

Multilingual man-machine communication

Contents

1	Overview	5
2	Introduction	6
2.1	Why do we need Automatic Speech Recognition (ASR)?	6
2.2	Where is Speech Recognition and Understanding useful?	6
2.2.1	Human-Machine Interaction	6
2.2.2	Human-Human Interaction	6
3	Basics	7
3.1	Why is ASR so difficult?	7
3.2	Speech units	7
3.3	Problems with different Languages	7
3.4	k -means clustering	8
3.5	Comparing two utterances	8
4	Signal processing	9
4.1	How can we extract features?	9
4.2	Acoustic Features in sampled signal	9
4.3	Frequency domain and Fourier transformation	9
4.4	Short-Term Spectral Analysis	10
4.5	Linear Predictive Coding	10
4.6	Cepstra	10
4.6.1	Mel-frequency cepstrum (MFC)	11
4.7	Feature vectors	11
5	Hidden Markov Models	12
5.1	Why do we use HMMs?	12
5.2	The three great problems	12
5.2.1	Evaluation	12
5.2.2	Decoding	12
5.2.3	The Backward algorithm	13
5.2.4	Learning / optimization	13
5.3	HMM state tying	14
5.3.1	Fully-continuous HMMs	14
5.3.2	Semi-continuous HMMs	14
5.3.3	Phonetically-tied semi-continuous HMMs	15
5.4	HMM training	15
5.5	HMM parameter initialization	15
6	Acoustic modeling	16
6.1	Discrete HMMs in continuous space	16
6.2	Source coding	16
6.3	Continuous HMMs	16
6.3.1	Conversion to semi-continuous model	16
6.3.2	Conversion to shared semi-continuous model	16
6.3.3	Conversion to tied semi-continuous model	16
6.4	Parameter tying	17
6.5	Lexicon	17
6.5.1	Pronunciation variants	17
6.6	Context dependent acoustic modeling	17
6.6.1	Crossword context modeling	18
6.6.2	Position dependent modeling	18
6.6.3	Parameter tying	18

6.7	Clustering	18
6.7.1	Continuous parametric models	18
6.7.2	Discrete models	19
6.7.3	Kai-Fu Lee	19
6.7.4	Decision trees	19
7	Language modeling	21
7.1	Deterministic vs. Stochastic Language Models	21
7.2	N-grams	21
7.2.1	Bigrams vs. Trigrams	22
7.3	Perplexity	22
7.4	Smoothing	22
7.4.1	“Add-one” smoothing	22
7.4.2	Backoff smoothing	22
7.4.3	Linear interpolation	24
7.5	n -gram classes	24
7.6	Different kinds of language models	24
7.6.1	Cache language models	24
7.6.2	Trigger language models	24
7.6.3	Multilevel language models	24
7.6.4	Interleaved language models	25
7.6.5	Morpheme-based language models	25
7.6.6	Context-free grammar language models	25
7.6.7	Tree-based language models	25
7.6.8	HMMs for language modeling	25
7.7	Vocabulary selection	26
7.8	n -gram pruning	26
7.9	Problems with spontaneous speech	26
7.10	Unknown words	26
7.11	OOV words	27
7.12	Problems with different languages	27
7.13	Problems with Recognition Errors	27
7.14	Language model adaptation	27
7.14.1	Retrieval of adaptation data	27
7.14.2	Model interpolation	28
7.14.3	Constraint specification	28
7.14.4	Meta-information extraction	28
8	Search / Decoding	29
8.1	DTW optimization	29
8.2	Viterbi optimization	30
8.3	From Isolated to Continuous Speech	30
8.3.1	Cut Continuous Speech into Single Words	31
8.3.2	Two-level DTW	31
8.3.3	Depth-first search	31
8.3.4	Viterbi beam search vs. A^* stack decoder	31
8.3.5	One stage dynamic programming	32
8.4	Optimization	32
8.4.1	Pruning with Beam search	32
8.5	Searching with LMs	32
8.5.1	Problems	32
8.5.2	Search with vs. without Language Model	32
8.5.3	Tree-Search with LMs	32
8.5.4	Unigram lookahead	33

8.5.5	Multi-pass searches	33
8.5.6	Multiple hypothesises	34
8.6	Output Formats	34
8.7	Confidence Measures	34
8.8	Search with Context-Dependent Models	34
8.9	Search with context dependent phonemes	34
9	Text-to-speech synthesis	35
9.1	Text analysis	35
9.2	Linguistic analysis	35
9.2.1	Bootstrapping	36
9.2.2	Prosody	36
9.3	Waveform synthesis	36
9.3.1	Diphones	36
9.3.2	Unit selection synthesis	36
9.3.3	Cluster unit selection	37
9.3.4	Build your own Synthetic Voices	37
9.3.5	Multilingual Speech Synthesis	37
10	Spoken dialog systems	38
10.1	Dialog strategies:	38
11	Multilingual speech processing	40
12	Prüfungsfragen	42
12.1	Allgemeine Fragen	42
12.2	Vorverarbeitung	42
12.3	Akkustisches Modell	44
12.4	Sprachmodell	45

1 Overview

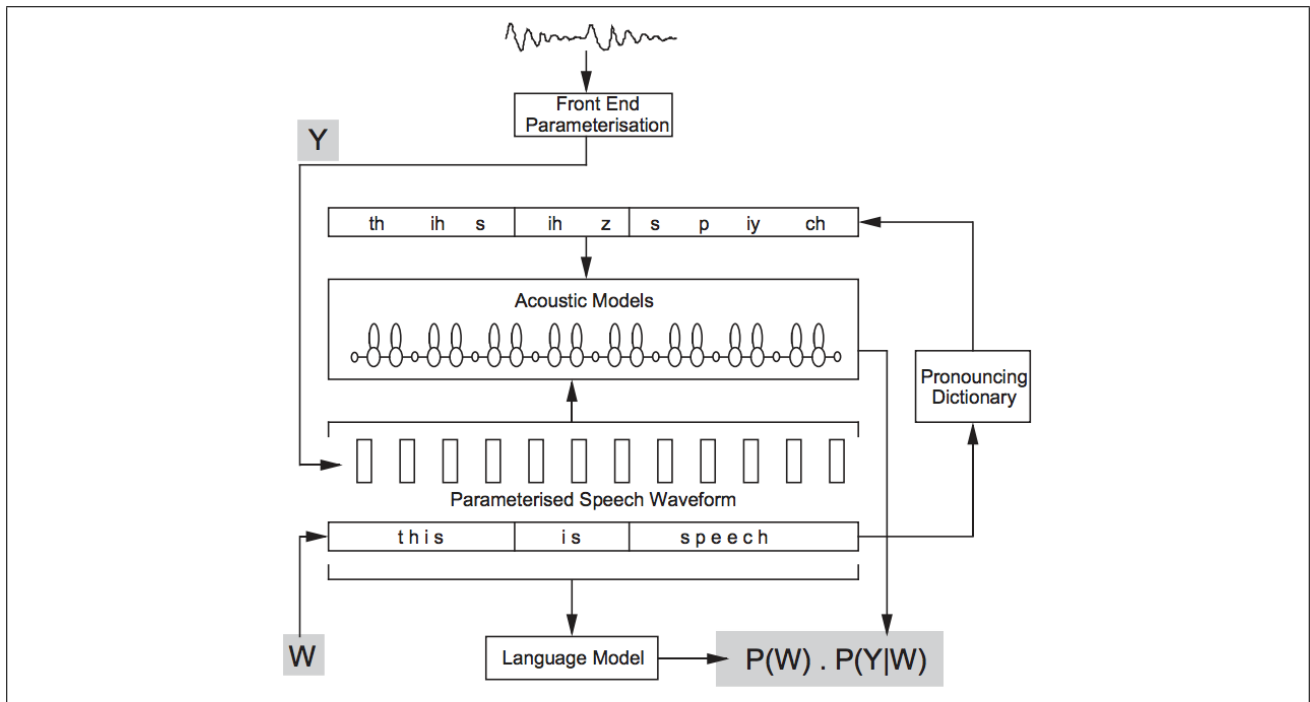


Figure 1: Overview according to Young

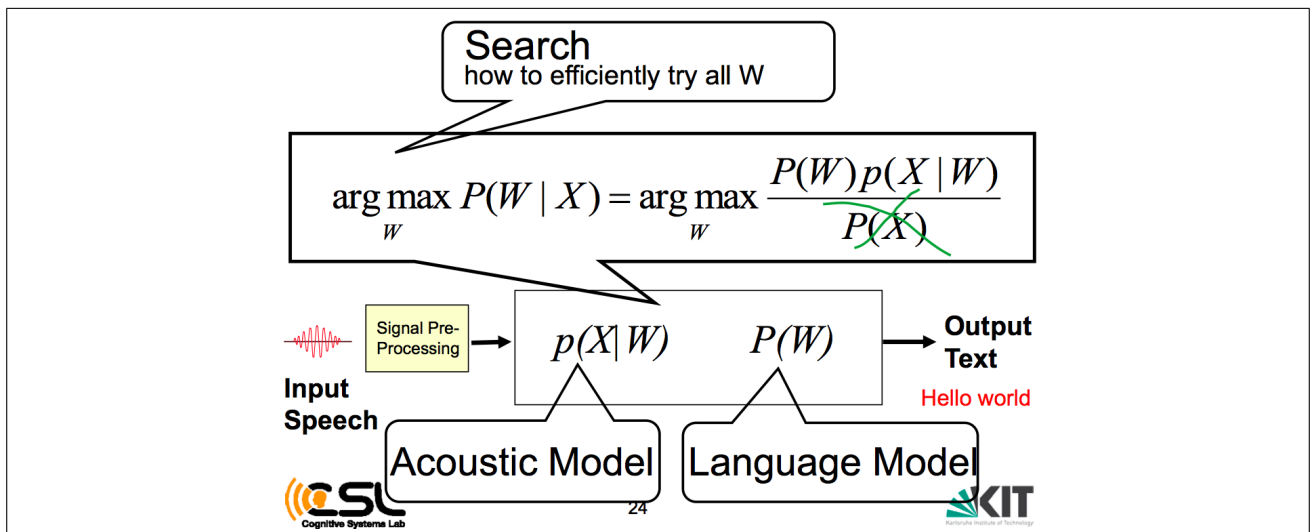


Figure 2: Overview according to Schultz

2 Introduction

2.1 Why do we need Automatic Speech Recognition (ASR)?

Advantages	Disadvantages
Speech is a natural way of communication Allows an additional communication channel Hands and eyes are free for other tasks Mobility Some communication channels are speech-only	When you are not allowed to speak, it is unusable Unsatisfactory recognition rate Problems with accents, dialects and code-switching Cultural factors (e.g. uncertainty avoidance) Speech input is still more expensive




Mode	Speed	Reliability	Devices	Practice	Other Tasks
Handwriting 	200-500	Recognition errors	Tabloid, scanner, BIG	Literacy	Hands and eyes busy
Typing 	200-1000	~100% Typos	keyboard, BIG	Literacy, yes for fast input	Hands and eyes busy
Speech 	up to 4000	Recognition errors	Micro, SMALL	NO	Hands and eyes free

Figure 3: Input Characters per minute

2.2 Where is Speech Recognition and Understanding useful?

2.2.1 Human-Machine Interaction

- Applications for Remote Control
 - Using the phone to operate machines
- Hands/eyes are busy or not usable
 - Speech recognition while driving
 - Nurse bots for physically disabled
- Authentication
 - Speaker Identification/Verification/Segmentation
 - Language/Accent Identification
- Entertainment
- Indexing and Transcribing Acoustic Documents
 - Archive, Summarize, Search and Data Retrieval

2.2.2 Human-Human Interaction

- Communication across language barriers
 - Simultaneously translating one language into the other
- Support human interaction
 - Meeting and lecture systems
 - Speech therapy support

3 Basics

3.1 Why is ASR so difficult?

- Complexity
 - typically 32000 Bytes per second (**amount of data**)
 - 50 phonemes, 5000 sounds, 100000 words (**class inventory**)
 - exponential growth of possible sentences (**combinatorial explosion**)
- Segmentation
 - no boundary markers
 - continuous flow of samples
- Variability
 - anatomy of vocal tract, speed, loudness, stress, dialect, mood (**speaker**)
 - noise, channel conditions, cocktail effect (**channel, environment**)
- Ambiguity
 - similar sounding words (**homophones**)
 - interface vs in her face (**word boundaries**)
 - He saw the Grand Canyon flying to New York (**semantics**)
 - Time flies like an arrow (**pragmatics**)

3.2 Speech units

- **phone**: a sound one produces, such as [p]
- **phoneme**: an abstract unit, such as /h/
- **morpheme**: the smallest part of speech with semantic meaning, for example un-break-able

When the meaning of an utterance can be changed by replacing one phone in the utterance with another phone, the replaced phone and the phone that replaces it are said to form a **minimal pair**. A phoneme may correspond to one phone or a set of phones. In the latter case, all phones of the set are called **allophones** of the phoneme. For example, /o/ and /u/ are phonemes in English because there are minimal pairs that differ only in these phonemes, such as "bone" and "boon". The English phoneme /p/ has two allophones, one with aspiration and one without it.

3.3 Problems with different Languages

More detailed LM-Handling in chapter 7.9!

- Words:
 - Indo-European: Segmentation via whitespace
 - Japanese, Chinese, Thai: No segmentation
- Spoken words: Words are not marked by boundaries. Units can have intonation phrases (like ,)
- Compounding (e.g. German): Donau-Schiff-Fahrts-Kapitäns-Mütze
 - *How to handle*: Decompose compound words into constituent parts and carry out pronunciation and language modelling on the decomposed parts
- Highly inflected languages(e.g. Turkish, Arabic, Slavic): *Inflection*: Modification of a word to express different grammatical categories (e.g. David-san)
 - *How to handle*: Specific components for modelling inflection

3.4 k -means clustering

Given an observation (x_1, \dots, x_n) , find $k \leq n$ clusters $S = \{S_1, \dots, S_k\}$, such that:

$$\operatorname{argmin}_S \sum_{i=1}^k \sum_{x_j \in S_i} \|x_j - \mu_i\|^2$$

where μ_i is the mean of points in S_i .

3.5 Comparing two utterances

Major problems:

- When does the utterance start?
- *Solution: Speaker has to push/ Always records and pattern recognition*
- Number of speech vectors is varying (speaker-dependent, on purpose, non purposely)
- *Solution: Dynamic Time Warping*
 - works like the dynamic editing distance algorithm (DP)
 - Cost function: vector-vector distance
 - *Pro:*
 - * only needs current and previous frame
 - * solves the different time-axis-alignment problem
 - *Cons:*
 - * Does not generalize
 - * Speaker dependent
 - * Need examples for each word for each speaker
 - * Computationally expensive for large vocabularies
 - * Continuous speech cannot be segmented except for human recognition because of **Co-Articulation, Hesitation and Hard decisions**.
 - * → Works only for small vocabularies, speaker dependent recognition and isolated words

4 Signal processing

Remember: Parametric representation depends on targeted application (speech coding, speech synthesis, speech recognition)

4.1 How can we extract features?

Our goal: Continuous signal \rightarrow Discrete representation

- sampling (x-Axis)
 - Aliasing possible if sampling rate is not at least 2 times maximal frequency (Nyquist or sampling theorem)
- quantization (y-Axis)
 - SNR (Signal to Noise Ratio) $\frac{\text{power}(\text{signal}[i])}{\text{power}(\text{noise}[i])}$
 - SNR low \rightarrow Recognition performance is low
 - speech signal is usually between 50dB and 60dB

4.2 Acoustic Features in sampled signal

- Amplitude
- Power (Integral of squared signal)
 - useful for detecting speech, syllabus, phrase boundaries
- Peak to Peak
- Root mean square
- Zero crossing
 - can help distinguishing some weak sounds from silence
- voiced/unvoiced

4.3 Frequency domain and Fouriertransformation

Why preform signal processing in the frequency domain?

- Human hearing is based on frequency analysis
- Use of frequency analysis often facilitates understanding
- Use of frequency analysis often simplifies signal processing

We can transform signals from time domain into frequency domain via **Fouriertransformation**

- *continuous*: $F(w) = \int f(t) e^{-iwt} dt$ - inverse: $f(t) = \frac{1}{2\pi} \int F(w) e^{-iwt} dw$
- *discrete*: $F(x[k]) = \sum x[k] e^{-iwt} dt$ - inverse: $x[k] = \frac{1}{2\pi} \int F(x[k]) e^{-iwt} dw$

Fouriertransformation have got some useful computational properties:

- Modelling a channel often can be accomplished by *convolution*
- A *convolution* in time domain is a *multiplication* in frequency domain and a *summation* in log-frequency domain
- Due to invertibility no information is lost

4.4 Short-Term Spectral Analysis

Problem: Frequency over an entire utterance does not help much because most acoustic events are not static, have durations in the range of 10 to 100ms and need more detailed analysis.

Idea: To better analyse our utterance we partition the utterance into short segments which may overlap each other. This is done by short-term spectral analysis.

The general idea of short-term spectral analysis is to

- multiply the speech signal with a window function
 - *Reason:* Conventional Fourier analysis does not capture time varying nature. Furthermore it smooths the resulting spectrum
 - *Some popular window functions:* Hamming (most used), Hanning, Rectangle, Gaussian
- apply the Discrete Time Fourier Transform (DTFT)
- use a filterbank (such as *Mel* or *Bark*)
 - *Reason:* Fourier coefficients reflects too much of the signals microstructure which contains a lot of redundancies and misleading information. We have to filter according to the human ear (works also with filterbanks!)

Notice: Instead of multiplying the speech signal with a window function and applying DTFT afterwards, you can also apply DTFT on the window function first and then multiply with the speech signal. The result will be the same.

4.5 Linear Predictive Coding

LPC is an alternative to Fourier analysis. When using LPC, we assume that a speech signal is produced by a buzzer (*glottis*) at the end of a tube (*vocal tract*), with occasional added hissing and popping sounds. Then, the intensity and frequency can be estimated with linear prediction:

$$\tilde{s}(n) = \sum_{i=1}^p \alpha_i s(n-i) \quad (1)$$

Some sounds, for example nasal sounds, cannot be described by this model. This means that there will be an error, called the residue, which is stored and used to model other sounds:

$$e(n) = s(n) - \tilde{s}(n) \quad (2)$$

The goal of the prediction is to minimize the squared error $\sum e(n)^2$. This error can be further transformed into cepstral coefficients using the Z-transform.

4.6 Cepstra

The speech signal $s(n)$ can be described by a convolution of stimulating signal $u(n)$ (caused by the glottis) and an impulse response $h(n)$ (caused by the vocal tract and the lips):

$$s(n) = u(n) * h(n) \quad (3)$$

In speech analysis, we are particularly interested in discovering the formant structure. This means that we have to "get rid" of the stimulating signal:

$$\mathcal{F}(s) = \mathcal{F}(u) \cdot \mathcal{F}(h) \quad (4)$$

$$\mathcal{F}^{-1}(\log(\mathcal{F}(s))) = \mathcal{F}^{-1}(\log(\mathcal{F}(u))) + \mathcal{F}^{-1}(\log(\mathcal{F}(h))) \quad (5)$$

Note that after the inverse Fourier transform, we are not in the time domain, but in the cepstral domain (also called quefrency). The lower cepstral coefficients describe the macro structure of the signal whereas the higher cepstral coefficients describe the micro structure of the signal. Thus, the higher cepstral coefficients are often discarded.

In order to separate the signal's components, we have to apply a *lifter* (setting some of the coefficients to 0) to filter out the information we need. This is done by removing cepstral summits with a lowpass lifter.

A cepstrum in general is:

- efficient and robust coding of speech information
- information about rate of change in the different spectrum bands
- useful for separating signals (like the source and filter in the Source-Filter-Model)

4.6.1 Mel-frequency cepstrum (MFC)

MFCCs are coefficients that collectively make up an MFC. MFCCs are commonly derived as follows:

1. segment incoming waveforms into frames (10ms)
2. apply the Fourier transform to a signal to compute frequency response
3. group the frequency response into 25-40 buckets by applying a Mel filterbank (triangular weighting function)
4. take the log of the powers at each Mel "bucket"
5. take the discrete cosine transform (DCT) of the list of mel log powers
6. the MFCCs are the amplitudes of the resulting spectrum

Usually, we take the first coefficient (the power of the signal) and 12 more coefficients, amounting to a total of 13 numbers.

4.7 Feature vectors

During further processing, we often assume that the speech signal at a given point of time is independent from the signal before and after that point of time. This is a poor assumption. So, if we have a feature vector x (if we're using MFCCs, we have 13 coefficients), we also store the first and second derivatives of x , namely Δx and $\Delta\Delta x$ (if we're using MFCCs, we now have 39 numbers - a huge reduction, compared to 8000 numbers in the original speech signal).

5 Hidden Markov Models

5.1 Why do we use HMMs?

- mathematical foundation
- works with speech units shorter than words
 - subword unit occurs often
 - training easier
 - less data required
- Recognize speech from any speaker
- Recognize continuous speech
- Recognize untrained words due to subwords compositions

5.2 The three great problems

5.2.1 Evaluation

Problem: Given an HMM λ and an observation $O = (x_1, \dots, x_T)$, compute the probability $P(x_1, \dots, x_T | \lambda)$.

A weaker problem: Given an HMM λ an observation $O = (x_1, \dots, x_T)$ and a state sequence $(s_{q_1}, \dots, s_{q_T})$, compute the probability $P(x_1, \dots, x_T | \lambda, (s_{q_1}, \dots, s_{q_T}))$.

Solution:

$$\pi(s_{q_1}) b_{q_1}(x_1) \prod_{k=1}^{T-1} a_{q_k q_{k+1}} b_{q_{k+1}}(x_{k+1}) \quad (6)$$

This solution is of course not applicable to the evaluation problem because the state sequence is unknown.

Solution: the Forward algorithm. Define:

$$\alpha_1(j) = \pi(s_j) \cdot b_j(x_1) \quad (7)$$

$$\alpha_t(j) = \sum_i a_{ij} \cdot \alpha_{t-1}(i) \quad (8)$$

5.2.2 Decoding

Problem: Given an HMM λ , and an observation $O = (x_1, \dots, x_T)$, find the most likely state sequence s_{q_1}, \dots, s_{q_T} :

$$\begin{aligned} \operatorname{argmax}_{(q_1, \dots, q_T)} P(q_1, \dots, q_T \mid x_1, \dots, x_T, \lambda) &= \operatorname{argmax}_{(q_1, \dots, q_T)} \frac{P(q_1, \dots, q_T, x_1, \dots, x_T \mid \lambda)}{P(x_1, \dots, x_T \mid \lambda)} \\ &= \operatorname{argmax}_{(q_1, \dots, q_T)} P(q_1, \dots, q_T, x_1, \dots, x_T \mid \lambda) \end{aligned}$$

Solution: the Viterbi algorithm. Define:

$$z_1(j) = \pi(j) \cdot b_j(x_1) \quad (9)$$

$$z_t(j) = \max_i (a_{ij} \cdot z_{t-1}(i)) \cdot b_j(x_t) \quad (10)$$

Both the Forward and the Viterbi algorithm often multiply many small numbers. This can lead to floating point underflow errors. As a solution, one can either scale the factor by a constant or use logarithms to turn the multiplications into additions.

5.2.3 The Backward algorithm

Obviously:

$$P(x_1, \dots, x_T \mid \lambda) = \sum_i P(x_1, \dots, x_T, q_t = i \mid \lambda) \quad (11)$$

Define:

$$\beta_t(i) = P(x_{t+1}, \dots, x_T, q_t = i \mid \lambda) \quad (12)$$

$\beta_t(i)$ is the probability of observing the partial observation x_{t+1}, \dots, x_T and being in state s_i . Then:

1. Initialization: $\beta_T(i) = N^{-1}$
2. Induction: $\beta_t(i) = \sum_j a_{ij} \cdot b_j(x_{t+1}) \cdot \beta_{t+1}(j)$
3. Termination: $P(x_1, \dots, x_T \mid \lambda) = \sum_j \beta_T(j)$ (shouldn't this be $\sum_j \beta_1(j)$?)

5.2.4 Learning / optimization

Goal: Given an observation O and an HMM λ , find another HMM λ' such that $P(O \mid \lambda') > P(O \mid \lambda)$.

Solution: We use the Baum-Welch algorithm, which is an EM algorithm (expectation-maximization). The Baum-Welch algorithm uses a forward-backward procedure.

First, define $\gamma_t(j)$, which is the probability that the system λ is in state s_j at time t when producing the observation $X = (x_1, \dots, x_T)$:

$$\gamma_t(j) = \frac{P(q_t = j \mid X, \lambda)}{P(X \mid \lambda)} = \frac{\alpha_t(j) \cdot \beta_t(j)}{\sum_i \alpha_t(i) \cdot \beta_t(i)} \quad (13)$$

To optimize a_{ij} , we need the probability $\xi_t(i, j)$ of being in state i at time t , transition from state i to j and be in state j at time $t + 1$.

$$\begin{aligned} P(q_t = 1, q_{t+1} = j \mid X, \lambda) &= \frac{P(q_t = 1, q_{t+1} = j, X \mid \lambda)}{P(X \mid \lambda)} \\ &= \frac{\alpha_t(i) \cdot a_{ij} \cdot b_j(x_{t+1}) \cdot \beta_{t+1}(j)}{P(X \mid \lambda)} \\ &= \xi_t(i, j) \end{aligned}$$

As a reminder (this follows from the rules of conditional probabilities):

$$P(A \mid B, C) = \frac{P(A, B \mid C)}{P(B \mid C)} \quad (14)$$

Define:

- $\sum_t \gamma_t(i)$ expected # of times i is visited
- $\sum_t \xi_t(i, j)$ expected # of transitions from i to j
- $\pi(i)$ expected frequency in i at time $(t = 1) = \gamma_1(i)$
- a_{ij} expected # transitions from i to j **div** expected # of transitions from i
- $b_i(v_k)$ expected # in i and observing k **div** expected # in state i

Now, we can define the new HMM $\lambda' = (S, \pi', A', B', V)$:

- $\pi'(i) = \gamma_1(i)$
- $a'_{ij} = \sum_t \xi_t(i, j) / \sum_t \gamma_t(i)$
- $b'_i(v_k) = \sum_t \gamma_t(i) \cdot \delta(x_t, v_k) / \sum_t \gamma_t(i)$ where $\delta(x, y) = (x \doteq y)$

Note that running the Viterbi algorithm leads to somewhat similar results. Training an HMM with Viterbi is faster, but when only little data is available, Baum-Welch performs better.

5.3 HMM state tying

When using Gaussian mixtures to model HMM output or transition probabilities, mixing more Gaussians doesn't necessarily lead to better results. This is due to an effect somewhat similar to overtraining.

- we use a simple three-state bakis model (similar to left-right model) for each phone
- the $b_i(x)$ are often Gaussian mixtures
- discrete HMMs are rarely used

5.3.1 Fully-continuous HMMs

- states that correspond to the same acoustic phenomenon share the same "acoustic model"
- for example, all /g/ phonemes may share the same model for the g - b HMM state
- the training data can be exploited better
- emission parameters can be estimated more robustly
- don't evaluate b for every state - save computation time!

See also figure 4.

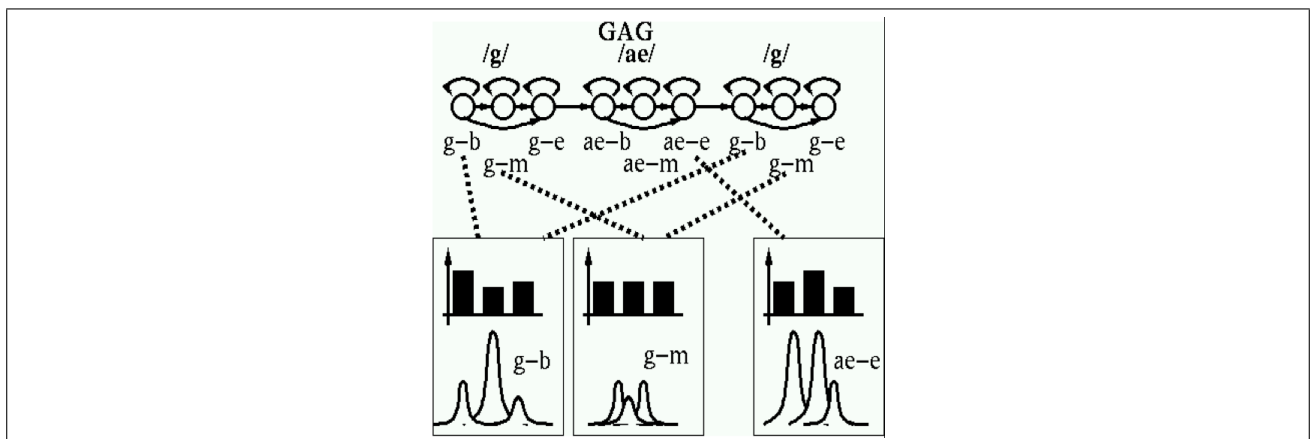


Figure 4: A fully-continuous HMM

5.3.2 Semi-continuous HMMs

- only one codebook of Gaussians in the system
- every acoustic model has its own set of *mixture weights*

See also figure 5.

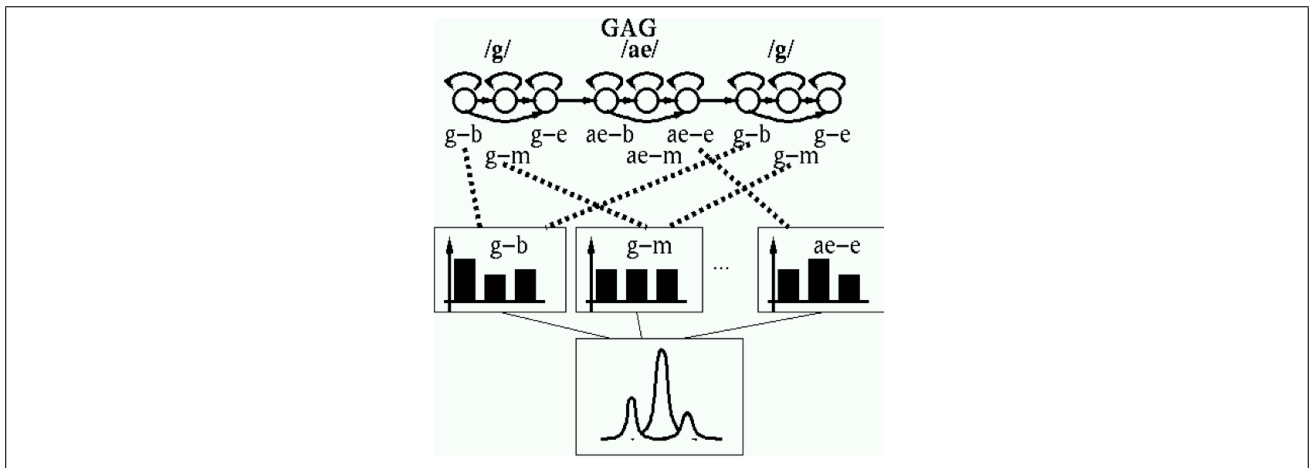


Figure 5: A semi-continuous HMM

5.3.3 Phonetically-tied semi-continuous HMMs

One codebook per phone that is shared across polyphones. For example, $/g/$ might have one codebook and $/ae/$ another.

5.4 HMM training

A typical HMM training session might look like this:

1. Initialization:
 - initialize all parameters randomly
 - or use pre-labeled data and compute initial parameters with e.g. **k-means**
2. Iterative optimization / training:
 - for all training utterances, run *Baum-Welch* and update the HMM accordingly
 - continue for a defined number of iterations or until there are no more improvements
3. Evaluation by recording an utterance and testing the HMM.

5.5 HMM parameter initialization

Use labeled data:

- automatic labeling
 - use an existing recognizer
 - probably suboptimal
- hand-made labels
 - manually assign HMM states to speech frames
 - probably time consuming
- use partial labels

6 Acoustic modeling

6.1 Discrete HMMs in continuous space

In ASR terminology, an HMM is *discrete* if the output space is partitioned into discrete, non-overlapping regions. The most common way to find such regions is to use k -means clustering. Then, the HMM states can emit discrete indices identifying one particular region.

A simple example of this is the regular partitioning of the F1-F2 formant space.

6.2 Source coding

- Scalar quantization:
 - for a scalar observation x , compute a quantization $y = Q(x)$
 - can be simple, such as rounding (e.g. $1.3 \rightarrow 1.0$ or $1.3948 \rightarrow 1.40$)
- Vector quantization:
 - divide large set of vectors into groups using e.g. k -means clustering
 - identify vectors by their group index
 - lossy data compression

6.3 Continuous HMMs

In ASR terminology, an HMM is *continuous* when the output space is continuous and so are the emission probabilities b_i . Often, Gaussians $b_i = \mathcal{N}(\mu_i, \Sigma_i)$ are used (μ_i is a k -dimensional vector, Σ_i is a $k \times k$ covariance matrix).

It is also possible to use a mixture of Gaussians:

$$b_i = \sum_{j=1}^M c_{ij} \mathcal{N}(\mu_{ij}, \Sigma_{ij}) \quad (\text{or rather: } b_i = \sum_{j=1}^M c_{ij} \mathcal{N}(\mu_j, \Sigma_j) \quad ?)$$

where c_{ij} are the mixture weights.

If time is not an issue, we can use a continuous model. Otherwise, we will have to save computation time by altering our model.

6.3.1 Conversion to semi-continuous model

We “pull out” each Gaussian and collect them in a codebook. There will be a maximum of $M \cdot N$ Gaussians, where N is the number of states.

6.3.2 Conversion to shared semi-continuous model

We extend the mixture weights to include Gaussians that were initially assigned to other states. This means that many mixture weights which previously were 0 are now greater than 0. This results in even more parameters.

Problem: If we create one codebook for each emission probability, we end up with *a lot of* parameters. We may not have enough data to train our model.

Solution: Parameter tying. The most flexible way of parameter tying is to use an arbitrary number of Gaussian codebooks and an arbitrary number of mixture weight sets.

6.3.3 Conversion to tied semi-continuous model

Reduce the size of the codebook to a smaller size $L < M \cdot N$. Share Gaussians that belong to similar units (*tie units*). This “ties” the codebook elements (and states) together. The number of models to train is now manageable.

6.4 Parameter tying

There are two main approaches to parameter tying:

- **Knowledge driven:** Have an expert sit down and decide which parameters to tie.
- **Data driven:**
 - let n be the number of training vectors for a codebook, set the codebook size to $n \cdot f$ where f is an empirically found factor
 - run k -means with different values for k , choose the k with the best result
 - start with one codebook, train, once we have enough data, split
 - start with huge codebook, remove Gaussians with too few training counts, retrain
 - merge and split

The following parameters can be tied:

- sets of Gaussians
- within a mixture: have different Gaussians use the same covariance matrix
- transitions: different HMM states share the same transition probabilities

6.5 Lexicon

Problem: How do we connect our HMMs to words?

Solution: With a pronunciation dictionary.

In the simplest architecture, a word is represented as a sequence of phonemes and each HMM state represents one phoneme. In the lexicon, each word is bound to a phoneme sequence.

6.5.1 Pronunciation variants

One word can be pronounced differently due to:

- coarticulation effects
- dialects (e.g. *thing* vs. *thang*)
- various correct pronunciations (e.g. $a = AE$ or $a = EY$)
- correct pronunciation differs from most commonly used ones (e.g. *February* vs. *Febury*)

To solve this problem, we add variants to the pronunciation dictionary. When building an HMM for a word, we build multiple, alternative state sequences.

6.6 Context dependent acoustic modeling

Consider the pronunciation of *train* and *tell*. Common lexicon entries will be $T-R-EY-N$ and $T-EH-L$, but the actual pronunciation often sounds more like $CH-R-EY-N$ and $T-HH-EH-L$. The phoneme T sounds different depending on the phoneme following it.

At first, one might try to alter the lexicon entries, i.e. $CH-R-UW$ instead of $T-R-UW$, but the CH in *true* sounds different from the one in *church*.

So, we introduce new acoustic units, such that lexicon entries might look like $T(R)-R-EY-N$ and $T(vowel)-EH-L$. This means that we use context-dependent models of the phoneme T .

6.6.1 Crossword context modeling

It is not a good idea to assume that the left context of a word is silence. Because of this, we extend context modeling across word boundaries while building a sentence HMM.

Go to AM1 26/30 for more on this subject.

6.6.2 Position dependent modeling

The same phoneme (with the same context) is pronounced differently depending on its position in the word. As an example, consider the *D*-phoneme in *Idaho* and *my dad*. To solve this problem, we use position dependent modeling, where *wb* indicates a word boundary. In *Idaho*, this will become $D(AY, AE)$, whereas in *my dad*, it will become $D(wb)(AY, AE)$.

6.6.3 Parameter tying

With so many different states, even with lots of training data, the HMM cannot be trained enough. As a solution, we use parameter tying to combine multiple contexts into the same context class:

- Knowledge based:
 - let a linguist define classes C_i of phonemes (e.g. vowels, consonants, fricatives, etc.)
 - then, define generalized triphones for a monophone M : $M(C_l, C_r)$
- Data driven:
 - use some algorithm with an optimization criterion to decide which contexts should be tied
- Backoff procedure:
 - if we have enough training data for triphones, use triphones $P_m(P_l, P_r)$
 - otherwise, if we have enough training data for left or right context biphones ($P_m(P_l)$ or $P_m(P_r)$), use them
 - otherwise, as a fallback, use an indepent model P_m

6.7 Clustering

There are two different approaches to clustering:

- **bottom-up clustering** (agglomerative): start with one class for each phone, then combine “similar enough” classes until satisfied
- **top-down clustering** (divisive): start with one class for all phones, divide a class if it seems like a “good idea”, continue until satisfied

Both approaches result in a *clustering tree*.

Problem: How do we know whether it’s a good idea to separate a class?

Solution: Use a *distance measure*.

6.7.1 Continuous parametric models

- $d(f, g) = \int \min(f(x), g(x))$
- Kullback-Leibler divergence: $d_{KL}(f, g) = \int f(x) \ln \frac{f(x)}{g(x)}$
- entropy distance: $d(f, g) = H(f + g) - \frac{1}{2}H(f) - \frac{1}{2}H(g)$

6.7.2 Discrete models

Semi-continuous and discrete HMMs are represented by *discrete* distributions.

- $H(f) = \sum_i f[i] \log_2(\frac{1}{f[i]})$
- $d(f, g) = H(f + g) - \frac{1}{2}H(f) - \frac{1}{2}H(g)$
- obviously: $f = g \Rightarrow H(f) = H(g) = H(f + g) \Rightarrow d(f, g) = 0$

As an example, consider $f = \{\frac{1}{2}, \frac{1}{2}\}$, $g = \{\frac{3}{4}, \frac{1}{4}\}$. Then, $f + g = \{\frac{5}{8}, \frac{3}{8}\}$. It follows that $H(f) = 1$, $H(g) \approx 0.81$, $H(f + g) \approx 0.95$ and finally $d(f, g) \approx 0.05$.

Weighted discrete entropy distance

Consider three models M_1, M_f, M_{f+} , where M_1 is a model trained with many samples, M_f is a model trained with few samples and M_{f+} is a model that is trained with few samples, but more than M_f was trained with. Combining M_1 with M_f should have a smaller impact on the distance compared to combining M_1 with M_{f+} . Thus, we weigh the model entropy by the number of training samples:

$$d(f, g) = (n_f + n_g)H(f + g) - n_fH(f) - n_gH(g)$$

6.7.3 Kai-Fu Lee

1. train semicontinuous models for all three states of each triphone, e.g. triphone $A(AE, K)$ - b , $A(AE, K)$ - m , $A(AE, K)$ - e .
2. initialize a context class for every triphone (a context class is defined by three distributions, e.g. T_{17-b} , T_{17-m} , T_{17-e})
3. compute all distances between different context classes of same phone:

$$d(C_i, C_j) = E(C_i-b, C_j-b) + E(C_i-m, C_j-m) + E(C_i-e, C_j-e)$$

4. replace the two classes with the smallest class by their combination
5. try to improve distance by moving any element from any class into any other class

Note: This algorithm is completely data driven. Step 5 is expensive but important.

Kai-Fu Lee's algorithm produces *generalized triphones*. For example, $A(H, U)$ and $A(B, P)$ might be combined into $A\mathfrak{H}$, and $A(L, N)$ and $A(M, M)$ might be combined into $A\mathfrak{L}$.

6.7.4 Decision trees

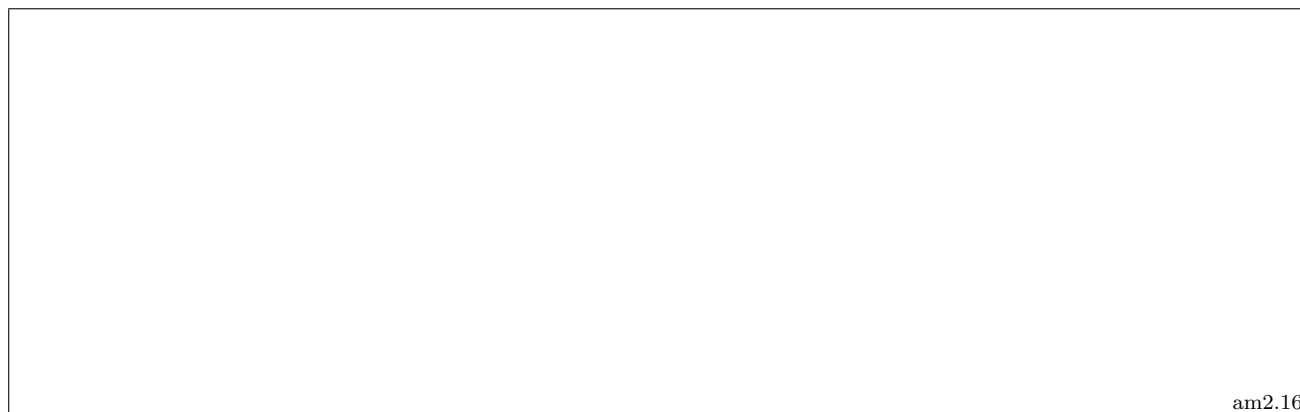
After clustering, we might end up with two classes:

$$C_1 = \{P_1(P_2, P_3), P_1(P_4, P_5)\} \quad C_2 = \{P_1(P_6, P_7), P_1(P_8, P_9)\}$$

Problem: If we want to recognize the word sequence P_3, P_1, P_7 , do we use C_1 or C_2 to model $P_1(P_3, P_7)$?

Solution: Build a decision tree.

1. initialize one cluster containing all contexts
2. for all clusters and questions: compute distance of subclusters
3. perform the split that gets the largest separation



am2.16

Figure 6: Growing the decision tree.

4. continue until satisfied

See figure 6 for an example.

Note: It doesn't matter which questions we ask, as long as our question set allows for a variable enough separation.

7 Language modeling

Remember the fundamental problem of speech recognition:

$$\operatorname{argmax}_W P(X \mid W) \cdot P(W)$$

The purpose of the language model is to compute the probability $P(W)$ of a word sequence $W = (w_1, \dots, w_n)$.

A LM should ...

- improve speech recognizer by adding another independent information source.
- disambiguate homophones, e.g.: “*I OWE YOU TOO*” vs “*EYE O U TWO*”.
- reduce search space since not all word sequences are possible.
- Analysis to *understand* utterance, disambiguate homonyms (bank: money vs river).

7.1 Deterministic vs. Stochastic Language Models

In formal language theory (see Chomsky hierarchy) $P(W)$ is either 0 or 1. One way to model a deterministic language is with grammars. This deterministic approach is unsuitable for ASR because spoken language can not be completely covered by grammar and is often ungrammatical, anyway. Thus, stochastic LMs, e.g. *N-grams*, are employed.

7.2 N-grams

The difficulty of recognizing a word sequence is correlated to the *branching degree*. For example, the branching degree of “*I apologize for being late, I’m very —!*” is smaller than the one of “*Hello, my name is —.*”.

The choice of the next word depends on the entire history. Since there are too many histories to test them all, we have to find equivalence classes \mathcal{C} so that:

$$P(w \mid \text{history}) = P(w \mid \mathcal{C}(\text{history}))$$

We can use different equivalence classes using information about:

- grammatical content
- Part-of-Speech (POS) of previous word(s)
- semantic meaning of previous word(s)
- context similarity
- some kind of automatic clustering

A simple, yet common choice for the equivalence classes are *N-grams*:

$$P'(w_k \mid w_1, \dots, w_{k-1}) = P(w_k \mid w_{k-(n-1)}, \dots, w_{k-1})$$

Usually, $P(w \mid \text{history})$ is estimated by counting relevant occurrences:

$$P(w \mid \text{history}) = \frac{\text{Count}(\text{history} \cdot w)}{\text{Count}(\text{history})}$$

N-grams are helpful because in HMM-recognizers the language model is responsible for the computation of the transition probabilities between acoustic models on a word level. They can be computed on the fly and may depend on previous words.

7.2.1 Bigrams vs. Trigrams

While Bigrams can be easily worked into an HMM-recognizer (you only need to take w_t into consideration for the probability of w_{t+1}), Trigrams often require a larger history, especially if a word can have many predecessors.

Possible solutions are to use a *time asynchronous search* or use so-called *poor man's trigrams* (e.g. consider only the predecessor's best predecessor).

Additionally the test data coverage of Trigrams is smaller than with Bigrams and the estimation of $P(w_k \mid w_{k-2}, w_{k-1})$ is more difficult than $P(w_k \mid w_{k-1})$.

7.3 Perplexity

To compare to different language models, we need an independent measure. Since the performance of a recognizer not only depends on the Language model, but also the acoustic model and the combination mechanism of the acoustic model with the language model, we use *perplexity*.

A language model is “good” if it has a low *perplexity*. The perplexity PP of a language model P on the test set $W = (w_1, \dots, w_n)$ is:

$$PP = P(w_1, \dots, w_n)^{-1/n} (= 2^{H(W)})$$

7.4 Smoothing

The key problem of n -grams is the *data sparseness* problem: many n -grams will not occur in the training data, and many others will only occur very few times, resulting in inaccurate calculations. For example, in a several million word collection of English text, 50% of trigrams occur only once, and 80% of trigrams occur less than 5 times. As a start, we assign all strings a non-zero probability, in an effort to prevent ASR errors.

The idea of smoothing is to “free probability mass” from well-trained events and “redistribute” it to unseen events. This is somewhat similar to the idea of taxation.

7.4.1 “Add-one” smoothing

Add 1 to all bigrams:

$$\begin{aligned} P_{one}(w_i \mid w_{i-1}) &= \frac{C(w_{i-1}w_i) + 1}{C(w_{i-1}) + n_v} \\ &= (1 - \lambda(w_{i-1})) \frac{C(w_{i-1}w_i)}{C(w_{i-1})} + \lambda(w_{i-1}) \frac{1}{n_v} \end{aligned}$$

where $\lambda(w_{i-1}) = n_v / (n_v + C(w_{i-1}))$ and $n_v = |\{w : C(w) > 0\} \cup \{\text{UNK}\}|$. This means that we're mixing the estimated distribution with the uniform distribution. However, when n_v gets large, the estimation approximates the uniform distribution.

7.4.2 Backoff smoothing

Whenever possible, use higher-order models. Otherwise, for example with unseen events, backoff to lower-order models.

$$P_{bo}(w_i \mid w_{i-1}) = \begin{cases} \tilde{P}(w_i \mid w_{i-1}) & \text{if } C(w_{i-1}w_i) > 0 \\ \lambda(w_{i-1}) \cdot P_{bo}(w_i) & \text{otherwise} \end{cases}$$

Absolute discounting

Subtract a fixed D from each n -gram count:

$$P_{abs}(w_i \mid w_{i-1}) = \begin{cases} \frac{\max(C(w_{i-1}w_i) - D, 0)}{C(w_{i-1})} & \text{if } C(w_{i-1}w_i) > 0 \\ \lambda(w_{i-1}) \cdot P_{abs}(w_i) & \text{otherwise} \end{cases}$$

Note: This formula is probably incorrect. I think what we're trying to do is something like this:

$$P_{abs}(w_i \mid w_{i-1}) = \begin{cases} \frac{C(w_{i-1}w_i) - D}{C(w_{i-1})} & \text{if } C(w_{i-1}w_i) > D \\ \lambda(w_{i-1}) \cdot P_{abs}(w_i) & \text{otherwise} \end{cases}$$

Or possibly even a weighted approach.

Good-Turing estimation

The idea is to reallocate the probability mass of n -grams that occur $r + 1$ times in the training data to the n -grams that occur r times. In particular, reallocate the probability mass of n -grams that were seen once to the n -grams that were never seen.

For each count r , we compute an adjusted count r^* :

$$r^* = (r + 1) \frac{n_{r+1}}{n_r}$$

where n_r is the number of n -grams seen exactly r times. Then, we compute the discount d_r :

$$d_r = \frac{r^*}{r}$$

Katz smoothing

Use k as a threshold to apply discounting. Use Good-Turing for discounting.

$$P_{katz}(w_i \mid w_{i-1}) = \begin{cases} \frac{C(w_{i-1}w_i)}{C(w_{i-1})} & \text{if } r > k \\ d_r \cdot \frac{C(w_{i-1}w_i)}{C(w_{i-1})} & \text{if } k \geq r > 0 \\ \alpha(w_{i-1}) \cdot P_{katz}(w_i) & \text{if } r = 0 \end{cases}$$

Kneser-Ney smoothing

As an example, let's suppose that "San Francisco" is common, but "Francisco" occurs only after "San". Then "Francisco" will get a high unigram probability and so absolute discounting will give a high probability to "Francisco" appearing after novel bigram histories (for example, "Los Francisco"). Since "Francisco" only ever occurs after "San", it is better to give it a low unigram probability. All in all, we use absolute discounting with a special backoff function. Define:

$$\begin{aligned} \#(\bullet w_i) &= |\{w_{i-1} : C(w_{i-1}w_i) > 0\}| \\ \#(\bullet \bullet) &= \sum_{w_i} \#(\bullet w_i) \end{aligned}$$

Then, define:

$$P_{KN}(w_i) = \frac{\#(\bullet w_i)}{\#(\bullet \bullet)}$$

7.4.3 Linear interpolation

Define:

$$P_I(w_i \mid w_{i-2}w_{i-1}) = \lambda \cdot \tilde{P}(w_i \mid w_{i-2}w_{i-1}) + (1 - \lambda) \cdot P_I(w_i \mid w_{i-1})$$

Key difference: in the case of non-zero counts, interpolated models still use information from lower-order models while backoff models do not.

(Deleted Interpolation Smoothing)

7.5 n -gram classes

Another way to handle data sparsity is to group words into classes of similar semantic or grammatical behavior. For simplicity, let's assume that one word w_i can be assigned to only one class c_i . Then:

$$P(w_i \mid c_{i-(n-1)}, \dots, c_{i-1}) = P(w_i \mid c_i) \cdot P(c_i \mid c_{i-(n-1)}, \dots, c_{i-1})$$

n -gram classes have not improved recognition accuracy significantly.

7.6 Different kinds of language models

7.6.1 Cache language models

- **Idea:** Use a static and a dynamic component.
- **Static component:** $P_S(w_k \mid w_{k-(n-1)}, \dots, w_{k-1})$, i.e. the usual trained language model.
- **Dynamic component:** Complete n -gram language model constructed from the text dictated so far, e.g.:

$$P_C(w_k \mid w_{k-(n-1)}, \dots, w_{k-1}) = \frac{1}{2} \cdot f(w_k) + \frac{1}{4} \cdot f(w_k \mid w_{k-1}) + \frac{1}{4} \cdot f(w_k \mid w_{k-2}w_{k-1})$$

- **Total language model:** $P_T = \lambda_C \cdot P_C + (1 - \lambda_C) \cdot P_S$
- **Variation:** Compute P_C on last l words of cache, vary λ_C with cache size.

7.6.2 Trigger language models

- **Observation:** Often, when a content-carrying word occurs in a text, it is likely to occur again some time later.
- **Idea:** Use a standard interpolated n -gram model, but constantly update the unigram probabilities $P(w_k)$ for some words w_k .
- **Trigger:** For each word w define a trigger list $L(w)$ of words that are likely to occur some time later, e.g. $L(\text{money}) = \{\text{bank, savings, dollars, ...}\}$
- Then, for each word $v \in L(w)$, increase $P(v)$ by some value.

7.6.3 Multilevel language models

- **Idea:** Use language models LM_W for word sequences, LM_P for phrases and LM_S for sentences trained as regular n -gram grammars over their corresponding segments of speech.
- **Combination:** Let the total language model LM_T be a combination of the three models:

$$LM_T = \lambda \cdot LM_W + \lambda \cdot LM_P + \lambda \cdot LM_S$$

7.6.4 Interleaved language models

Sentences can often be split into two separate word sequences, one containing *content words* and the other containing *non-content words*:

sentence:	The	president	of the	software company	said	that they will	introduce	...
non-content:	The		of the			that they will		...
content:		president		software company	said		introduce	...

Based on this, we define:

$$P(w \mid \text{history}) = \begin{cases} P_C(w \mid C_1(\text{history})) & \text{if } w \text{ is a content word} \\ P_{NC}(w \mid C_2(\text{history})) & \text{if } w \text{ is a non-content word} \end{cases}$$

7.6.5 Morpheme-based language models

In heavily inflected or agglutinating languages, the word stem is succeeded or preceded by gender-, tempus-, numerus- and case-identifying elements. The probabilities of full words cannot be estimated robustly. However, a robust estimation can be given for the word stem. So, we don't compute the language model on sequences of words but on sequences of morphemes:

$$P(w_1, \dots) = \tilde{P}(w_{11}, w_{12}, \dots, w_{1n_1}, \dots)$$

7.6.6 Context-free grammar language models

Context-free grammars can be used instead of n -grams. The rules of the grammar can be written by an expert without the need for lots of training data. CFGs are powerful enough to represent large parts of natural languages and constrained enough to allow for efficient search space reduction. Furthermore, semantic analysis of speech is possible with CFG-derived finite state automata (\rightarrow figure 7).

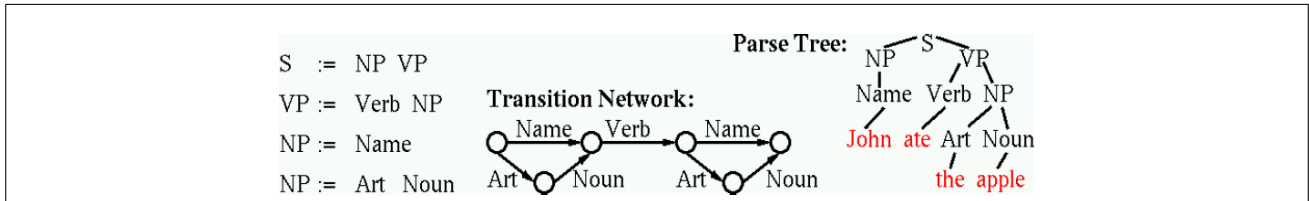


Figure 7: Context-free grammar language models.

7.6.7 Tree-based language models

n -grams have a very limited context and training the parameters is almost impossible for large values of n . We can use a CART-based technique (classification and regression trees), where every tree node asks about k previous words ($k \approx 20$). Every leaf node is a unigram probability distribution on our vocabulary. This is somewhat similar to clustering acoustic context-dependent models (\rightarrow figure 8).

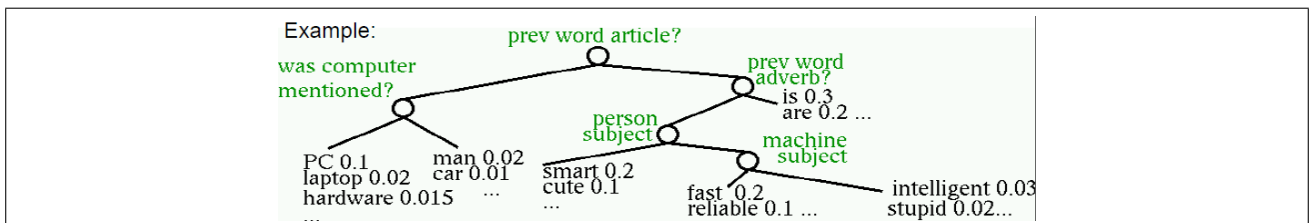


Figure 8: Tree-based language models.

7.6.8 HMMs for language modeling

Often, speech is carried out in phases, such as *greeting* \rightarrow *smalltalk* \rightarrow *good-bye*. Based on this, we can build an HMM, where every state emits language models or mixture weight distributions (\rightarrow figure 9).



Figure 9: HMMs for language modeling.

7.7 Vocabulary selection

A smaller vocabulary eliminates potentially confusing candidates and improves the word error rate. A larger vocabulary has a lower out-of-vocabulary rate, but the word error rate increases. The goal is to balance the WER and OOV rates. We do this by using a corpus of text to determine appropriate vocabularies. For example, we can pick data from a specific domain and use the most frequent words.

7.8 n -gram pruning

The size of higher-order n -gram models is often too large for practical applications. Therefore we reduce the size of N -gram models by **Pruning**. We minimize the relative entropy between original and pruned model and the performance loss. Experience shows that trigram models can be pruned by more than 25% without losing recognition accuracy.

7.9 Problems with spontaneous speech

- silence between words (“hello world” vs. “hello world”)
- hesitations (“ah”, “uhm”, ...)
- filler words and phrases (“I mean”, “you know”, ...)
- false starts, aborts, etc. (“let’s meet sun- no saturday”)
- non-verbal sounds (breathing, lip smacks, ...)

Possible solutions are:

- just ignore it (*duh!*)
- increase context width (e.g. use 4-grams instead of trigrams)
- skip spontaneous effects in the history

7.10 Unknown words

Note: Here, “unknown word” refers to a word not seen in the training data, but occurring in the decoder vocabulary and dictionary - as opposed to not being known at all.

In the acoustic model, we can handle words that have not been seen in the training set by defining their pronunciation(s) in the pronunciation lexicon. However, in most speech recognition tasks, we cannot train all words that might have to be recognized some time. So, how do we incorporate unknown words into the language model? Approaches to this problem include:

- rewrite all words in the training text that only occur once with UNK and treat every unknown word in the recognition run as UNK
- use a class-based language model and assign a class to every unknown word
- add new word into vocabulary and use cache language model to improve the parameters for subsequent recognitions

7.11 OOV words

OOV words are words that are not known at all. These words *must* lead to recognition errors. Ideally, every OOV word causes only one substitution, which would imply $WER = OOV$. Usually, there are follow-up errors: every OOV word causes between 1.5 and 2 errors on average.

7.12 Problems with different languages

- **Problem:** Highly inflecting languages pose more of a problem to language recognition. **Solution:** Use morpheme-based language models.
- **Problem:** Some languages, e.g. German allow arbitrary compounding of nouns. **Solution:** Decompose compound nouns.
- **Problem:** Some languages, e.g. Chinese have a different or no specified notion of words. **Solution:** Consider syllable-based language models.

7.13 Problems with Recognition Errors

Strong language model constraints can lead to follow-up errors: If a word is misrecognized, strong constraints can lead to the next word(s) to be misrecognized, too. An approach to solve this issue may be to change the effect of the language model on the recognition process depending on the current confidence.

7.14 Language model adaptation

Natural language is highly variable, depending on time (medieval German vs. current German), domain (fishing vs. politics), context (scientific paper vs. chatting with friends), etc.

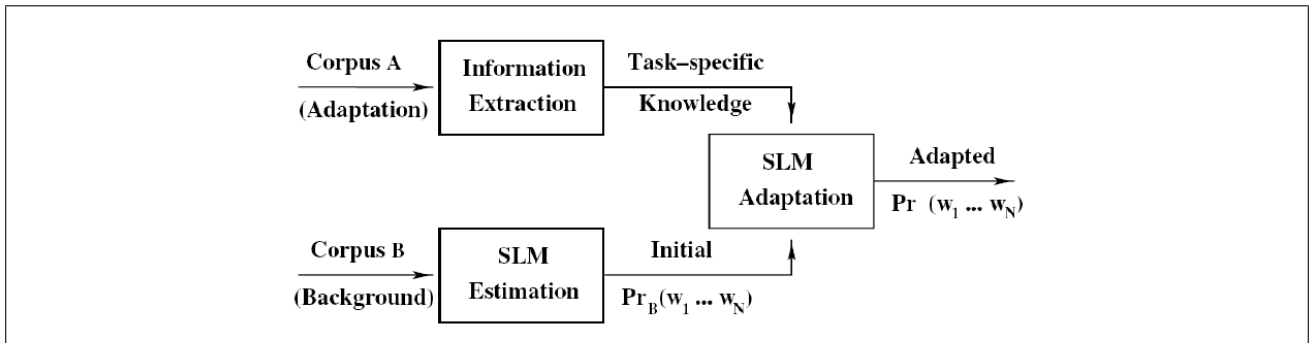


Figure 10: Language model adaptation framework.

Generally, we have a large background corpus \mathcal{B} and a small adaption corpus \mathcal{A} depending on the current task. The goal is to compute a robust estimate of the language model probability $Pr(w_1, \dots, w_N) = \prod_{q=1}^N Pr(w_q | h_q)$, where $h_q = w_{q-n+1}, \dots, w_{q-1}$. We can try to solve this problem by using language model adaptation (see also figure 10). There are three techniques of doing this:

- Model Interpolation
- Constraint Specification
- Meta-Information Extraction

7.14.1 Retrieval of adaptation data

We have to retrieve our adaption data from somewhere:

- **Grammar-based system:** Generate text using the grammar. (Random rules or use small corpus to learn rule weights)

- **Use recognition output:** Accumulate corpus \mathcal{A} during the recognition process, Take N-best or Confusion Networks as adaptation material; each hypothesis contributes to corpus according to its posterior likelihood
- **Search & Retrieve:** Use a small corpus \mathcal{A} as a seed for retrieval techniques (online databases and WWW).
- Augment data at the N -gram count level

7.14.2 Model interpolation

Idea: Derive task-specific (*dynamic*) SLM from Corpus \mathcal{A} and combine this with the background (*static*) SLM from \mathcal{B} .

1. **Linear Interpolation:** $P(w_q \mid h_q) = (1 - \lambda) \cdot P_A(w_q \mid h_q) + \lambda \cdot P_B(w_q \mid h_q)$

2. **Backoff:**

$$P(w_q \mid h_q) = \begin{cases} P_A(w_q \mid h_q) & \text{if } C(h_q w_q) \geq T \\ \beta \cdot P_B(w_q \mid h_q) & \text{otherwise} \end{cases}$$

3. **Dynamic Cache Model:** $P(w_q \mid h_q) = \sum_{\{c_q\}} P(w_q \mid c_q) \cdot P(c_q \mid h_q)$

4. **Maximum a-Posteriori:**

$$P(w_q \mid h_q) = \frac{\epsilon \cdot C_A(h_q w_q) + C_B(h_q w_q)}{\epsilon \cdot C_A(h_q) + C_B(h_q)}$$

7.14.3 Constraint specification

Idea: Extract features from corpus \mathcal{A} that are used as constraints for the adapted SLM. Constraint specification is more powerful than Model Interpolation since different weights could be assigned to each extracted feature separately.

1. **Exponential Models:** Historically constraint-based methods are associated with exponential models trained on Maximum Entropy (ME) criterion which leads to Minimum Discrimination Information estimation.
2. **MDI Adaptation**
3. **Unigram Constraints**

7.14.4 Meta-information extraction

Idea: Use corpus \mathcal{A} to extract background information about the underlying subject matter. Improve upon the background model based on semantic classification.

- Mixture Models
- Semantic Knowledge
- Syntactic Infrastructure
 - Structured Language Models
 - Syntactic Triggers
- Multiple Sources
 - Combination Models
 - Whole sentence Models

8 Search / Decoding

The entire set of possible HMM state sequences is called the **search space**. It is not feasible to evaluate every possible sequence of the search space to find the most likely one. Therefore we need an intelligent algorithm that scans the search space and finds the best (or at least very good) hypothesis.

Isolated word recognition:

- for every applicable vocabulary word, compute it's score (DTW) and it's likelihood (Forward algorithm)
- report the word with the best likelihood / score

Continuous speech recognition:

- not only find state sequence in words
- but also find the word sequence

8.1 DTW optimization

The goal of DTW Review is to identify an example pattern that is most similar to unknown input. For this we also need to compare pattern of different length. To achieve this we find the alignment between the unknown input and the example pattern that minimizes the overall distance with a given transition function (\rightarrow Figure 11).

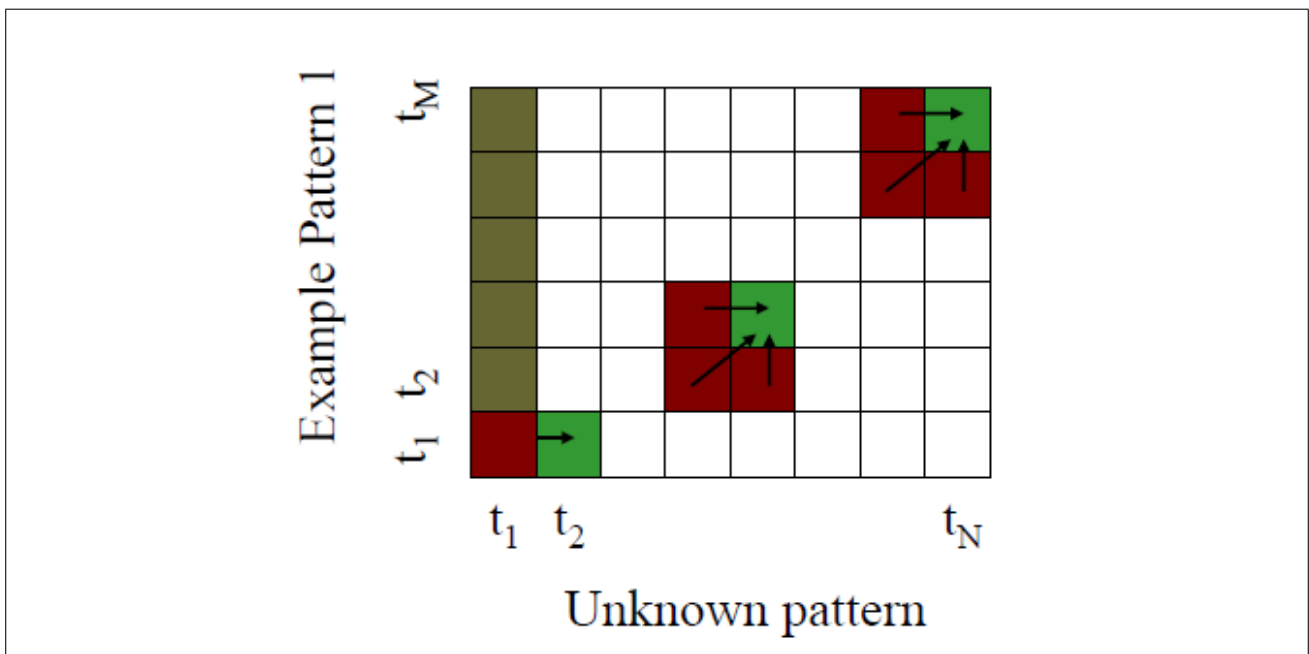


Figure 11: DTW Review

To save computation time we add constraints:

- define start and end point
- path should be close to diagonal
- no two successive horizontal / vertical steps
- pruning beam: eliminate losers (with score $s < s_{best} - \Delta$)
- usually, we're only interested in the final score, so we can overwrite older values

Drawbacks of DTW include speaker dependency and inefficiency for large vocabularies.

8.2 Viterbi optimization

The Viterbi algorithm finds the probability along with the most likely state sequence. If we operate in log space we can get rid of slow and error prone (overflow) multiplications. The state transition computation is then again based on distances (\rightarrow Figure 12):

$$-\log(P(x, y)) = \min(-\log(P(x-1, y)) + d(x, y), -\log(P(x-1, y-1)) + d(x, y))$$

$$d(x, y) = -\log(p(\text{frame}_x | s_y))$$

It works a lot like DTW (\rightarrow Figure 11). For both methods decisions on best predecessor states are final and scores and transition penalties have to be local.

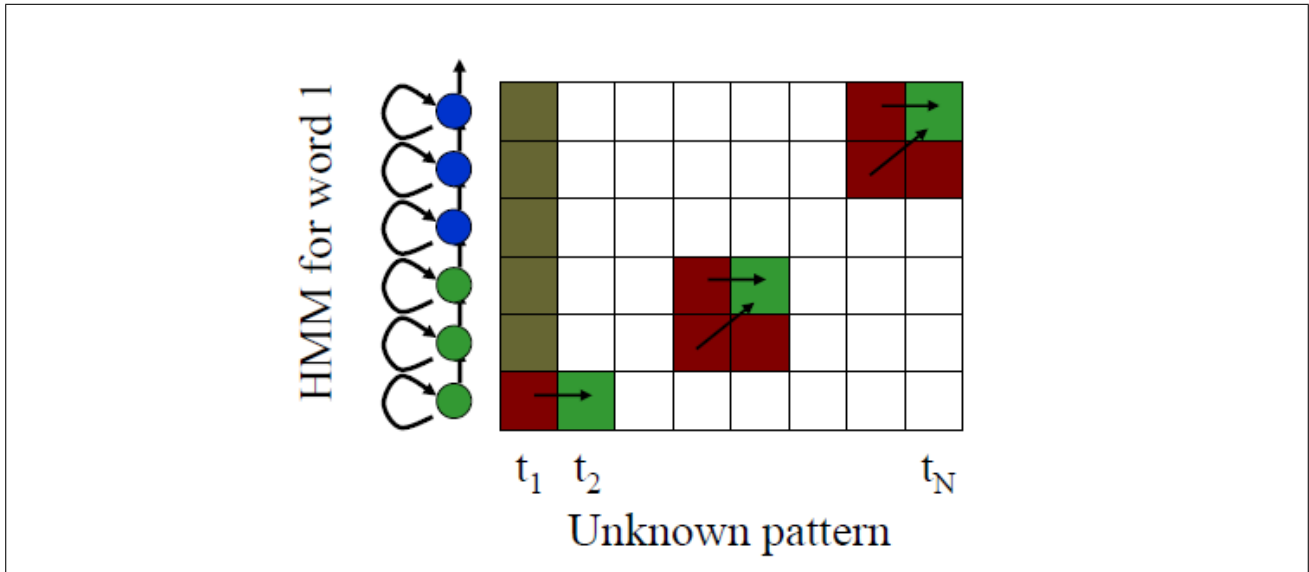


Figure 12: Viterbi Review

To further optimize the Viterbi algorithm, we use tree search: All words that start with the same model, start with exact the same computations (\rightarrow Figure 13).

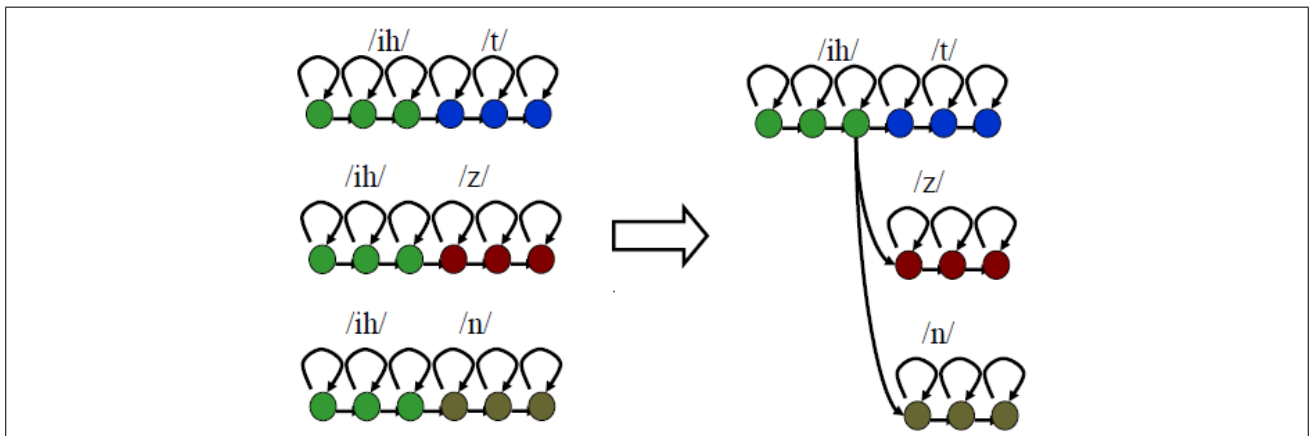


Figure 13: Viterbi tree search optimization.

8.3 From Isolated to Continuous Speech

In continuous speech different problems come together:

- sloppier
- higher speaking rate
- combinatorial explosion of things that can be said

- spontaneous effects (restarts, ...)
- Co-articulation
- Segmentation

There are several plans to solve those problems.

8.3.1 Cut Continuous Speech into Single Words

If we had an algorithm that would segment continuous speech into 1-word chunks, we could run DTW or Viterbi on each chunk. But right now there is no reliable segmentation algorithm for detecting word boundaries other than doing recognition itself.

8.3.2 Two-level DTW

We use DTW on two levels:

1. for each remotely likely word begin time, compute DTW with open end within all words
2. for each word start and end times calculated in the previous step, combine words with DTW based on calculated scores

Obviously, this is extremely costly for large vocabularies and longer words.

8.3.3 Depth-first search

Depth-first vs. breadth-first corresponds roughly to *time-asynchronous* vs. *time-synchronous*. In **time-synchronous** search, we expand every hypothesis simultaneously, while in **time-asynchronous** search, we expand partial hypotheses that have different durations (\rightarrow Figure 14).

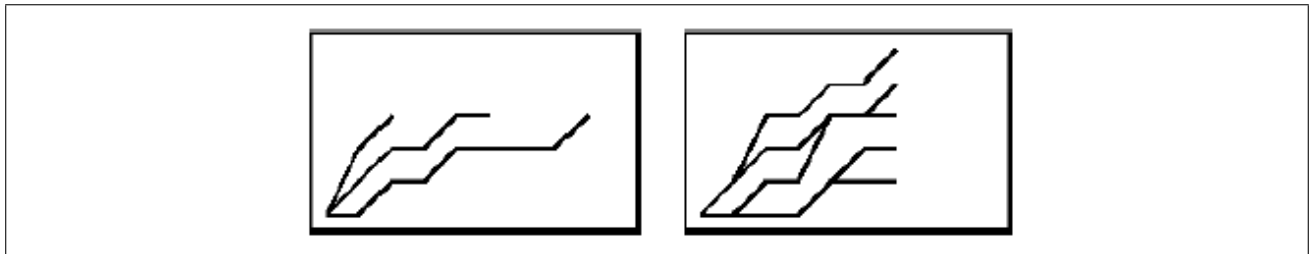


Figure 14: time-asynchronous vs time-synchronous

A^* search (usually implemented with a stack decoder):

A^* search is an example of depth-first search (or more specifically: admissible best-first search). With a stack decoder a time-asynchronous search can be implemented.

- Sort all partial hypotheses according to their accumulated score
- Expand the hypothesis with the highest success-expectation
- Success expectation = accumulated score + expectation of remainder score (A^*)
- Expansion of hypothesis increases its score (decreases likelihood) \rightarrow some other hypothesis might become best

8.3.4 Viterbi beam search vs. A^* stack decoder

Beam search doesn't require a heuristic and it is appropriate for parallel implementation. However, it is not admissible, that is it might not find the best solution. Nowadays, beam search is usually preferred over A^* (though A^* is taken for search through n -best and lattice structures).

8.3.5 One stage dynamic programming

Within words Viterbi is used. For the first state of each word, consider the best last state of all words as predecessor. In order to reconstruct the best word sequence, find the best predecessor word to the current word. Finally backtrack.

8.4 Optimization

All search techniques use two strategies for efficiency:

- **Sharing:** Keep intermediate results so that they can be reused by other paths without recomputation. For example, when computing the emission probabilities of “can” and “cash”, we can reuse the emission probabilities for the phonemes *C* and *AE*.
- **Pruning:** Disregard unpromising paths without wasting time exploring them further.

8.4.1 Pruning with Beam search

When using Beam search we define an “angle” for how much we want to look to the left and right. Typically that means we only expand states which have accumulated likelihoods greater than $beam \cdot likelihood_{best}$.

8.5 Searching with LMs

8.5.1 Problems

When using LMs the best predecessor is not the same for all words. When using bigrams, the best predecessor depends only on the current word, this is possible by storing one backpointer for each active word end. When using trigrams, the best predecessor also depends on the successor of the current word. Then, things “get ugly” and we have to approximate a solution.

8.5.2 Search with vs. without Language Model

The transition into an initial state of a word Z is computed by maximizing (Viterbi / DTW) the scores / accumulated distances $C(W)$ of all word-final states in the previous time frame and adding the local acoustic score $am(Z)$: e.g. $C(X) + am(Z)$ vs. $C(Y) + am(Z)$. When using a grammar, we additionally have to add to the accumulated score a language model score $lm(Z)$, e.g. $C(X) + am(Z) + lm(X, Z)$ vs. $C(Y) + am(Z) + lm(Y, Z)$. Without a grammar, the best predecessor state is the same for all word-initial states, whereas with a grammar, the best predecessor state also depends on the word transition probability.

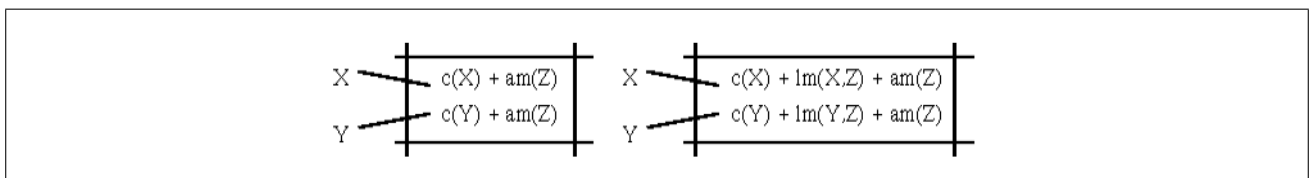


Figure 15: without grammars vs. with grammars

8.5.3 Tree-Search with LMs

When using tree search with a language model, we have a problem. After making a word-to-word transition, we don’t know which word we are entering, but this knowledge is required to properly compute the probability of the transition. The solution to this are **delayed bigrams**. We make every word’s final state unique. That way, we know which word we processed when we are in the final state. At that point, we can incorporate the language model information into our calculation (hence, *delayed*) (\rightarrow Figure 16).

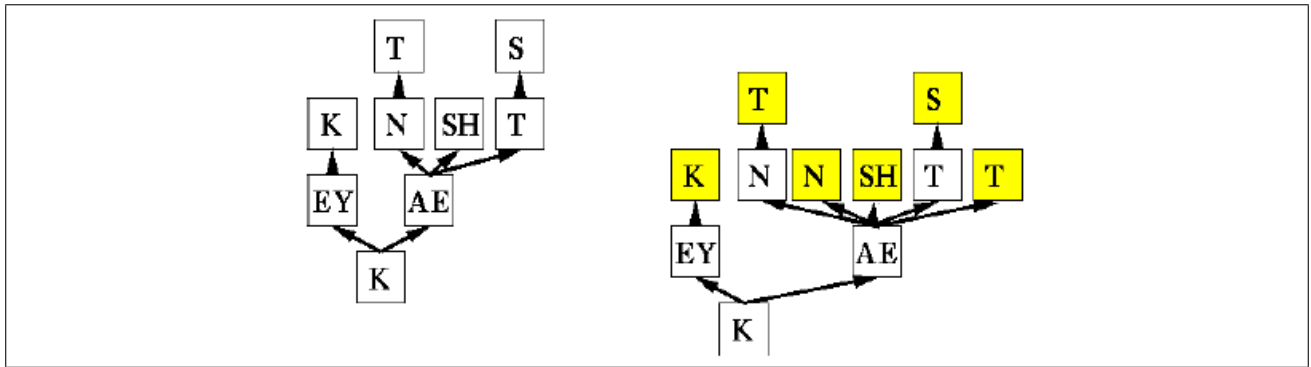


Figure 16: Delayed bigrams.

8.5.4 Unigram lookahead

No bigram information can be included until word identity is known. Instead, we estimate the unigram information of the remainder subtree and include the bigram information as soon as possible (see also figure 17).

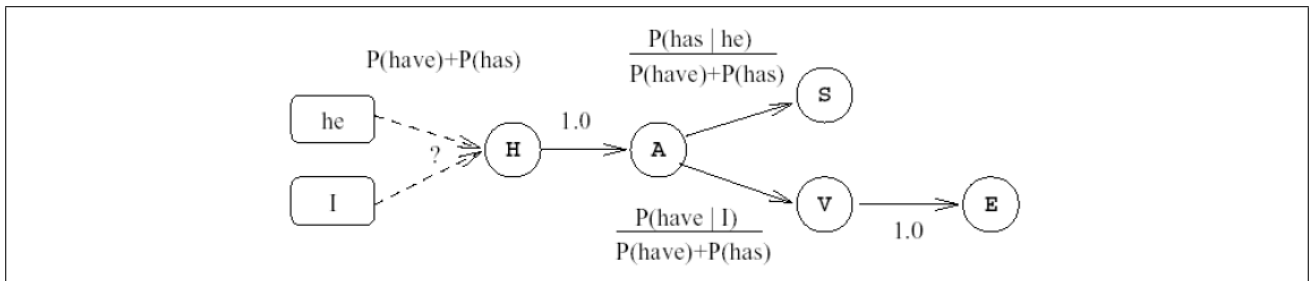


Figure 17: Unigram lookahead.

8.5.5 Multi-pass searches

Why use multiple passes in continuous speech recognition:

- we could use a good estimator for A^* search
- in pruning we could use a good lookahead to prune away the “right” parts
- recover from errors resulting from delayed bigrams

Forward-backward search: first, run backward pass, then run forward pass using backward scores to do pruning. Problems:

- when using a search pass to compute information for pruning, it has to be a lot faster than the actual search pass
- it cannot produce better results than the actual search pass
- no runon recognition possible (start processing before sentence has finished)
- where do we get backward bigrams from?

Example for a multi-pass search: in the first pass, use a narrow beam and a fast but weak acoustic model to prune some paths. In the second pass, use a regular Viterbi search with a wider beam and a slower but more powerful acoustic model, considering that some areas have already been pruned in the first run.

8.5.6 Multiple hypotheses

Reasons:

- do postprocessing on all hypotheses using additional knowledge
- speech understanding
- manual recovery options for the users

When recognizing isolated words, simply report not only the best, but the n best results. In continuous speech recognition, we can either use multiple recognizers and report all their results, or let Viterbi not only remember the best predecessor, but the n best predecessors.

A problem with multiple hypotheses is that the results are often quite similar in respect to their content words, while the non-content words are more variable. This results in many “useless” hypothesis. But if we simply increase n , the system may become too slow. The solution to this is to use **word lattices** (\rightarrow figure 18).

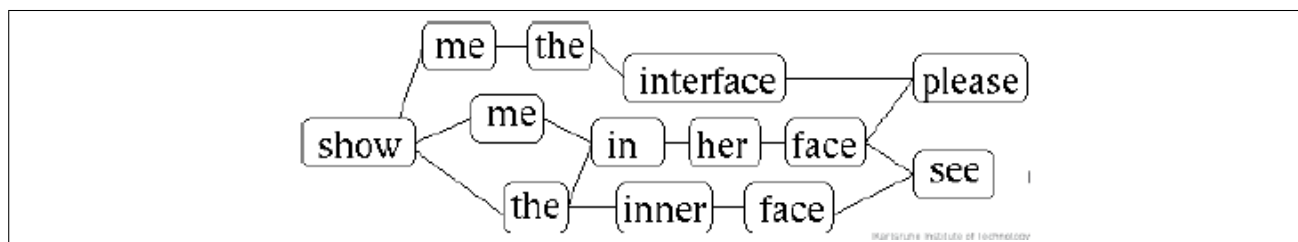


Figure 18: A word lattice.

8.6 Output Formats

There are different types of output:

- **1st-best** \rightarrow word string (baseline for WER)
- **N-best** \rightarrow N word strings (possibly lower WER, but more difficult)
- **Lattices** \rightarrow Time-marked directed, acyclic graph
- **Confusion networks** \rightarrow Directed graph with total ordering

8.7 Confidence Measures

Confidence can be associated with every item of a hypothesis, lattice, Confusion network, etc. It is useful for judging the output of a recognizer. Confidence measures can be derived from various features, e.g. Lattice Link probability, Hypothesis density or Acoustic stability.

8.8 Search with Context-Dependent Models

With Context-Dependent Models word-HMMs look a bit more complicated and the search tree becomes less compact.

8.9 Search with context dependent phonemes

We don't know the successor word when evaluating the last phone in a word. Therefore we keep multiple copies of last phonemes. If we are using delayed N -grams, the predecessor word is still unknown while evaluating the first phone. So we keep copies of the first phone for all varieties, transit from best into second phone, then when resolving the delayed N -gram, we fix the score by incorporating the score difference.

This makes the first and last phonemes the most expensive part of time-synchronous decoding. To solve this, we can use Phoneme Lookaheads. We predict how well each partial hypotheses will do in the next frames and then use the estimated future score to prune bad hypotheses.

9 Text-to-speech synthesis

“*Speech synthesis is the artificial production of human speech.*” Typically, text-to-speech synthesis involves three steps:

1. **Text Analysis:** from strings of characters to words
2. **Linguistic Analysis:** from words to pronunciations and prosody
3. **Waveform Synthesis:** from pronunciations to waveforms

We use the *mean mel cepstral distortion* (**MCD**) as a quality measure (smaller is better).

9.1 Text analysis

- character encodings
- word segmentation
- numbers, symbols, non-standard words
 - anything not directly in the lexicon
 - OOV or novel forms
 - abbreviations, letter sequences, acronyms
- may require special rules
 - train rules for homographs
 - numbers, roman numerals

Text processing errors are the most common complaints in TTS. Examples:

- “My cat who *lives* dangerously has nine *lives*.”
- “He stole \$100 from the bank.”
- “He stole *1996* cattle on 25 Nov. *1996*.”
- “It’s 13 *St.* Andrew *St.* near the bank.”

We employ a “bunch of hacky rules”:

- **splitter:** domain-independent tokenizer
- **token type classifier:** identify numbers and abbreviations, disambiguate homographs
- **token type expander**
- **language model**

Additionally there are models that help to analyze text from specific domains.

9.2 Linguistic analysis

The lexicon should be as comprehensive as possible, but this is impossible. So, we have to pronounce OOV words, too.

9.2.1 Bootstrapping

1. select some frequently occurring words and build lexical entries from them
2. select an article of text
3. synthesize each unknown word, add correct words to list, rebuild
4. repeat

Writing letter-to-sound rules by hand is difficult. So we can use an automatic process that builds them from lexicon using epsilon scattering, EM algorithm and CART models.

9.2.2 Prosody

- **Phrasing:** chunking in speech
- **Intonation:** Accents and F0 generation
- **Duration:** length of segments
- **Power:** Loudness

9.3 Waveform synthesis

Concatinative synthesis:

- random word / phrase concat
- phone concat
- diphone concat ($|\text{phones}|^2$ transitions)
- sub-word unit selection
- cluster-based unit selection

9.3.1 Diphones

- All phone-phone transitions in language
 - one occurrence of each
 - assume two-phone context is sufficient
 - may also include common consonant clusters
- Mid-phone is more stable so easier to join
- Inventory size: phones^2
- a safe way to build a synthesizer

9.3.2 Unit selection synthesis

Using a large, representative database with multiple examples of each unit type, we select *appropriate* units (phonetically and prosodically) with various unit selection algorithms (cluster, target-concatenation cost, structure matching). On the one side this method is famed for being *excellent*, but on the other it's also *incomprehensible*.

9.3.3 Cluster unit selection

- Cluster unit selection
- Decision tree partitioning on acoustic distance
- Viterbi selection through candidates

9.3.4 Build your own Synthetic Voices

The standard voice requires:

- A phone set
- Text Analysis
- Pronunciation
- Prosodic modeling
- Waveform Generation
- language dependent Extras (e.g. arabic vowels)

9.3.5 Multilingual Speech Synthesis

Unit selection is the standard method to select appropriate sub-word units from large databases. A relatively new technology is Parametric Synthesis “Clustergen (CG)”. CG is a HMM-generation based synthesis that uses cluster units to form models and generates from those models. Compared to Unit selection, CG can work with very little speech (about 10 minutes). But the speech sounds buzzy and lacks natural prosody.

10 Spoken dialog systems

A dialog is a verbal interaction of two or more agents over more than one utterance in order to establish a joint goal. A spoken dialog system is an autonomous, artificial system that engages in a dialog with a human and employs speech recognition and synthesis. Commonly, a complete interaction is referred to as a *session*, an *utterance* is one person's speech between two pauses, and a *turn* represents all utterances of one partner, until the other partner starts interacting. A *barge-in* happens when a turn is interrupted. There are two kinds of initiatives, the *system initiative*, e.g. "Please tell me your destination airport." (more rigid, less ASR errors) or the *mixed initiative*, e.g. "What flight are you interested in?" (less rigid, harder to recognize and understand).

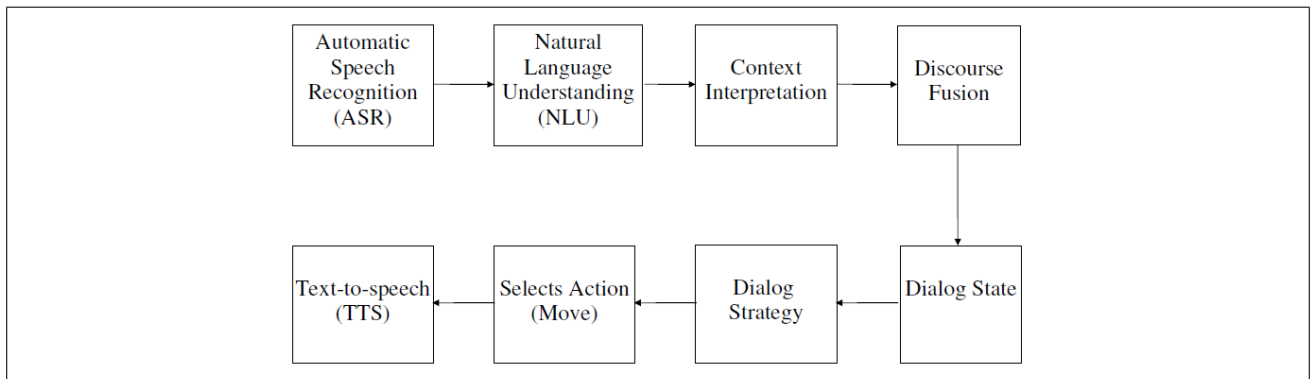


Figure 19: Components of a spoken dialog system.

A *discourse* is the entirety of information and structure of the ongoing dialog up to a certain point. Each new utterance must be interpreted in the light of the ongoing discourse and then included in it.

In order to represent a discourse, Montague grammars are commonly used (transform text into first order logic). Problem: "A man walks in the park. He whistles" $\rightarrow (\exists x : man(x) \wedge walksinpark(x)) \wedge (\exists x : whistles(x))$. What we need is *anaphora resolution*. Discourse Representation Theory (DRT) provides a construction algorithm based on syntax tree.

Don't forget *typed feature structures* (TFS). We can build *semantic grammars* with it. A TFS is a set of named attributes, so called features, which can either be of atomic type or another TFS. Two compatible TFS can be unified, resulting in a more specific TFS.

Semantic grammars are a convenient way for rule-based natural language understanding based on context-free grammar. Additionally a discourse can be separated into Attention, Intention and Discourse Structure.

10.1 Dialog strategies:

- **plan-based** system: the idea is that communication is the result of a plan (to do / achieve something)
- **grammar-based** system: there are adjacency pairs, such as *question* \rightarrow *answer*, *proposal* \rightarrow *acceptance*, etc.
- **finite-state grammars**: describe possible events with a finite state automata (simple to produce, but rigid)
- **rational agency**: formally describe rational behavior and then use formal proofs in order to deduce next steps
- **frame-based** systems: somewhat similar to web forms, additionally define *moves* (which may contain *conditions*) to ask for specific information
- **statistical systems**: learn optimal next system action from corpus of completed dialogs

The dialog system has to communicate with the ASR system. Returning only the best hypothesis is usually not sufficient. We may have to weigh hypotheses according to the dialog act expectations. Also, we should adapt the context to each turn (e.g. in a navigation system in Karlsruhe, “*Durlach*” will be more likely than “*Paris*”, which is different from most other contexts).

If our system is *multimodal*, we can also use gestures, etc. ALSO: Dialog systems should adapt to different types of users. This problem is currently not solved yet. A first attempt is to detect the user state by using multimodal fusion of biosignals.

11 Multilingual speech processing

We can't simply retrain on foreign data, because:

- different scripts, no vowelization, no writing system
- no word segmentation, rich morphology
- tonality, click sounds
- social factors such as trust, access, exposure and cultural background

For an overview of the system, see figure 20.

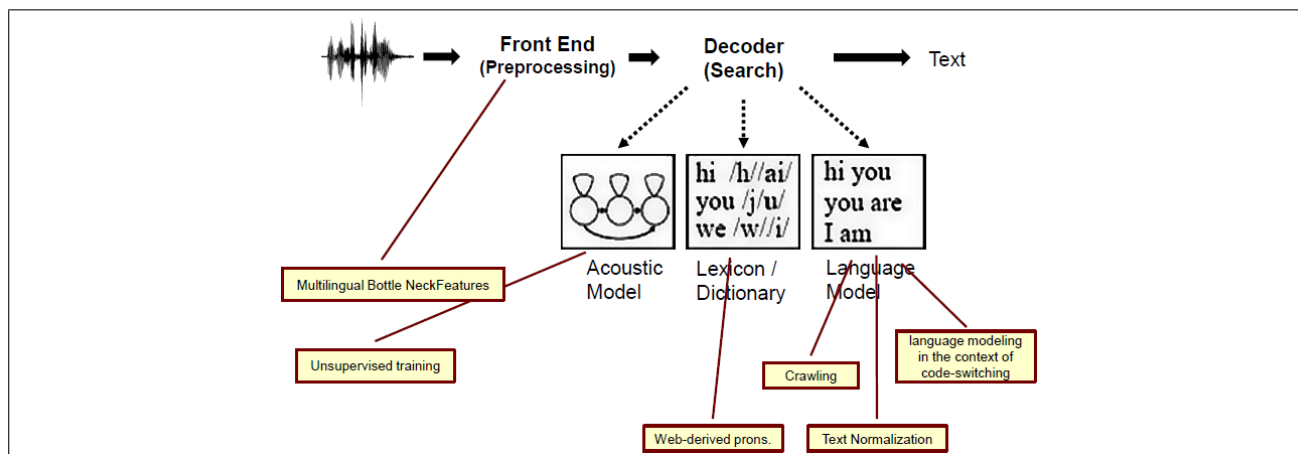


Figure 20: Overview of multilingual speech processing systems.

Let's try to build a language independent ASR system!

- language independent acoustic modeling
 - *International Phonetic Alphabet* easy to implement, easy to port to other languages
 - fully data-driven procedure
- universal language model: combine LMs of different languages to allow code switching

11.1 Multilingual acoustic modeling

1.
 - uniform multilingual speech database
 - monolingual recognizers for many languages
2.
 - combine monolingual acoustic models
 - share data across languages

11.2 Acoustic model combination

Sound production is *human-specific*, not *language-specific*. Thus, it is sufficient to represent sounds with the IPA.

1. build universal sound inventory based on IPA (485 sounds → 162 IPA sound-classes)
2. each sound class is represented by one phoneme which is trained through data sharing:
 - *m, n, s, l* occur in all languages
 - *a, e, i, o, u* and *p, b, t, d, k, g, f* occur in most languages
 - some data is not shared

11.3 Challenges

- Many ASR algorithms are language independent, but they require lots of training data. For many languages, not a lot of training data exists, so we need a different approach. For example, we might use intelligent learning systems.
- The contexts of sounds are language specific. How can we still use context dependent modeling?
 - multilingual decision context trees
 - specialize decision tree by adaptation

11.4 Web-derived pronunciations

Idea: Generate pronunciations based on phonetic notations found in the WWW using sites like Wiktionary.

11.5 Unsupervised training

Idea: decode untranscribed audio data, select data with high confidence, use selected data to train / adapt recognizer.

12 Prüfungsfragen

12.1 Allgemeine Fragen

- Was ist MMMK? Wofür steht MMMK?
 - *Multilinguale Mensch-Maschine Kommunikation*
- Wo findet MMMK Anwendung?
 - *Interaktion mit einem Computer/Maschine mittels natürlicher Sprache*
- Vor- und Nachteile gegenüber anderen Kommunikationsmitteln
 - *siehe Vor- und Nachteile im Skript*
- Welche Probleme können dabei auftreten?
- Zeichnen sie das Blockschaubild (Young oder Schultz)
 - *siehe Overview Figure 1 und 2*
- Geben Sie die Fundamentalformel (nach Bayes) an.
 - $\hat{W} = \operatorname{argmax}_W P(W|X) = \operatorname{argmax}_W \frac{P(X|W) \cdot P(W)}{P(X)}$
- Beschreiben Sie den Zusammenhang zwischen Fundamentalformel und Blockschaubild.
 - *$P(X|W)$ kommt vom Akustischen Modell, $P(W)$ vom Sprachmodell, $P(X)$ ist die a-priori Wahrscheinlichkeit, argmax ist der Decoder und \hat{W} die wahrscheinlichste Hypothese.*
- Was ist an Umgangssprache so schlimm?
 - *Grammatisch oft inkorrekt, Aussprache ggf. undeutlich bzw. untypisch*
- Was ist Co-Artikulation?
 - *Wenn die Aussprache eines Phonems/Wortes durch den umgebenden Kontext verändert wird.*
- Was ist ein Phonem?
 - *Kleinste Bedeutungsunterscheidende akkustische Einheit*
- Wie werden false starts modelliert?
 - *Als Hypothese bekommt man genau das raus, was gesagt wurde inklusive aller Mhms und Ähs.*

12.2 Vorverarbeitung

- Erklären sie die Funktionsweise eines Kondensatormikrophons.
 - *Zwei Membranen, eine dicke und eine dünne, die als Plattenkondensator fungieren. Die dünne Membran wird durch Schalldruckwellen angeregt und schwingt. Durch die Kapazitätsänderung kommt es zu einer messbaren Spannungsänderung.*
- Was macht man bei der Vorverarbeitung?
 - *Tiefpassfilter (Anti-Alias-Filter)*
 - *Abtasten (Sampling) [X-Achse diskret machen]*
 - *Quantisieren [Y-Achse diskret machen]*

- Fenstern
- Fouriertransformation um an Frequenzinformation zu gelangen
- Filterbänke anwenden zur Reduzierung der Dimension
- Logarithmieren (um einfacheren den Kanal loszuwerden)
- inverse Fouriertransformierte berechnen
- Ergebnis: Cepstralkoeffizienten
- Alternative: Linear Predictive Coding (berechne aus bisherigen Signalwerten die dann kommenden)
- Was ist ein Vorteil vom Cepstrum?
 - Der Vokaltrakt ist bei allen Menschen etwa gleich und daher sind die Cepstralkoeffizienten relativ robust und sprecherunabhängig.
- Was muss beim Abtasten (Sampling) beachten?
 - *Shannon-Nyquist-Theorem: Abtastfrequenz muss echt größer als der doppelten Signalfrequenz sein.*
- Was machen Quantisierung und Sampling?
 - *Quantisierung diskretisiert die Y-Achse und Sampling die X-Achse.*
- Was für ein Filter wird beim Sampling benötigt und warum?
 - *Tiefpassfilter, da wir ein bandbegrenztes Signal brauchen.*
- In welchem Frequenzbereich hört der Mensch?
 - *bis zu 32k Hz, Abtastfrequenz ist üblicherweise 8kHz*
- Warum wollen wir die Frequenzinformation haben?
 - *Das Ohr macht das auch so.*
- Was macht man nach dem Digitalisieren des Signals?
 - *Nach der Digitalisierung wird das Signal gefenstert.*
- Was für Fensterformen gibt es?
 - Rectangle, Gaussian, **Hamming**, Hanning
- Was wird beim Fenstern gemacht? Und warum wird das Signal gefenstert?
 - Man multipliziert das Signal mit einem Finite-Duration-Window und berechnet dann die DTFT. Das muss gemacht werden, da man ansonsten die zeitlichen Informationen nach der FT verlieren würde.
- Wieso muss man anschließend das Signal noch filtern? Was ist die Mel-Filterbank?
 - Viele der Fourierkoeffizienten enthalten redundante oder gar irreführende Informationen über die Mikrostruktur. Diese werden nicht benötigt. So reduziert man die Dimensionalität. ...
- Wie trennt man das stimulierende Signal von der "impulse response"?
 - Es gilt $s(n) = u(n) * h(n)$.
 $\rightarrow_{FT} F(s) = F(u) \cdot F(h)$
 $\rightarrow_{LOG} F^{-1}(\log(F(s))) = F^{-1}(\log(F(u))) + F^{-1}(\log(F(h)))$

- Was ist MFCC? Und LPC?
 - MFCC (Mel-Filter-Cepstral-Coefficients) bzw. Linear Predictive Coding, s. Skript
- Was macht man mit einem Featurevektor?
 - Berechnen der Deltas und DeltaDeltas um Zeitinformationen zu erlangen.
- Erläutere das Quelle-Filter-Model (Source-Filter Model).
 - s. Skript

12.3 Akkustisches Modell

- Zeichne HMM für ein Phonem/Wort.
 - *siehe HMMs*
- Was für Topologien gibt es? Welche wird üblicherweise verwendet?
 - *Linear, Left-to-Right, Alternative paths, Ergodic, **Bakis***
- Warum benötigen HMMs 3 Zustände pro Phonem?
 - *Beginn, Mitte, Ende*
- Woraus setzt sich eine HMM zusammen?
 - Fünftupel (S, π, A, B, V) :
 - * Zustandsmenge $S = \{s_1, s_2, \dots, s_n\}$ – n is the number of states
 - * Initiale Wahrscheinlichkeitsverteilung $\pi(s_i) = P(q_1 = s_i)$
 - * Matrix der Zustandsübergangswahrscheinlichkeiten $1 \leq i, j \leq n$,
 $A = (a_{ij}), a_{ij} = P(q_{t+1} = s_j | q_t = s_i)$
 - * Menge der Emissionswahrscheinlichkeiten $B = \{b_1, \dots, b_n\}, b_i(x) = P(o_t = x | q_t = s)$
 - * Symbolmenge $V = \{x_1, x_2, \dots, x_v\}$ (discrete) or $V = R^d$ (continuous)
- Was sind die drei Hauptprobleme von HMMs und mit welchem Algorithmus kann man sie jeweils lösen?
 - *Evaluation (Forward), Decoding (Viterbi) und Learning Problem (Forward-Backward Algorithm / Baum-Welch) [ausführliche Erklärung der Probleme und Algorithmen s. Skript]*
- Schreiben Sie [Forward/Viterbi/Forward-Backward/Baum-Welch] mit sämtlichen Formeln auf und erklären Sie die Vorgehensweise und Unterschiede.
 - *s. Skript*
- Was steht in einem Pronunciation Dictionary?
 - eine Liste von Wörtern mit zugehörigen Aussprachmöglichkeiten
- Wie sind die HMMs gestaltet, sodass man mit weniger Trainingsdaten bessere Ergebnisse erzielt?
 - *Die bei den Emissionswahrscheinlichkeiten verwendeten Gaußverteilungen werden kombiniert und verschieden gewichtet.*
- Wie verbindet man mehrere Wörter hintereinander?
 - (Sprachmodell)
- Wie lange ist man mindestens in einem State?

- 10ms
- Woher bekommt man die HMMs?
 - Im Pronunciation Dictionary steht, aus welchen Phonemen ein Wort besteht. Das baut man dann für die Wörter zusammen.
- Nenne ein Beispiel in dem Viterbi und Forward-Algorithmus nicht zum gleichen Ergebnis kommen.
 - viele Pfade mit ähnlicher Wahrscheinlichkeit, der wahrscheinlichste führt zu Wort X, alle anderen zu Y. Viterbi findet X, Forward findet Y
- Wie trainiert man HMMs?
 - Mit einem Expectation-Maximation-Algorithmus (EM-Algorithmus), wie beispielsweise Baum-Welch.
- Durch welche Parameter sind Gaußmixturenverteilungen definiert?
 - Mittelwert, Varianz und Mixturegewicht
- Welche Probleme können bei HMMs in der Modellierung im Sprachmodell auftreten?
 - Beim Eintritt in die HMM weiß man noch nicht, in welchem Wort man ist. Das ist bspw. für Bigramme schlecht. Daher verwendet man Delayed-Bigrams.
- Wie kann man Koartikulation modellieren?
 - Jedes Phonem für jeden Kontext modellieren. Zu viele Kontexte, also Kontexte in Klassen zusammenfassen, via bottom-up oder top-down.
- Wie kann man Viterbi (u.ä. Algorithmen) optimieren?
 - Beam Search
- Wie können unterschiedliche Aussprachevarianten modelliert werden? (Bsp.: T in True und Tall)
 - Markieren im Dictionary ODER Triphone (s. Skript)
- Wie funktioniert Top-Down oder Bottom-Up Clustering
 - s. Skript

12.4 Sprachmodell

- Was ist der Unterschied zwischen einem deterministischen und einem stochastischen Sprachmodell?
 - In einem deterministischen LM [Grammatik] hat ist $P(W)$ für jede Wortsequenz W entweder 1 oder 0. In einem stochastischen ist $P(W)$ eine Wahrscheinlichkeit.
- Geben Sie die Wahrscheinlichkeit für ein N-Gram an.
 - $P(w_k | w_1 w_2 \dots w_{k-1}) = P(w_k | w_{k-(n-1)} w_{k-(n-2)} \dots w_{k-1})$
- Wie werden die anfangs berechnet?
 - Man nimmt einen großen Textcorpus und zählt die Häufigkeiten. $P(w | history) = \frac{Count(history \cdot w)}{Count(history)}$
- Was ist Smoothing?

- *Allen Zeichketten wird eine Wahrscheinlichkeit ungleich 0 zugewiesen um ASR Fehler zu vermeiden.*
- Welche Smoothing-Verfahren gibt es?
 - *Add-One, Backoff Smoothing (Absolute discounting, Katz, Kneser-Ney) und Linear Interpolation (Deleted interpolation) [ausführliche Erklärungen der Verfahren s. Skript]*
- Wie funktioniert Class-Based Interpolation?
 - s. Skript

Danksagung

Guido Brückner für die Kapitel Overview, Basics, Signal processing, Hidden Markov Models, Acoustic modeling, Language modeling, Search, Text-to-speech synthesis, Spoken dialog systems und Multilingual speech processing