies force researchers to optimize and find best solutions to their desired needs.

The point of interest in this paper is automatic indexing using large vocabulary continuous speech recognition (LVCSR). Much has been done in this field, however a special and concrete demand arose. The task was to adapt an existing configuration to a concrete language, concrete vocabulary, and (as much as possible) to a concrete person.

2 General principles

Let us first take a look at the basic principles of a voice recognition system. The basic scheme is presented in Fig. 1. The system consists of two parts—the *trainer* and the *recognizer*. In order to build a

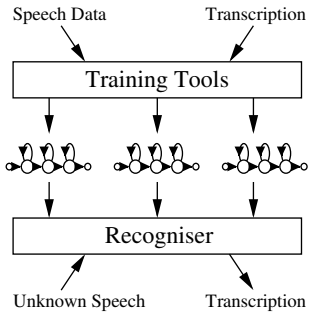

Figure 1: Recognition system schema

recognition system, we need to know, what kind of voice data we are going to deal with. This knowledge is determined by two aspects—*who* is speaking, and *what* the person is talking about. All of this is gathered in the training phase where an appropriate model is created using training utterances and their associated transcriptions. After the system is trained, it is ready to "recognize speech" i.e., to assign a transcription to an unknown speech.

The core of the system is a set of parametric models called the Hidden Markov Models (HMM) and we refer to the system as being HMM-based. It considers the speech signal as a message encoded in a sequence of symbols—*parameters* (sometimes referred to as parameter vectors, features, or coefficients). This topic will be described in Section 4.

Each HMM is a model of an acoustic unit, in our case a *phoneme* or its context dependent variant.[1] The principle is that one HMM *generates* a sequence of parameters which are the building blocks of the phoneme. An HMM model is actually a finite automaton of a start state, the *emitting states* which generate the parameters, and an end state. The start and end states are *non-emitting* i.e, they don't generate any parameters. The transitions between the states, as well as the parameter generation are given by some probabilistic functions. See Fig. 2 for an HMM schema.

The process of recognizing a sequence of units is then defined as finding a sequence of HMMs (a *path* in an HMM space) that are most likely to generate the input sequence of parameters $\mathbf{O}$. Mathematically

$$I^\star = \arg\max_I \{\mathcal{P}(S_I|\mathbf{O})\} \tag{1}$$

[1]Many publications, including [4], start their speech recognition chapters by describing isolated word recognition. There, words are the elementary acoustic units. However, in the LVCSR systems, it is not possible to create an HMM for each single word.

where S is a set of all available acoustic units and I is a sequence of indexes to S.

Unfortunately, there is no way to evaluate $\mathcal{P}(S_I|\mathbf{O})$ directly, but we can use Bayes formula:

$$\mathcal{P}(S|\mathbf{O}) = \frac{\mathcal{P}(\mathbf{O}|S)\mathcal{P}(S)}{\mathcal{P}(\mathbf{O})} \tag{2}$$

where

- $\mathcal{P}(S)$ is the a-priori probability of the acoustic unit (which can be determined by the language model).
- $\mathcal{P}(\mathbf{O})$ is the a-priori probability of the observation sequence. As it is constant for any sequence, it does not affect the maximization problem.
- $\mathcal{P}(S|\mathbf{O})$ is the probability of the sequence $\mathbf{O}$ given the unit S. This probability is evaluated using the HMM.

2.1 HMMs in deep

Let's take a deeper look at Figure 2. As mentioned before, HMM is a finite automaton "generating" sequences of observation.

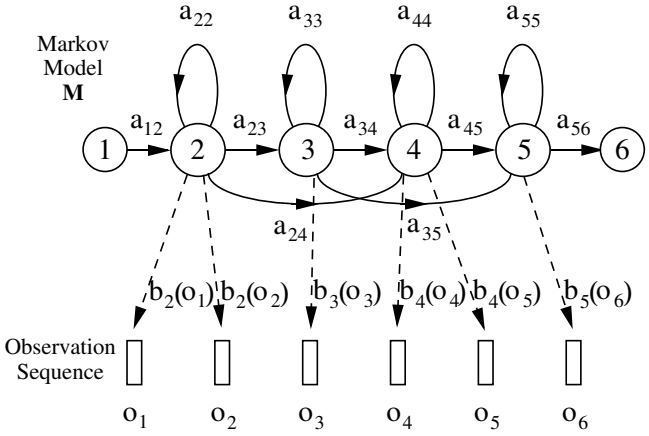

Figure 2: The Markov Generation Model

The a_{ij} symbols represent a probability of transition from state i to state j. This way, the HMM body can be described by a *transition matrix* where rows and columns express the starting and ending states, respectively. It has to be provided that

$$\sum_j a_{ij} = 1. \tag{3}$$

The $b_j(\mathbf{o}_t)$ are called *emission probability density functions* (PDFs) and they give the likelihood that state j generates parameter vector $\mathbf{o}_t$. This function can be either *discrete* (i.e., given by a table of values) or *continuous* (i.e., described by a mathematical function). The latter will be used in this project and is defined by a product of Gaussian distributions:

$$b_j(\mathbf{o}_t) = \prod_{i=1}^{P} \mathcal{N}(\mathbf{o}_t; \mu_{ji}, \sigma_{ji}) = \prod_{i=1}^{P} \frac{1}{\sigma_{ji}\sqrt{(2\pi)}} e^{-\frac{[\mathbf{o}_t - \mu_{ji}]^2}{2\sigma_{ji}^2}} \tag{4}$$

where P is the dimension of the parameter vector $\mathbf{o}_t$, μ_{ji} is the mean, and σ_{ji} is the standard deviation, both for state j and parameter vector element i. Such equation assumes, that the parameters are not correlated. Otherwise, a full covariance matrix equation is used:

$$b_j(\mathbf{o}_t) = \frac{1}{\sqrt{(2\pi)^P |\mathbf{\Sigma}_j|}} e^{-\frac{1}{2}(\mathbf{o}_t - \mu_j)^T \mathbf{\Sigma}_j^{-1} (\mathbf{o}_t - \mu_j)}, \tag{5}$$

where μ_j is the mean vector, $\mathbf{\Sigma}_j$ is the covariance matrix and $|\mathbf{\Sigma}_j|$ is its determinant.

This way, the model is defined by a transition matrix and the matrices of means and standard deviations. In most recognition systems, however, 1-Gaussian models do not suffice and therefore more Gaussian distribution mixtures (sum of more weighted P-dimensional distributions) are used instead. Going even further, the parameter vectors might be divided into *streams*, each having its own Gaussian mixture.

The probability that the model M generates a sequence of parameters $\mathbf{O}$ is given by a sequence X of states it passes. When we defined the recognition as finding the best path, we need to determine the unique probability that the model M generates a sequence $\mathbf{O}$. There are two possible ways to do this:

1. *Baum-Welch probability* is defined as sum of probabilities of all possible paths through the model:

$$\mathcal{P}(\mathbf{O}|M) = \sum_{\{X\}} \mathcal{P}(\mathbf{O}, X|M) \tag{6}$$

2. *Viterbi probability* is defined as the maximum probability of all possible paths through the model:

$$\mathcal{P}^\star(\mathbf{O}|M) = \max_{\{X\}} \mathcal{P}(\mathbf{O}, X|M) \tag{7}$$

Figure 3 shows an example of the possible states transitions with the best path depicted by the bold line.

The elegance of HMMs also lies in their non-emitting states. They do not generate any parameters but they can be used to connect to

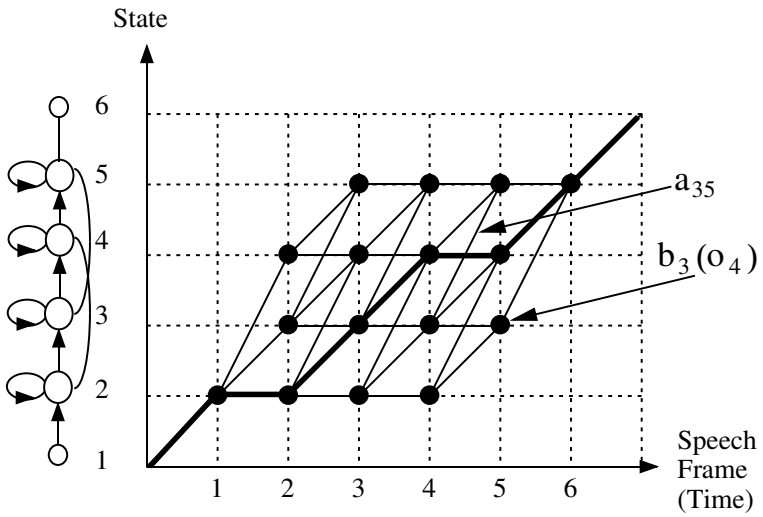

Figure 3: Possible state sequences

another HMM. This way, more HMMs can be glued together and form a complex HMM. The *dictionary* (see Section 5 for details) defines possible connections of phoneme-level HMMs so complex word-level models can be created and words can be recognized. When, in addition, a *language model* is used (see Sec. 6), the system is extended by information about each word's (i.e., complex model) probability of occurrence so the probability maximization problem (i.e., the recognition) does not depend only on acoustic models, but is as well affected by some grammar. The language model probability influence can be controlled by a weight coefficient called the *grammar scale factor*.

Figure 4 illustrates an example of a complex HMM.

2.2 Context-dependent phonemes

One of the most serious problem of the phoneme-based approaches is their *context dependency*. For instance, note the difference of `n` in the word "not" and `n` in the "bank". Let us now define *context-dependent phonemes* as phonemes that depend on their neighbors. In this work, we will use direct left and direct right context phonemes only i.e., the *triphones*. The triphone notation syntax is

```
$triphone = $l - $p + $r
```

where `$l`, resp. `$r` is the left, resp. the right context. Symbol `$p` represents the phoneme. See the example below for the words' "not" and "bank" notation:

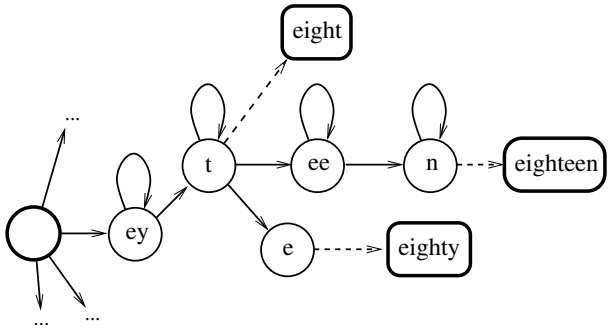

Figure 4: An example of a complex HMM

```
not   ->  n+o n-o+t o-t
bank  ->  b+a b-a+n a-n+k n-k
```

The context dependent phonemes carry a number of difficulties. For example:

- insufficient number of occurrences for the trainer in the training data is available
- assuming P phonemes, there is possibly P^3 triphones (in our system, this number is 10100), thus, it is impossible to train one model for each single triphone.

3 Training the models

As described in previous sections, the models are defined by the transition matrix, the matrix of means, and the matrix of standard deviations, which have to be computed in order to create the models. The algorithm of creation is an iterative process. First a rough estimation is done and then an algorithm of *Baum-Welch re-estimation* is applied. It re-computes the $\mathcal{P}(\mathbf{O}|M)$ using a *forward-backward* algorithm until a desired accuracy is achieved. See [4] for a complete description of the algorithm and a practical example can be found in [9]. Although this training technique can produce high performance system, improvements can be made using speaker adaptation, that is, further re-estimation of the models using one speaker parameters only. This issue is discussed in Section 8.

4 Speech parameterization

In order to be able to describe a time interval on the speech waveform, we need to retrieve its parameters. The process of parameterization splits into several stages.

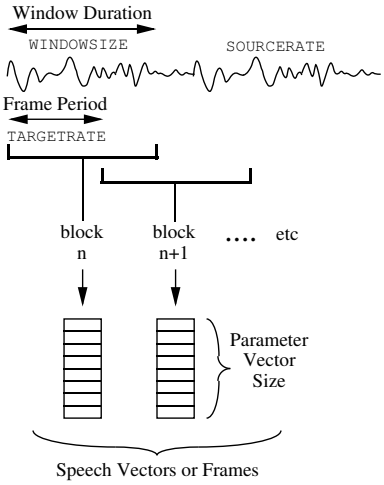

Figure 5: Parameter retrieval

4.1 Segmentation

When analyzing a discrete waveform in time, the characteristics of the currently processed scalar sample cannot be retrieved. Therefore segmentation of the waveform into constant-length intervals—*windows*—is performed. The longer the interval is, the better it describes its characteristics, however the resolution decreases and its attribute of being "almost" stationary disappears. Therefore a compromise time length has to be chosen. Furthermore, the intervals overlap in order not to loose the boundary properties.

4.2 Pre-processing

Independent of what parameter kind is used, there are some pre-processing operations that can be applied before performing the signal analysis:

- DC mean can be removed from the source waveform. The DC offset is usually added by the A–D convertor.

- pre-emphasizing the signal by applying the first order difference equation in each window (i.e., boosting higher frequencies in the signal spectrum).
- tapering of samples in each window by application of the *Hamming window*.

In practice, all three of the above are usually applied.

4.3 Voice activity detection

Voice activity detection (VAD) is a process in which the signal is scanned and segmented to regions of "silence" and "voice". After such segmentation, only the active regions are used for recognition or other power-consuming process while the silent regions are ignored and therefore a lot of resources can be saved.

The correctness of the VAD depends on the decision algorithm which the VAD system uses for determining silence and voice. These algorithms use various approaches, starting with simple amplitude thresholding functions, going through signal spectrum analysis, and ending with highly sophisticated systems (e.g, neural networks, HMMs).

4.4 Parameter retrieval

Uptill now, we haven't exactly specified what the parameters are and how they are retrieved. The choice lies on the user and his specific needs. Among the possible kinds, there are:

- linear prediction coefficients
- filterbank coefficients
- cepstral features
- perceptual linear prediction coefficients

In addition, the parameter stream can be extended by delta, acceleration, and third differential coefficients.

Most often, cepstral features in the form of Mel-Frequency Cepstral Coefficients (MFCCs) are required. In this project, MFCCs are the "core parameters" used for the speech recognition. Filterbank coefficients are used in the supporting layer algorithms, though.

No matter what kind of parameters is used, there lies a detailed underlying theory behind each. Let us look at the ones used in this project.

4.4.1 Filterbank analysis

The human ear resolves frequencies non-linearly across the audio spectrum and empirical evidence suggests to design and use a similar non-

linear front-end in order to improve recognition performance. Filterbank provides a straightforward way to obtaining the non-linear frequency resolution. However, filterbank amplitudes are highly correlated and thus, the use of cepstral features is practically mandatory and inevitable if the data is to be used in a HMM based recognizer with diagonal covariances.

Fig. 6 illustrates the general form of a filterbank, which gives approximately equal resolution on a *mel-scale*[2]. The filters used are tri-angular and they are equally spaced along the mel-scale. To implement such filterbank, the window is transformed using a Fourier transformation and the magnitude (or alternatively power) is taken. The magnitude coefficients are then *binned* by correlating them with each triangular filter (i.e., each FFT magnitude coefficient is multiplied by the corresponding filter gain and the results are accumulated). Each bin, then, holds a weighted sum representing the spectral magnitude in that filterbank channel.

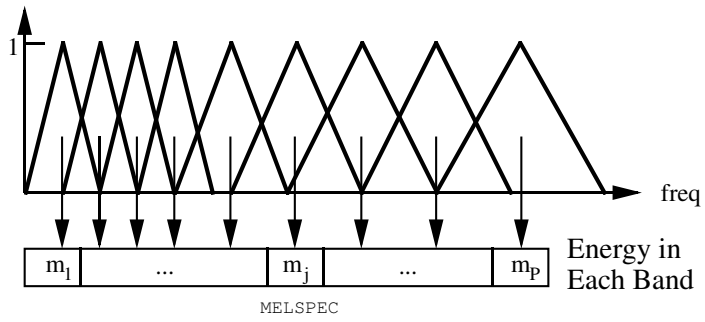

Figure 6: Mel-Scale Filter Bank

4.4.2 Cepstral analysis

Currently, cepstral coefficients are the most often used parameters in speech recognition. They are indicated by setting the target to Mel-Frequency Cepstral Coefficients (MFCCs). They are realized by discrete cosine transform (DCT) with mathematical notation

$$c_{mf}(n) = \sum_{i=1}^{N} \log m_i \cos\left(n(i-0.5)\frac{\pi}{N}\right) \ , \tag{8}$$

where N is the number of filterbank channels m_i.

[2]Mel-scale is a logarithmic-like function corresponding to the human ear frequency sensitivity.

MFCCs give good discrimination and lend themselves to a number of manipulation. In comparison to classical filtration where spectra were multiplied, thus hiding the factor components, the MFCCs' log domain turns the multiplication into much simpler addition, which can be removed by subtracting the cepstral mean value from the input vectors[3].

5 The Dictionary

The dictionary determines the way in which each word is expanded to the level of phonemes. It is a list of records, each containing a single word and its attributes needed for the recognition[4].

Each line of the dictionary file follows this format:

```
WORD [ '['OUTSYM']' ] [PRONPROB] P1 P2 P3 P4 ....
```

where `WORD` represents the word, followed by the optional parameters `OUTSYM` and `PRONPROB`, where `OUTSYM` is the symbol to output when that word is recognized[5] and `PRONPROB` is the pronunciation probability (0.0–1.0) as one word might have more than pronunciation. `P1`, `P2`, ... is the sequence of phonemes or HMMs to be used in recognizing that word. The output symbol and the pronunciation probability are optional. If an output symbol is not specified, the name of the word itself is output. If a pronunciation probability is not specified then a default of 1.0 is assumed. Empty square brackets, `[]`, can be used to suppress any output when that word is recognized (e.g., a silence or a noise). For example, a dictionary might contain

```
bit           b  ih t
but           b  ah t
dog    [woof] d  ao g
cat    [meow] k  ae t
start  []     sil
end    []     sil
```

If a word has more than one pronunciation, then multiple records are added:

```
the           th iy
the           th ax
```

[3]This process is called *Cepstral Mean Normalization* (CMN). The mean is estimated by computing the average of each cepstral parameter across each input speech file.

[4]Uptill now we haven't talked about the HTK tool yet, however we will start to demonstrate the problems in the context of HTK and give an appropriate explanation where necessary for future research.

[5]HTK requires this to be enclosed in square brackets, `[` and `]`

corresponding to the stressed and unstressed forms of the word "the".

The dictionary can also include the context-dependent models. For example, for the words bit and but, the records might be written as

```
bit             b+ih  b-ih+t ih-t
but             b+ah  b-ah+t ah-t
```

There are tools available in the HTK that can perform the context expansion automatically, however it might be helpful to set these records explicitly as performance increases.

To summarize this section, the dictionary tells the recognition system about all possible phoneme combinations in the meaning of building blocks for possible words.

6 The Language model

Language model, in our context, is a list of pairs, whose first field is an n-gram (i.e., sequence of n symbols), and the second field is a number representing a probability of occurrence (the n-gram probability)[6]. Such facility is called the *n-gram language model* (LM) and is used to predict each symbol in the sequence given its $n-1$ predecessors.

The creation of LM assumes that the probability of a specific n-gram occurrence in the input text can be estimated from the frequency of its previous occurrence in some training text. The process of building the LM is done in three stages.

1. The *appropriate* training text is scanned and the n-grams are counted and stored in a database of gram files.
2. Classification, mapping, and sequencing is performed.
3. The counts are used to compute the desired probabilities.

Later in the text, we will talk about two similar things—the n-gram counts and LM itself. The difference between these two facilities is in the number associated with the n-gram. While n-gram count gives an integer occurrence count (i.e., how many times the n-gram occurred in the scanned text), the LM gives a logarithmic probability of the n-gram occurrence.

To be able to say, how good the LM is, we can define a measure called *perplexity*. It expresses the similarity between the LM and a model that would be created from an unseen text-set. In simple words, the smaller the test-set perplexity, the better the model. The theory of the LMs is broad and is well described in [4].

Let's go back to the previous paragraph. Note the word *appropriate* in point 1. The reason that we have included it there, is that the use

[6]The physical order in a file is a little different. The first field is occupied by the probability and the rest of the record is reserved for the n-gram.

of an essentially static and finite training text makes it difficult to generate a single LM, which would be well-matched to varying test material. Suppose, we have a well trained LM on newspaper text. Such LM would be a good predictor for the dictating news reports but when used with a predictor for a recipe-book, it would give poor results.

This issue is the point of interest of this paper. The idea is to use sophisticated techniques in LM training. See Section 10 for details on our approach to Czech LM trining.

7 Recognition and LVCSR

7.1 The Network

The recognition network is a list of sequences of words that can be recognized and it controls the operation of decoding. It is compiled from word-level network, a dictionary and a set of HMMs. A word-level network typically represents either a *task grammar* which defines all of the legal word sequences explicitly or a *word loop* which simply puts all words of the vocabulary in a loop and therefore allows any word to follow any other word. Such configuration allows for *continuous speech recognition* which accepts any word as an input. An example of such network is shown in Fig. 7.

Word-loop networks are often—including our experiments—augmented by an LM. Networks can also be used to define phone recognizers and various types of word-spotting systems, which is not the topic of this work.

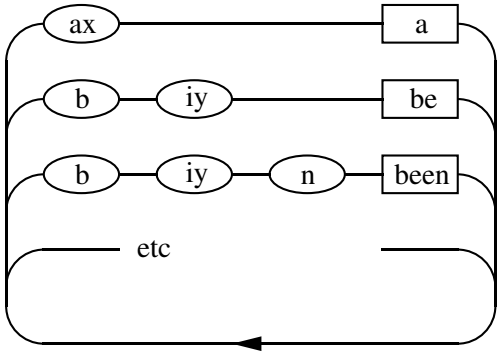

Figure 7: Recognition Network for Continuous Word Recognition

7.2 Recognition

Recognition problem of finding a solution to Eq. 1 is usually solved by maximizing the Viterbi probability. This problem is usually implemented using *token passing* approach. In addition, we will treat the probabilities in a logarithmic manner to avoid floating point out-of-range problems. The key steps of the algorithm are as follows:

1. **Initialize**: set tokens in all initial states (usually one state) to zero.
2. **Iterate**:
   (a) Pass a copy of every token in state i to all connecting states j, incrementing the logarithmic probability of the copy by $\log a_{ij} + \log b_j[\mathbf{o}_t]$.
   (b) Examine the tokens in all states and discard all but the one with the highest probability
   (c) If end-of-word state is observed then:
      i. language model probability is picked from the LM, scaled by the *grammar scale factor* and added to the token's probability field
      ii. a parameter called the *word insertion penalty* is added to the token's probability field
      iii. the word is written to the token's list of words
3. **Terminate** Examine the tokens in the ending state N and discard all but the one with the highest probability, whose value is equal to $\log \mathcal{P}^{\star}(\mathbf{O}|M)$. The token's list of words is then equal to the recognized word sequence.

The behavior of tokens can be controlled by a constraint called *pruning*. Pruning sets a threshold, which excludes all tokens, whose value is worse than the best token minus the pruning value. It can be set either hardly (i.e., we set the final number), or softly (i.e., we set the starting number, which can be increased by given step in case that success is not achieved, until a final limit is reached).

Sometimes, we are not interested in the best path only but we would like to obtain a *lattice* of hypotheses. This is achieved by storing more than just the best token in the end-of-word state. A word-level graph is created this way. An example is shown in Fig. 8. The edges express the hypotheses and store not only the word, but they may also hold its acoustic and language costs.

7.3 Operation modes

The recognizer input control in the form of the network allows three modes of operation, all of which will be used in the work:

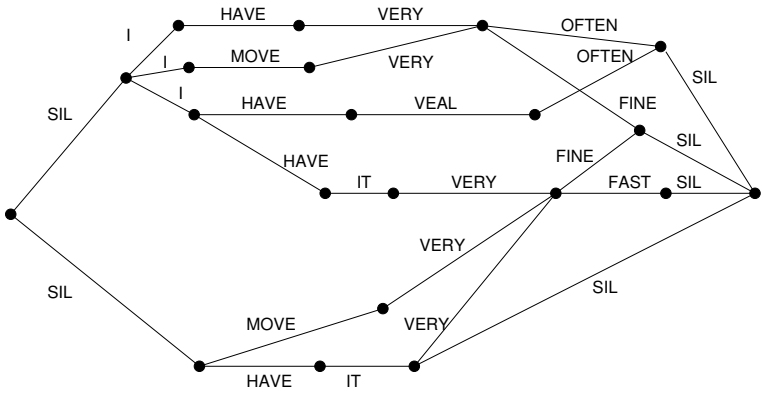

Figure 8: Lattice sample

1. *Recognition*

   This is the standard operation, described in previous chapter. In recognition, the recognition network is compiled from a task-level word network.

2. *Forced alignment*

   Here, the recognition network is created from a word transcription and a dictionary. The output is a lower level (phoneme) transcription aligned in time. This mode will be useful during speech adaptation which requires the phoneme-level transcription as input.

3. *Lattice rescoring*

   In this case, the network is a lattice generated previously by a recognizer. This mode will be useful during experiments as rescoring is less resource and time consuming than recognizing. Rescoring is then used to quickly evaluate the performance and effectiveness of recognition techniques.

7.4 Recognizer output

The recognizer can give its output in two forms. Either we want the real best path in the form of a *label file*, or we might want to experiment or tune the system using the *lattices*.

- The label file, or the master label file, is a structured text file of recognized blocks, where the first level holds the recognized block—the block's name is specified here as well—and the second level (i.e., the content of the block), is the transcription it self. The transcription has the following structure:

```
start_frame end_frame recognized_word
```

- Sometimes, however, we want to see deeper into the process of recognition. It would be nice to see not only the best path in the search but maybe bigger variety of probable paths could be interesting. In that case we want to create the *lattices* (see Fig. 8 for illustration). The structure, or better say the size of the graph is given by the parameters passed to the decoder.

8 Speaker adaptation

Even though the training of the system can be done very precisely, there is always a gap caused by not knowing the speaker (i.e., we don't know the parameters of his/her vocal tract). If we, however, have the information, we can adapt the trained models to that speaker and improve the system's performance. This turns the *speaker independent* recognizer into a *speaker dependent* system.

Speaker adaptation can be performed in different modes. If a real transcription of the adaptation data is given, then we talk about it as *supervised adaptation*, while there are techniques which don't require labeled data and are called *unsupervised adaptation*. Another classification could be considered when we have a single block of adaptation data (*static adaptation*) or the data is coming in batches (*incremental adaptation*).

It was a planned part of this project to transcribe some of the wave data, and perform an adaptation using these data. Concerning the above modes, our adaptation can be classified as *supervised static adaptation*, so we will concentrate the theory explanation just in this scope. The full description is in [4].

There are two techniques of adaptation, the first of which is *maximum likehood linear regression* (MLLR) and the second is *maximum a-posteriori* (MAP) adaptation. MLLR adaptation can be applied in both incremental and static modes, while MAP supports only the static mode.

A procedure called *forced alignment* usually needs to be done before anything else is started. The problem is, that the transcribed data are presented at word-level, while the HMMs are phoneme-based. The forced alignment algorithm expands the words into context-dependent phonemes using the dictionary, and assigns time stamps to them.

Model training is done in steps—*iterations*. With each iteration, the model is more likely to fit the speaker's voice, because the previous result[7] is re-adapted by the training data. However, it has to

[7] in the meaning of models from the previous iteration. When talking about the first iteration, the unadapted, original models are meant.

be provided, that the models are not over-trained. In that case (especially., when not enough training data is available), the models are very unflexible. When the speaker's voice changes only in a slight extent (e.g., due to illness, acoustic environment, microphone set-up, etc.), the models fail during the recognition by being too exact.

Let's take a closer look at the two techniques.

8.1 MLLR

MLLR is based on computing a set of transformations that reduce the mismatch or the difference between the initial model definition and the adaptation data. More specifically, MLLR technique estimates a set of linear transformations for the mean and the variance parameters of a Gaussian mixture of the HMM system. The aim is to shift the the component means and alter the variances in the initial system so that each state in the HMM system is more likely to generate the adaptation data.

MLLR technique can be applied in various manners depending on the amount of available adaptation data. If only a "small" amount of data is available, then a global transformation can be generated (i.e., a global transform is applied to every Gaussian component in the HMM model set). As more adaptation data are coming, improved adaptation is possible by increasing the number of transformations. Each transformation is now more specific and is applied only to a certain groupings of Gaussian components (e.g., silence, vowels, stops, glides, nasals, etc.—each can be transformed individually).

MLLR makes use of the *regression class tree* to group Gaussian mixtures in the HMM set. The tree is constructed in a way that acoustically similar models are clustered together. The deeper the tree node (i.e., the cluster) is, the more specific group of model can be clustered and adapted. Each terminal node (*base regression class*) corresponds to one Gaussian component. The adaptation, then, is performed in a top-down manner. The tree can grow as long as there is enough data for each cluster.

8.2 MAP

This adaptation technique is sometimes referred to as Bayesian adaptation. MAP adaptation makes use of a prior knowledge about the model parameter distribution. Therefore, if we know what the parameters of the model are likely to be (before observing any adaptation data) using the prior knowledge, we might be able to make good use of the limited adaptation data to obtain a decent MAP estimate. This type of prior is also known as *informative prior*. For MAP adaptation purposes, the informative priors that are generally used are the

speaker-independent model parameters.

The update formula for a single stream system for state j and mixture m is:

$$\hat{\mu}_{jm} = \frac{N_{jm}}{N_{jm} + \tau}\bar{\mu}_{jm} + \frac{\tau}{N_{jm} + \tau}\mu_{jm} \tag{9}$$

where μ_{jm} is the speaker independent mean, $\bar{\mu}_{jm}$ is the mean of the observed adaptation data, N_{jm} is the occupation likehood of the adaptation data. The τ parameter represents a weighting of the a priori knowledge to the adaptation speech data. The higher the τ is, the less the speaker independent models are affected.

To point out the main idea, while MLLR moves all Gaussians specified by the regression tree, MAP transforms only those models, that it has enough information about.

9 Overal System Diagram

The overall system schema is shown of Fig. 9. It comprises all units that were discussed in the previous sections.

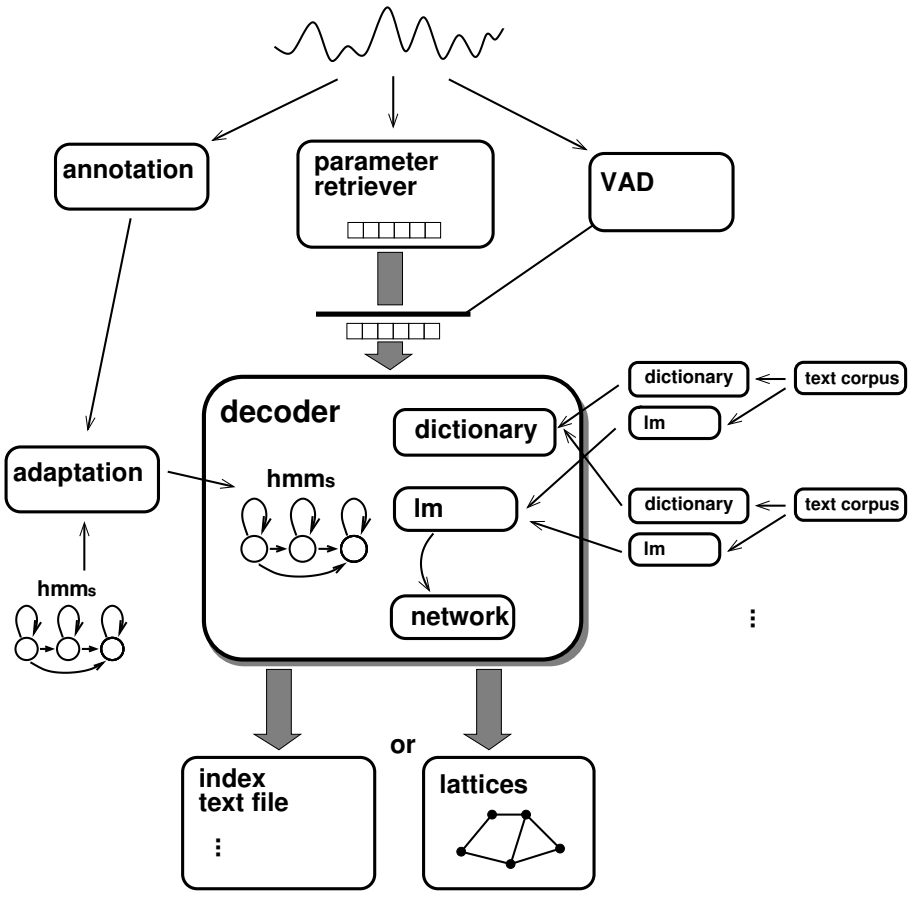

Figure 9: The overall system schema

10 Research

The previous chapters describe an LVCSR system which was implemented and used for research. The aim is to get recognition *accuracy* as high as possible. This is achieved by improving and tuning all parts of the system. We focused on language model training.

As every speech is specific for its vocabulary it is clear that the language model needs to be adapted to it. The primary task was to build an LVCSR system for automatic lecture indexing. The two main problems connected with this task are

- every lecturer uses different vocabulary and way of expression
- every subject reflects different vocabulary

The first problem could be solved by creating an LM for every lecturer and could be given to the LVCSR system as a parameter. However such LM would have to be explicitly created which means the person's speech would have to be recorded and manually transcribed into text form. On one hand, this task is very expensive. On the other hand, once such LM is created, the lecturer can use it forever. Currently, we don't have possibilities to train such LM's and thus we will not consider this problem. Also, when a lecture is given, usually only a special vocabulary is used which does not give the lecturer a lot of freedom in usingsing his own way of expression.

The second problem can be solved by creating an LM either from the transcribed spoken text or by parsing some written text. In our project we perform the first of these options as manual transcription was inevitable step for speaker adaptation. However in real world, this moves us to the first problem discussed above. An option is to parse available written text. This is extremely comfortable as usually plenty of litarature to a given subject is available. The only difficulties might be in parsing mathematical or other special symbols, for which parsers are currently being developed.

The resulting LM covers most of the special vocabulary comprised in the lecture, however the lecturer uses colloquial way of expression and usually doesn't strictly use the "literature language". The point of interest so far was to find out, how merging of LM's improves recognition. The idea is to have a general LM, which models the most common speech, and a special LM which is specific to the recognition task i.e., either to the speaker or the subject.

Merging of LM's was done using the SRILM toolkit and the principle lies in redistribution of probabilities of the special (lecture) LM in the general LM. The merging can be scaled i.e., different weight can be assigned to each LM. Let λ be the weight of the universal LM.

We have mixed the LMs and watched the perplexity and out-of-vocabulary word rate on a testing set. The following table shows the

result.

Figure 10 illustrates the same results in graph, only for the perplexity, though.

Table 1: LM perplexity result

universal LM (λ)	lecture LM	perplexity	OOV
0.00	1.00	173.322	1043
0.10	0.90	416.503	151
0.25	0.75	383.069	151
0.50	0.50	394.229	151
0.75	0.25	480.457	151
0.90	0.10	635.235	151
1.00	0.00	1213.53	368

It is interesting, that the lecture LM alone gave the smallest perplexity, while the universal LM application resulted in enormously high perplexity. In [7], the authors observed the same thing.

We expected the trend of the perplexity to be increasing in all of its extent, however, there is a "bump" at $\lambda = 0.1$. This is a critical point, where mixing of the two LMs gave unexpected results, i.e. the value of the n-gram probability was `NAN`. The reason is that the lowest n-gram probabilities were so low, that decreasing their importance by setting λ to 0.1 and not having them included in the lecture LM caused the float-point numbers to cross to the zero values. Such result was obviously unacceptable as the recognizer couldn't handle that, so we had to set pruning options.

The OOV parameter shows, that there are words in the testing corpus, that none of the two LM contains. The universal LM alone did better than the lecture LM alone, though. This result is caused by the fact, that the lecture LM misses 1043 words, which it is supposed to recognize. The universal LM's vocabulary is not perfect as well, but great improvement is noticeable. These numbers probably show that the universal LM covers most of the basic vocabulary and only needs to be refined by some small specialial text corpus.

Note that $0 < \lambda < 1$ does not have any impact on the LM vocabulary, only the LM perplexities are different. Mixing the LMs did not eliminate any words from neither LM.

10.1 Results in Recognition

In this experiment, we wanted to find out, which of the (either mixed or stand-alone) LMs would give the best results in decoding (i.e., the

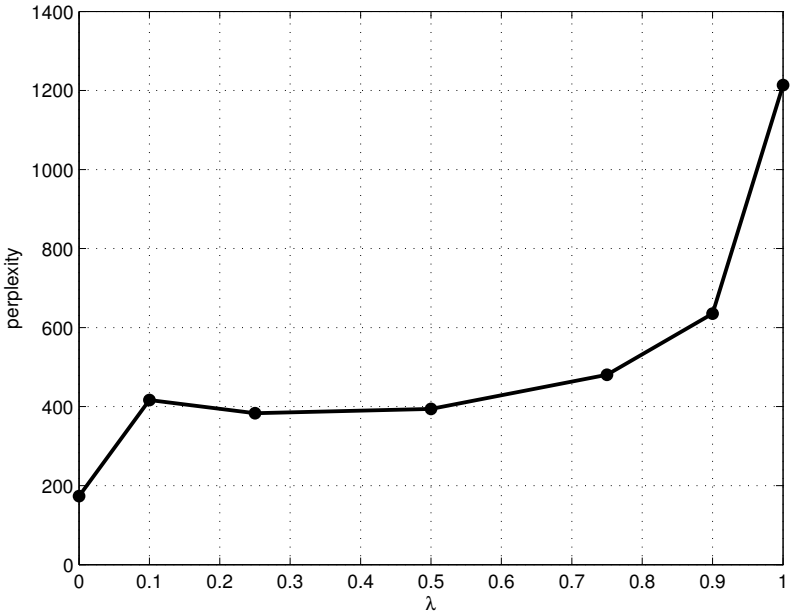

Figure 10: Perplexity of LMs at different mixture Lambda

accuracy of decoding). We chose a single 32-Gaussian, MAP adapted HMM set and we have run the decoding with constant parameter settings. We set the grammar scale factor to 12 and the insertion penalty to -10. We have run seven tests, each for different value of λ. The following listing shows the result statistics and Fig. 11 illustrates the accuracy in graph.

We see, that even though the lowest perplexity was measured when $\lambda = 0$, the accuracy does not follow this trend. The expected result was that the lower the perplexity would be, the higher accuracy would be achieved. However, and [7] observed the same results, the opposite (except for $\lambda = 1$) turned out to be true.

As for the stand-alone LMs, the universal LM shows better results than the lecture LM. This is caused by the fact, that although the lecture LM fits the testing data better, the universal LM covers much more of the testing vocabulary and the recognizer is able to find the missing words.

Table 2: LM perplexity result

λ	Accuracy
0.00	35.76
0.10	42.37
0.25	43.86
0.50	45.60
0.75	46.89
0.90	46.93
1.00	43.35

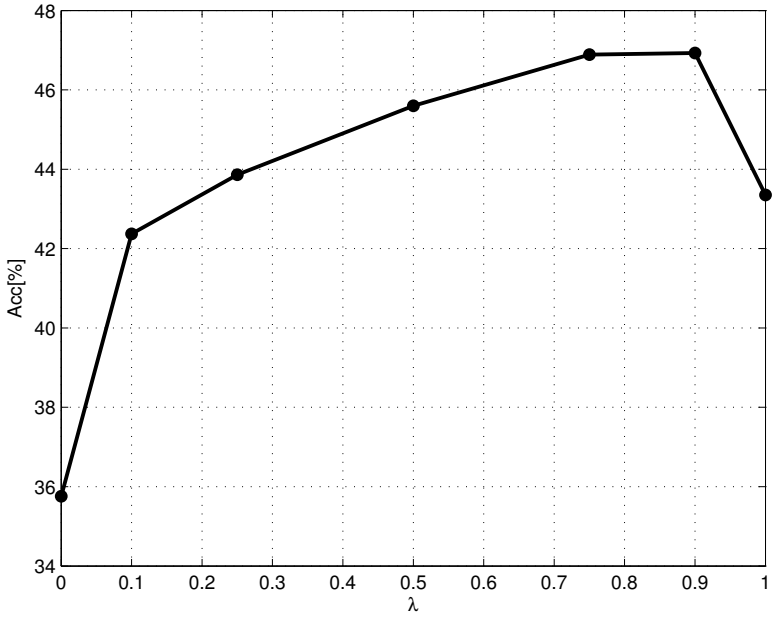

Figure 11: LM accuracy according to λ

11 Conclusion

The paper gives an outline to large vocabulary speech recognition system synthesis and introduces an approach to language model training. The language model was created by merging the universal and special models. With the primary configuration, we found out that using the special LM gives the poorest results (accuracy of 35.76%) as it comprises very specific vocabulary. The best results were achieved when the universal and special models were mixed with weights $\lambda = 0.75$ (accuracy of 46.89%). Such system was built and after acoustic adaptation, the highest reached accuracy was 63.01%.

There is an issue about Czech language as such. It belongs to a group of *inflexional* languages and thus needs special treatment. The words need to be split into logical sub units so that higher range of words is modelled more efficiently. We are working on simple stem-inflection models and we expect further improvements. Such models are based on splitting the words into the stem and inflection which gives the model some hierarchy so larger vocabulary can be comprised in (pysically) smaller but more accurate language model.

References

[1] Černocký, J.: *Hidden Markov Models – an Introduction* [Lecture background to the Digital Speech Processing (CZR) course]. Brno University of Technology, Faculty of Information Technology. Document accessible at URL http://www.fit.vutbr.cz/~cernocky/oldspeech/lectures/hmm.pdf, January 2005.

[2] Černocký, J.: *Complements to Speech Recognition Using Hidden Markov Models* [Lecture background to the Digital Speech Processing (CZR) course]. Brno University of Technology, Faculty of Information Technology. Document accessible at URL http://www.fit.vutbr.cz/~cernocky/oldspeech/lectures/hmm.pdf, January 2005.

[3] Černocký, J.: *Cepstrální analýza řeči a její aplikace* [Lecture background to the Digital Speech Processing (CZR) course]. Brno University of Technology, Faculty of Information Technology. Document accessible at URL http://www.fit.vutbr.cz/~cernocky/oldspeech/lectures/ceps.pdf, January 2005.

[4] Young, S., Evermann, G., Hain, T., Kershaw, D., Odell, J., Ollason, D., Povey, D., Valtchev, V., and Woodland, P.: *The HTK Book*. Entropics Cambridge Research Lab., Cambridge, UK, 1996

[5] Moore, J., Kronenthal, M., Ashby, S.: *Guidelines for AMI Speech Transcriptions*, 10 February 2005, Version 1.2

[6] Willet, D., Neukirchen, C., Rigoll, G.: *DUCODER — THE DUISBURG UNIVERSITY LVCSR STACKDECODER*. Department of Computer Science, Faculty of Electrical Engineering, Gerhard-Mercator-University, Duisburg, Germany

[7] Park, A., Hazen, T.J., Glass, J.R.:*Automatic Processing of Audio Lectures for Information Retrieval: Vocabulary selection and Language Modeling*, MIT Computer Science and Artificial Intelligence Laboratory, 32 Vassar Street, Cambridge, MA 02139, USA, 2005

[8] URL: http://www.speech.sri.com

[9] Chalupníček, K.: *Test rozpoznávače z projektu COST 249 na české databázi SpeechDat-E* [Year project], Brno University of Technology, Faculty of Electrical Engineering and Communication, 2001/2002

[10] Modlisation et Exprimentation pour le Traitement des Informations et des Signaux Sonores METISS [2004 research project activity reports], URL: http://www.inria.fr/rapportsactivite/RA2004/metiss2004/metiss.html (9.5.2005)