**Overview and summary of important topics taught**

**NPFL038, Fundamentals of Speech Recognition and Generation**

**Vilém Zouhar, January 2020**

# Phonetics and Phonology

## Sound and Human Speech Systems

*Source: Huang - Chapter 2.1*

### Components of speech transmission

- Message formulation
- Language system
- Neuromuscular mapping
- Vocal tract system
- *Sound transfer via air*
- Cochlea motion
- Neural transduction
- Language system
- Message coprehension

### Intensity perception

Sound amplitude is measured on an adjusted logarithmic scale: $10 \cdot \log_{10}\left(P/P_0\right)$, where $P_0$ is the lowest audible power level (amplitude for 1kHz). The graph of minimal power needed for a given frequency (from $10^1$ to $10^4$) is a skewed U curve with minimum between $10^3$ and $10^4$ (human speech).

### Speech production

Human articulators include:

- Teeth
- Lips
- Nasal cavity
- Hard palate
- Velum
- Tongue
- Vocal cords

Human sounds are often distinguished into two groups: consonants and vowels. Consonants include an obstruction in the airflow, whilst in vowels, we just adjust the tube the air flows through. There is a chart for both consonants and vowels, which maps articulator position to IPA. Vocal cords can produce the fundamental frequency $F_0 \approx 100 - 300kHz$ (zeroeth formant), which is later modified by the tube.

### Spectrograms

We analyze sounds by taking a look at their spectrograms. We pick an interval (e.g. 5ms) and put the collected data points through the Fourier transform, which tells us which frequencies with which intensities are represented in the recording. We put this data into one bar ($\mathbf{y}$ is the frequency, point intensity is the frequency value). Stacking multiple bars next to each other provides us with a histogram.

## Basics

*Source: Huang - Chapter 2.2*

### Phonemes

Basic building blocks of human sounds. All possible human sounds are codified in the International Phonetic Alphabet (which also distinguishes between possible and used, possible and not used, and not possible).

Vowels can be recognized by a unique $(F_1, F_2)$ combination and also by the lip shape and tongue position. This creates a two-dimensional graph. A special class of vowels are diphthongs (two vowels joined together), which is produced by moving the articulators while articulating. Consonants are created with an obstruction. We distinguish the following manners:

- plosive - the airflow is stopped for a while
- nasal
- fricative - turbulent airstream
- retroflex
- lateral - the airflow goes by the sides of the mouth
- glide

together with the following places of obstructions:

- labial
- labio-dental
- dental
- alveolar
- palatal
- velar
- glottal

A two-dimensional graph is part of IPA.

### Allophones

We consider two sound to be allophones if the speakers consider them to be the same (if they do not distinguish two different words and the rule can be described). For example /l/ has many different allophones in English. This is part of phonology (language-specific study).

# Vector Quantization

*Source: Fink - Chapter 4*

The task of vector quantization is to assign every vector from possibly infinite space to some representation vector with minimal error. The number of representation vector is fixed. We can look at some finished vector quantization model either as a *codebook* of vector representants or as a partition of all possible vectors. One implies the other. From partition, we must choose the centroid and the other way around we can assign every vector the closes representant in the codebook. This minimizes the overall error (average of distances). The optimality is described in the book as the *nearest neighbour condition* and the *centroid condition*.

## Algorithms

The goal of vector quantization algorithms is to find the best possible codebook (of a given size) given a sample set of data. Since there is no closed-form formula for computing the best codebook, so all algorithms are inherently suboptimal.

### Lloyd's Algorithm

This algorithm is based on the dual view (codebook vs. partition). We start with a random codebook, define the partition, compute the new codebook etc.

1. Randomly select $N$ vectors from the sample space
2. $R_i = \{x | y_i = \text{argmin } d(x, y_i)\}$
3. $y_i = cent(R_i)$
4. If error rate under some threshold, exit; otherwise `goto 2`.

It is very intuitive, but the drawback is that it is very sensitive to initialization and can take long to compute when unoptimized. It is sometimes incorrectly called the *k-Means Algorithm*

### LBG Algorithm

This algorithm is deterministic. It is based on splitting each representation vector into two distinct.

1. Start with $Y = \{cent(S)\}$
2. Set $Y = \{y_1 - \epsilon, y_1 + \epsilon, y_2 - \epsilon, y_2 + \epsilon, \ldots\}$
3. Optimize this codebook by applying Lloyd's algorithm
4. If desired size of codebook reached, exit; otherwise `goto 2`

Without further augmentations, it can produce codebooks with $2^n$ vectors.

### k-Means Algorithm

This algorithm aims to be faster than the previous ones. It does not recompute the whole partition at once, but incrementally adds single vectors and recomputes only the affected partition.

1. Randomly select $N$ vectors from the sample space, which will be the initial codebook
2. Take a yet unprocessed sample vector $x$ (sequentially)
3. Find the corresponding partition: $y_i' = \text{argmin } d(x, y_i')$
4. $R_i = R_i \cup \{x\}$, $y_i = cent(R_i)$
5. Stop when all sample vectors processed

## Mixture density models

Instead of describing only a discrete number of points, we can make use of normal distributions to model the probability landscape much more precisely. The first idea would be to take the result of one of the previous algorithms and substitute the representation vectors with a normal distribution (with parameters from the partition). This, however, does not yield useful results. A small improvement would be to use different metrics during the computation instead of the euclidean one, such as *Mahalanobis distance*.

### EM Algorithm

This algorithm defines a general way by which to create a probability model (as a sum of Gaussians) from a set of sample vectors. It tries to maximize the probability of seen data given the parametric boundaries (limited number of parameters). The probability of a single vector given the means $\mu_i$, the covariances $C_i$ and prior probabilities $c_i$: $p(x|\theta) = \sum c_i \mathcal{N}(x|\mu_i, C_i)$ ($\mu$ and $C$ are part of $\theta$).

The probability of the training data can be written as $\sum \ln p(x_i|\theta) = L(\theta|\{x_1, x_2, \ldots\})$. This way we can represent that we pick the best possible model. Since the likelihood function is monotonic, maximizing $L(\theta|\{x_1, \ldots\})$ is the same as maximizing $\sum \ln p(x_i|\theta)$.

Input: $\omega = \{x_1, x_2, \ldots, x_T\}$, output: codebook of size $N$

1. Choose model parameters $(c_i, C_i, \mu_i)$ at random
2. Compute class posterior probabilities for every $x \in \omega$:

$$P(\omega_i|x, \theta) = \frac{c_i \cdot \mathcal{N}(x, \mu_i, C_i)}{\sum c_j \cdot \mathcal{N}(x, \mu_j, C_j)}$$

3. Compute data likelihood given the model:

$$L(\theta|\omega) = \sum_{x \in \omega} ln \sum c_j \mathcal{N}(x|\mu_j, C_j)$$

.

4. Maximize the parameters:

- Class probability is the average from all of data, this makes it also normalized.

$$c_i = \frac{\sum_{x \in \omega} P(\omega_i|x, \theta)}{|\omega|}$$

- Centers are computed as average of vectors which are significant for a given class:

$$\frac{\sum_{x \in \omega} x \cdot P(\omega_i|x, \theta)}{\sum_{x \in \omega} P(\omega_i|x, \theta)}$$

- Covariances (assuming independence) are computed also from vectors which are significant for a given class:

$$\frac{\sum_{x \in \omega} xx^T \cdot P(\omega_i|x, \theta)}{\sum_{x \in \omega} P(\omega_i|x, \theta)} - \mu_i \mu_i^T \Big( = \frac{\sum_{x \in \omega} (x - \mu_i)(x - \mu_i)^T \cdot P(\omega_i|x, \theta)}{\sum_{x \in \omega} P(\omega_i|x, \theta)} \Big)$$

5. Terminate if the difference in likelihood is negligible: $\frac{L(\theta^m|\omega) - L(\theta^{m-1}|\omega)}{L(\theta^m|\omega)} < \epsilon$.

## Usage

Vector quantization (especially mixture density models) is used in Speech Recognition (with HMMs) to model the probability of a (speech) vector in a specific state, which stores some probability model (that we train).

# HMM

*Source: Fink - Chapter 5*

HMM is generally a five-tuple $(S, s_0, \Sigma, A, B)$:

- $S$ - set of all states
- $s_0$ - the initial state
- $\Sigma$ - the output alphabet
- $a_{ij}$ - transition probability from $s_i$ to $s_j$
- $b_i(z)$ - probability of producing $z$ in state $s_i$ (in more general HMM the symbol production happens at the transitions)

We have two goals:

- given an HMM compute the probability of it generating a specific string (+computing the most probable sequence which generated the string)
- given a set of training data and an HMM architecture find the parameters which best explain the data

## Trellis (Forward algorithm)

We could compute the probability of the string being generated by this HMM by considering all possible paths. This could be done by a brute force approach (simply using recursion), but this would lead to $O(TN^T)$ time complexity, which is undesirable. Since HMM have limited memory and are time-invariant, we can make use of dynamic programming.

We proceed by time steps. In each step for every state, we sum the probabilities from the previous states multiplied by the transition and multiply all of this by the probability of producing the desired symbol.

$$\alpha_{t+1}(j) = \big(\sum \alpha_t(i) a_{ij}\big) \cdot b_j(O_{t+1})$$

This way we only need $O(N)$ memory and $O(TN)$ time. At the end the value $\sum_i \alpha_T(i)$ is the probability of the string being generated in any way.

The alpha values are called *forward variables*.

## Viterbi evaluation (recognition)

The previous approach was precise in computing but did not give us any hint about which sequence of states generated the output string. It can be easily modified by changing the iteration step from summation to the max function.

$$\alpha_{t+1}(j) = \big(\max \alpha_t(i) a_{ij}\big) \cdot b_j(O_{t+1})$$

At this point we also have to add a pointer at $s_j$, which points to argmax $\alpha_t(i)a_{ij}$, so that we can later reconstruct the sequence. This time we need $O(TN)$ space, but still $O(TN)$ time. We choose the sequence which ends with argmax $\alpha_T(i)$. We also get the probability of this sequence generating the output string for free.

## Parameter estimation

The second task with HMM deals with parameter estimation. As in vector quantization, we want to model the probability of the training data. We will need to compute the

probability of the automata being in a state $s_i$ at time $t$. Please note that the forward probabilities are not enough, because we do not know whether a given path will be successful or not. Generally we can approximate the parameters as:

$$\hat{a}_{ij} = \frac{\text{expected number of transitions from } i \text{ to } j}{\text{expected number of transitions from } i}$$

## Forward-Backward algorithm

To compute the probability of automata being at state $s_i$ in time $t$ ($P(S_t = i|O, \lambda)$ we will need something symmetrical to forward variables, but which can also model the path from the end. Given an HMM and an output string $O$ we can compute the forward variables $\alpha$, but starting from the end (of the HMM and also of $O$) we can get backward variables $\beta$ in the very same way. We can then get the posterior probability:

$$\gamma_t(i) = P(S_t = i|O, \lambda) = \frac{\alpha_t(i)\beta_t(i)}{P(O|\lambda)}$$

## Baum-Welch

One iteration of Baum-Welch tries to increase the probability of the training data. It does it so using the previous general formula based on expected transitions.

$$\hat{a}_{ij} = \frac{\sum_t P(S_t = i, S_{t+1} = j|O, \lambda)}{\sum_t P(S_t = i|O, \lambda)} = \frac{\sum_t \gamma_t(i, j)}{\sum_t \gamma_t(i)}$$

$$\hat{b}_j(o_k) = \frac{\sum_t P(S_t = j, O_t = o_k|O, \lambda)}{\sum_t P(S_t = j|O, \lambda)} = \frac{\sum_{t:O_t=o_k} P(S_t = j|O, \lambda)}{\sum_t P(S_t = j|O, \lambda)} = \frac{\sum_{t:O_t=o_k} \gamma_t(j)}{\sum_t \gamma_t(j)}$$

We can define the $\gamma$ variable for transitions as:

$$\gamma_t(i, j) = P(S_t = i, S_{t+1} = j|O, \lambda) = \frac{P(S_t = i, S_{t+1} = j, O|\lambda)}{P(O|\lambda)} = \frac{\alpha_t(i)a_{ij}b_j(O_{t+1})\beta_{t+1}(j)}{P(O|\lambda)}$$

This should be very intuitive, as we take the forward probability, compute the transition and production probability and multiply it by the backward probability.

### Mixture components

When using mixture models we will need to estimate also the gaussian parameters instead of $b_i\hat{(o)}$. The parameters are:

- Mixture class weights $c_{jk}$ (state $j$, class $k$)
- Mean vectors $\mu_{jk}$ (state $j$, class $k$)
- Covariance matrix $C_{jk}$ (state $j$, class $k$)

We also need to define a mixture density alternative to $\gamma_t(i)$, which also incorporates the selection of one specific gaussian:

$$\xi_t(j, k) = P(S_t = j, M_t = k|O, \lambda) = \frac{\sum_i \alpha_{t-1}(i)a_{ij}c_{jk}\mathcal{N}(X|\mu_{jk}, C_{jk})\beta_t(j)}{P(O|\lambda)}$$

Intuitively one can think of this as the *total* (forward-backwards) probability of being in a state $j$ at time $t$ and choosing a specific gaussian class $k$.

The class weight for a state can be then then approximated by normalizing the sum of probabilities of choosing the specific class:

$$\hat{c}_{jk} = \frac{\sum \xi_t(j,k)}{\sum \gamma_t(j)}$$

Similarly as in the EM algorithm, we can approximate the mean and the covariance:

$$\hat{\mu}_{jk} = \frac{\sum \xi_t(j,k) x_t}{\sum \xi_t(j,k)}$$

$$\hat{C}_{jk} = \frac{\sum \xi_t(j,k)(x_t - \hat{\mu}_{jk})(x_t - \hat{\mu}_{jk})^T}{\sum \xi_t(j,k)} \Big( = \frac{\sum \xi_t(j,k) x_t x_t^T}{\sum \xi_t(j,k)} - \hat{\mu}_{jk}\hat{\mu}_{jk}^T \Big)$$

Also the transition probability remains:

$$\hat{a}_{ij} = \frac{\sum \gamma_t(i,j)}{\sum \gamma_t(i)}$$

**Algorithm**

1. Initialize all parameters and Gaussians from the sample distribution
2. Update parameters by the previous formulas
3. Recompute the training data probability $P(O|\lambda)$. Repeat if it improved significantly.

## Viterbi training

The difference between Baum-Welch and Viterbi training is very similar to the difference between Trellis and Viterbi evaluation. Instead of the forward and backward variables computed from Trellis, we will use their deterministic counterparts. We use $\chi_t(i)$ instead of $\gamma_t(i)$ or $\xi_t(j,k)$. It is defined as 1 iff $s_i$ is the $t$-the state from the best scoring sequence, otherwise 0.

**Algorithm**

1. Initialize all parameters and gaussians from the sample distribution
2. Compute the best scoring path using Viterbi evaluation
3. Recompute the probabilities

- State transition probability:

$$\hat{a}_{ij} = \frac{\sum_t \chi_t(i)\chi_{t+1}(j)}{\sum_t \chi_t(i)}$$

- Symbol output probability:

$$\hat{b}_j(o_k) = \frac{\sum_{t:O_t=o_k} \chi_t(j)}{\sum_t \chi_t(j)}$$

4. Recompute the training data probability $P(O|\lambda)$. Repeat if it improved significantly.

Viterbi training is much faster and intuitive, but we are unable (it is much more complicated) to use mixture models and no direct analytical formulas exist.

# Speech Synthesis

*Source: Tokuda et. al. - Section 1 - 3, Huang - Chapter 4*

## F0

In speech recognition, we usually disregard the excitation part. In speech synthesis we this is modelled by a pulse and white noise function. The system (excitation) then flips between these two. The input vectors are timestamps with spectral parameters (mel-cepstral) + derivatives and excitation ($\log F_0$) + derivatives. Because of voicing, we need to distinguish between the used frequency and not voicing. This creates a mixed probability space with both continuous values and one discrete symbol (0). The training algorithm needs to consider this.

The resulting speech vectors are then produced as a combination of this excitation function and linear time-invariant system.

## State duration

Given a phonetic transcription, we can construct and then run an HMM. We are not interested in the produced vectors (the sound quality would be very bad), but more important are the state durations. Then we assume the whole state has a certain mean and variance and try to create a smoothed curve through the whole graph (illustrated well in Fig. 5 in *Tokuda et. al.*).

During initialization, we can use phoneme-level aligned data, so that we know the average duration of each phoneme.

## Dynamic features

Derivatives are part of the observation vector and so have their own set of gaussian parameters. They have to be taken into consideration when smoothing the curve (the derivatives define the smoothing). We need this, because given a state sequence, the probability would be maximized by using an average vector from each state. This does not happen in reality, as there are transitions.

## Context dependency

To improve system quality, we can add linguistic information to the observation vector. Such information can be:

- Current, preceding and succeeding phonemes
- Position within the current syllable, word or phrase
- Current, preceding and succeeding stress and accent
- Adjacent syllables lengths

Encoding for each feature must be thoroughly considered, as some are probabilities and some discrete values.

## Speaker characteristics

We can start with an *average voice model* and then on low data estimate the transform for every phoneme using for example decision tree. This way we can mimic another voice. We can also combine two or more models to create a new voice.

# Miscellaneous

Arbitrary speech recognition related topics which were discussed, but did not fit into any of the previous sections.

## Triphoneme (context dependent phonemes) sharing

*Source: Young et. al. - Chapter 10.5*

To combat data sparsity and perform well on unseen triphonemes, we can cluster some states (which correspond to phonemes) together and then they can share parameters in training. For clustering, we can use a decision tree with Yes/No phonetic questions. Such questions only regard the left and right context (adjoining phonemes). Questions regarding the middle phoneme itself would be useless because we allow clustering only for the same middle phonemes.

In HTK we provide the phonetic questions and the tree is built using HHEd.

## Mixture incrementing

*Source: Young et. al. - Chapter 10.6*

To better model the training data we can increase the number of Gaussians. We select the gaussian with the highest weight, make a copy and divide both weights by two. This would result in two identical Gaussians, which is undesirable, so we add $+\theta \cdot \mu$ to the first mean and $-\theta \cdot \mu$ to the second one. HHEd uses $\theta = 0.2 \cdot \sigma$ (standard deviation.)

It is a good idea to increase the number of Gaussians by 1 or 2, then reestimate and then increment again instead of increasing it by a large number at once.

## Automatic Speech Recognition Evaluation

To evaluate ASR we can make use of the Levenshtein Distance. Usually, we consider the distance from reference to source and we thus get:

- $C$ - number of correct words
- $I$ - number of inserted words
- $D$ - number of deleted words
- $S$ - number of changed words

Word error rate (WER) can be thus defined as $\frac{I+D+C}{C+S+D} = \frac{I+D+C}{\#\text{words in reference}}$.

HResult uses word accuracy, which is computed as $\frac{C-I}{N}$ and correctness, which we get as $\frac{C}{N}$. It also computes the number of sentences with all words recognized correctly.

## Grammars

### Single-word recognition

A simple grammar, which connects the start state with all words (uniform or unigram probability) and all word with the end state (100% transition probability).

### Speech recognition

Either a full grammar description (finite state automata in the Chomsky hierarchy) or some stochastic graph, based on zerogram, unigram, bigram or trigram models.

### Mel-frequency cepstrum

*Source: https://en.wikipedia.org/wiki/Mel-frequency_cepstrum*

After we take the Fourier transform, we put it through the mel scale, so that it is relativized as to what are people sensitive to. Discrete cosine transform is almost like the Fourier transform inverse.

1. Take the Fourier transform of (a windowed excerpt of) a signal
2. Map the powers of the spectrum obtained above onto the mel scale, using triangular overlapping windows
3. Take the logs of the powers at each of the mel frequencies
4. Take the discrete cosine transform of the list of mel log powers, as if it were a signal
5. The MFCCs are the amplitudes of the resulting spectrum

## Sources


- Fink, G. A. (2014). Markov models for pattern recognition: from theory to applications. Springer Science & Business Media.
- Huang, X., Acero, A., Hon, H. W., & Foreword By-Reddy, R. (2001). Spoken language processing: A guide to theory, algorithm, and system development. Prentice hall PTR.
- Tokuda, K., Nankaku, Y., Toda, T., Zen, H., Yamagishi, J., & Oura, K. (2013). Speech synthesis based on hidden Markov models. Proceedings of the IEEE, 101(5), 1234-1252.
- Young, S., Evermann, G., Gales, M., Hain, T., Kershaw, D., Liu, X., ... & Valtchev, V. (2002). The HTK book. Cambridge university engineering department, 3, 175.