Automatic Speech Recognition
HMM/GMM Paradigm and WFST Framework

Nikolaos Flemotomos

Signal Analysis and Interpretation Laboratory
University of Southern California

Most of the content in this manuscript has been adapted from specific chapters in my thesis "Robust Acoustic Features for Distant Speech Recognition" for my Diploma in Electrical and Computer Engineering at the National Technical University of Athens (March, 2016).

Current version (translated from the Greek language and revised): March, 2019

Contents

Chapter 1

Automatic Speech Recognition

1.1 Feature Extraction

For the task of speech recognition, we should first of all find the suitable feature sets which have the ability to distinguish a part of the speeh signal from another one, while grouping similar parts together, in such a way that finally the same phonemes are grouped together and can be distinguished from all the others. Thus, when it is time to recognize a new phoneme, we hope that its features will be such that it will be successfully clustered into the right group.

Several such feature sets have been proposed, each one of which requires a specific procedure of processing the speech signal. In this Section we will mention the elementary rules behind all the relevant procedures and mainly the importance and the qualitative analysis of "feature extraction". Details on specific sets can be found in Chapter 3.

Initially, speech signal is nothing more than a longitudinal wave, which is a sequence of compressions and rarefactions propagating in the air, thanks to the elastic property of the latter [1]. To represent and store such a signal in a copmuter the first step is to record it throuhg a microphone. Microphone is a transducer which converts the acoustical signal into an electrical one, or, in other words, the acoustical sound energy into electrical energy. In the case of an ideal microphone (a non-existent machine which we are using in our analysis for simplicity), the produced electrical waveform should have the exact same characteristics as the acoustical waveform, with the compressions in the air corresponding to positive electric potentials and the rarefactions corresponding to negative potentials.

In the simplest case, the sound wave bumps into a thin metal wire called diaphragm, causing its vibration with a frequency equal to the wave frequency. The vibration of the diaphragm takes place inside a static magnetic field. But the movement of a conductor perpendicular to the magnetic field lines has as a result the flowing of electric current through the conductor. Thus, alternating current flows in the closed circuit of the microphone and the frequency is the same as the frequency of the sound wave. This current is the electrical output of the microphone.

The next step is the conversion of the analog electrical signal into a digital one through two processes; namely sampling, which is the periodic measurement of its amplitude, and quantization, which is the mapping of real values into discrete bins which can be represented by a finite number of bits (e.g. 8 or 16) [2]. In order for the sampled signal to be an accurate and unique representation of the initial signal, the sampling frequency should satisfy the Nyquist condition, according to which it should be at least double

the maximum frequency of the signal, given that the signal is band-limited [3]. Even though the speech signal is not strictly band-limited, its frequency-domain information is rapidly reduced in high frequencies, while almost the entire information in human speech is below $10kHz$, thus a sampling frequency equal to $20Khz$ is sufficient. Telephone speech is filtered and, as a result, all the information is below $4kHz$, which means that for telephone speech processing the sampling rate is usually $8kHz$. The most common and widely used sampling frequency for human speech recorded by microphones, and for the purposes of Automatic Speech Recognition, is $16kHz$.

After the aforementioned process, the speech signal is represented as a sequence of quantized samples. Subsequently, the signal is segmented into small, usually overlapped, frames of fixed length. Even though there have been approaches on using frames of duration up to $200-300msec$, traditionally the common frame length is about $10-30msec$.

The main idea behind windowing the signal is based on that: Speech is a random signal with large variations through time, without some periodicity and without satisfying the necessary conditions of stationarity in the general case. However, both its time and spectral features can be considered fixed for segments of length equal to $10-30msec$ [3]. This short-time sationarity of human speech gives the opportunity of using classic digital signal processing techiques, such as the Fourier analysis, which would be impossible to use directly on the signal before framing.

Through the desired feature extraction module, each frame is represented by a vector in $\mathbb{R}^d$. Denoting as $\mathbf{o}_i$ the vector which represents the i^{th} frame, we get the sequence $O = \mathbf{o}_1, \mathbf{o}_2, \mathbf{o}_3, \cdots$, which is called the sequence of observations. The task of Automatic Speech Recognition is now reduced to finding the most probable word sequence $\hat{W} = \hat{w}_1, \hat{w}_2, \hat{w}_3, \cdots$, given the observation sequence O [1]. Formally, we have

$$\hat{W} = \underset{W \in \mathcal{W}}{\operatorname{argmax}} P(W|O)\,, \tag{1.1}$$

where $\mathcal{W}$ is the set of possible word sequences and the estimation of the probability $P(W|O)$ is based on the acoustic and language models. In particular, according to the Bayes rule,

$$P(W|O) = \frac{p(O|W)P(W)}{P(O)}\,, \tag{1.2}$$

thus we get

$$\hat{W} = \underset{W \in \mathcal{W}}{\operatorname{argmax}}\, p(O|W)P(W)\,, \tag{1.3}$$

where $p(O|W)$ is the acoustic likelihood of O for W and $P(W)$ is the prior probability of W [4].

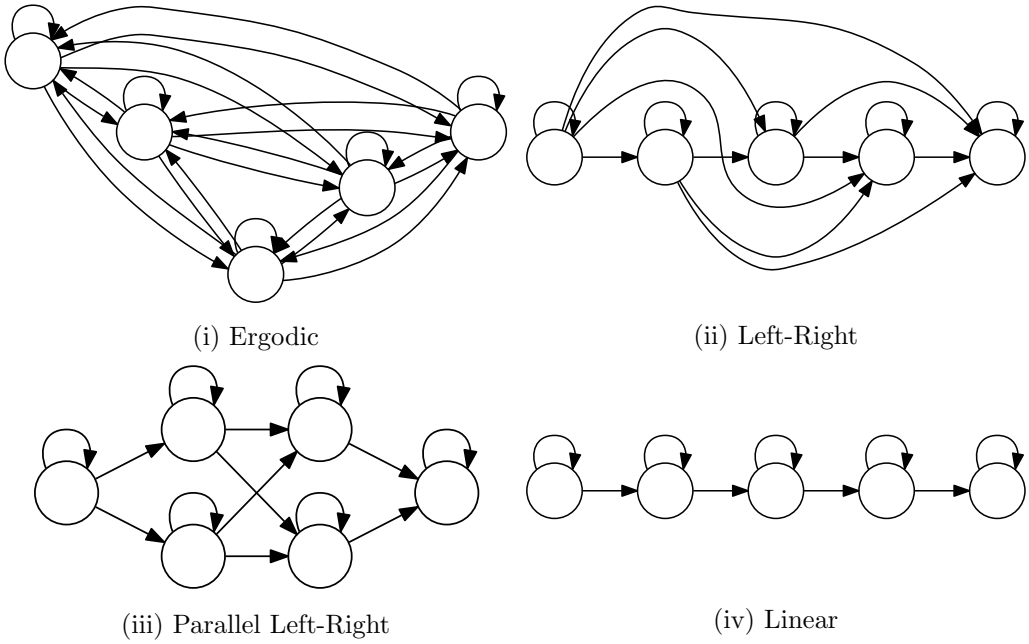
1.2 Acoustic Model

By the term "acoustic model" we refer to the statistical model which is used during speech recognition for the computation of the acoustic likelihood $p(O|W)$. If the sequence of observations O was generated by the sequence of words W, this likelihood is expected to have a large value.

[1] It is noted that there is not a one-to-one correspondence between the indices of the elements in O and the elements in W.

1.2.1 Hidden Markov Models

In practice, the most commonly used acoustic models are Hidden Markov Models (HMMs), and in particular, left-right HMMs with allowed transitions only between consecutive states or self-loops, which are also called linear HMMs [5]. The structural form of such an HMM is illustrated in Figure 1.1iv.



(i) Ergodic (ii) Left-Right

(iii) Parallel Left-Right (iv) Linear

Figure 1.1: Examples of HMM topologies. Linear HMMs are the ones which are mainly used in Automatic Speech Recognition.

Formally, any HMM is fully defined by the 6-tuple [6]

$$\lambda = \{Q, V, A, B, \pi, F\}\,, \tag{1.4}$$

where

- $Q = \{q_1, q_2, \cdots, q_N\}$ is a finite set of states,
- $V = \{\mathbf{v}_1, \mathbf{v}_2, \cdots, \mathbf{v}_M\}$ is a finite set of observations that the HMM can produce (one observation per state),
- $A = \{a_{ij}\}, i, j = 1, \cdots, N$ is the set of the transition probabilities from a state i to a state j:
$$a_{ij} = P(q_j \text{ in } t+1 | q_i \text{ in } t), \quad \sum_j a_{ij} = 1\,, \tag{1.5}$$
- $B = \{b_j(k)\}, j = 1, \cdots, N, k = 1, \cdots, M$ is the set of probabilities (or probability distributions) of producing an observation $\mathbf{v}_k$ from a state q_j:
$$b_j(k) = P(\mathbf{v}_k \text{ in } t | q_j \text{ in } t), \quad \sum_k b_j(k) = 1\,, \tag{1.6}$$

- $\pi = \{\pi_i\}, i = 1, \cdots, N$ is the set of initial state probabilities:

$$\pi_i = P(q_i \text{ in } t = 1), \quad \sum_i \pi_i = 1\,, \tag{1.7}$$

- F is a finite set of final states, subset of V.

A possible observation in a state is a d-dimensional vector, as it has been computed during the feature extraction step. Since in the general case the elements of the vector can have any value on the real axis, without any quantization, the models in use are usually continuous, thus the condition (1.6) becomes $\int_{-\infty}^{\infty} b_j(x)dx = 1$. According to the definition, any observation generated by the HMM when this is in a particular state, depends only on that state and not on previous ones. In other words, in an HMM, Markov assumption holds.

The acoustic likelihood with an HMM, that is the probability that an HMM with parameters λ generates a sequence of observations O, is computed in practice using the Viterbi algorithm, based on the most probable sequence of states Q^*. The so-called Viterbi score is given as

$$\begin{aligned} P_V &= \max_Q P(O, Q|\lambda) = P(O, Q^*|\lambda) && (1.8)\\ &= \max_{q_1, q_2, \cdots, q_T} \left\{ \pi_{q_1} b_{q_1}(\mathbf{o}_1) a_{q_1 q_2} b_{q_2}(\mathbf{o}_2) \cdots a_{q_{(T-1)} q_T} b_{q_T}(\mathbf{o}_T) \right\} \\ &= \max_i \delta_T(i)\,, && (1.9) \end{aligned}$$

where we have defined the recursive function

$$\delta_t(i) = \begin{cases} \pi_i b_i(\mathbf{o}_1) & , t = 1 \\ \max\limits_i \{\delta_{t-1}(i) a_{ij}\}\, b_j(\mathbf{o}_t) & , t = 2, 3, \cdots, T \end{cases} . \tag{1.10}$$

We can see that the Viterbi score is not an accurate estimation of the acoustic likelihood, since that would be equal to the sum of all the probabilities of the possible sequences of transitions that the HMM could follow. Thus, we would get

$$p(O|\lambda) = \sum_Q P(O, Q|\lambda)\,. \tag{1.11}$$

This computation could again be done using dynamic programming, after replacing the max by a summation in equations (1.9), (1.10), which is exactly what the so-called Forward algorithm does. However, this algorithm is not used in practice, since the final recognition result is not substantially affected [4]. More details about Viterbi algorithm will be presented in Section 1.4.

Every HMM state can be viewed as corresponding to a phoneme, or, more generally, to a Subword Unit (SU), such as a phonelike unit (PLU), a syllable, or a semi-syllable. PLUs are based on phonemes, but are not identical to them, because they are modeled based on the acoustic similarity instead of the linguistic one [6]. For problems with a relatively small vocabulary, we could use as elementary units even the words themselves, but for continuous speech with large vocabulary, this is not an option.

In practice, an SU is not represented by a singe HMM state, but by an elementary HMM with a fixed number of states (5-10 states) [6]. For instance, we could model a phoneme by a 5-state HMM, like the one illustrated in Figure 1.1iv. Self-loops are used in the specific topology, so that any SU, and any state, can have arbitrarily long duration.

Because of coarticulation, the acoustic properties of each phoneme are affected by the neighboring phonemes and are not the same in all acoustic environments. This phenomenon is due to the continuous nature of human speech and the inevitable inertia of articulators (vocal cords, tongue, lips, etc.) responsible for the speech production [7]. For that reason, it is common not to use phonemes (or PLUs) as the elementary units, but SUs taking into account the preceding and succeeding phonemes (or PLUs), that is context-dependent SUs, such as triphones and quinphones. For example, the phrase *get up* consists of the phonemes /G/ /EH/ /T/ /AH/ /P/[2], but of the triphones /(sil)G(EH)/ /(G)EH(T)/ /(EH)T(AH)/ /(T)AH(P)/ /(AH)P(sil)/.

Having chosen the fundamental subword unit to be used, and modeling each one by an elementary HMM, each word can be represented by concatenations of such elementary HMMs, while a sentence can be represented by allowing transitions between HMMs modelling different words. The knowledge of which HMMs have to be combined for the generation of a word is given by the pronunciation lexicon, which is a mapping (not one-to-one) between each word and its pronunciation (or pronunciations).

1.2.2 Gaussian Mixture Models

As has been already mentioned, the possible observations in each HMM state are vectors in the space $\mathbb{R}^d$. The most common way of modelling the probability of an observation in a state is through (d-dimensional) Gaussian Mixture Models (GMMs) [8].

A GMM is nothing more than a linear superposition of gaussian distributions, such that the probability that in state i we get the observation $\mathbf{x}$ is modeled as

$$b_i(\mathbf{x}) = \sum_{m=1}^{M_i} c_{im} \mathcal{N}(\mathbf{x}|\boldsymbol{\mu}_{im}, \boldsymbol{\Sigma}_{im}) \,, \tag{1.12}$$

where M_i is the number of components in the mixture, which may vary between the states, and the parameters c_{im} satisfy the required conditions so that the combination in (1.12) is convex:

$$c_{im} \geq 0 \; \forall m \text{ and } \sum_{m=1}^{M_i} c_{im} = 1 \,. \tag{1.13}$$

It is noted that in practice any probability distribution can be estimated with an arbitrarily good precision by a GMM, given a sufficiently large number of gaussians.

Usually, for the task of Automatic Speech Recognition, gaussians with diagonal covariance matrices are chosen. Although full matrices lead to more accurate modelling and potentially better recognition results, they are related to two serious problems [2]. First, they require a much bigger amount of data for their trainin and second, the relevant algorithms need much more running time. Under the assumption of diagonal covariance matrix, every gaussian distribution is independent with respect to any dimension, so it is

[2]The CMU Pronouncing Dictionary http://www.speech.cs.cmu.edu/cgi-bin/cmudict

computed as

$$\mathcal{N}(\mathbf{x}|\boldsymbol{\mu}_{im}, \boldsymbol{\Sigma}_{im}) = \prod_{p=1}^{d} \frac{1}{\sqrt{2\pi\sigma_{imp}^2}} \exp\left\{-\frac{(x_p - \mu_{imp})^2}{2\sigma_{imp}^2}\right\}, \tag{1.14}$$

where μ_{imp} and σ_{imp}^2 are the mean and variance, respectively, of the p-th dimension of the m-th gaussian of the mixture corresponding to the i-th HMM state.

1.2.3 Training the Acoustic Model

Combining HMMs and GMMs yields an efficient and elegant statistical modeling for the speech recognition task, which is, however, based on a series of parameters which require a suitable training procedure for their estimation. In particular, we need to train the parameters a_{ij} and π_i, appearing in equations (1.5), (1.7), as well as the parameters c_{im}, μ_{im}, Σ_{im}, appearing in equations (1.12) - (1.14). After training the GMMs, we directly get the set of probabilities $B = \{b_j(k)\}$, since this is exactly what they model. The algorithm which is used for the required training is Expectation - Maximization (EM).

Of course, we need some initial estimations and the simplest way to get them is using a technique called flat start [2]. According to this approach, we set equal to 0 the transition probabilities which we want by construction to be "inactive" in the HMM, something that guarantees they will remain 0 after the training as well, as will become obvious later. All the other transitions are initially considered to be equiprobable. For instance, for a 5-state model, like the one in Figure 1.1iv, we get the initial transition matrix

$$A = \begin{bmatrix} 0.5 & 0.5 & 0 & 0 & 0 \\ 0 & 0.5 & 0.5 & 0 & 0 \\ 0 & 0 & 0.5 & 0.5 & 0 \\ 0 & 0 & 0 & 0.5 & 0.5 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix}.$$

Obviously, for such a topology, we will also have

$$\pi_i = \begin{cases} 1 & , i = 1 \\ 0 & , i \neq 1 \end{cases}.$$

As far as the required GMM initializations are concerned, all the mean values and variances are initialized with the corresponding sample mean values and variances, as estimated on the training set.

In the HMM case, EM algorithm is known as forward-backward [6] and is based on the forward probability, as mentioned in Subsection 1.2.1, and the backward probability. The problem during HMM training is, given a sequence of observations O and a set of the possible HMM states, to estimate the unkown parameters λ such that the probability $P(O|\lambda)$ is maximized.

Forward probability is defined as the probability that an HMM with parameters λ generates at time t a sequence of observations $\mathbf{o}_1\mathbf{o}_2\cdots\mathbf{o}_t$ and is in state i:

$$\alpha_t(i) = P(\mathbf{o}_1\mathbf{o}_2\cdots\mathbf{o}_t, q_t = i|\lambda) \tag{1.15}$$

$$= \begin{cases} \pi_i b_i(\mathbf{o}_1) & , t = 1 \\ \sum_i \{\alpha_{t-1}(i)a_{ij}\}\, b_j(\mathbf{o}_t) & , t = 2, 3, \cdots, T \end{cases}. \tag{1.16}$$

As we have already seen, the last formula can be used for the precise estimation of the acoustic likelihood $p(O|\lambda)$ of an HMM, since

$$\begin{aligned} p(O|\lambda) &= \sum_{Q} P(O,Q|\lambda) \\ &= \sum_{q_1,q_2,\cdots,q_T} \left\{ \pi_{q_1} b_{q_1}(\mathbf{o}_1) a_{q_1 q_2} b_{q_2}(\mathbf{o}_2) \cdots a_{q_{(T-1)} q_T} b_{q_T}(\mathbf{o}_T) \right\} \\ &= \sum_{i} \alpha_T(i)\,. \end{aligned} \tag{1.17}$$

Similarly, backward probability is defined as the probability that an HMM with parameters λ being in state i for time t generates a sequence of observations $\mathbf{o}_{t+1}\mathbf{o}_{t+2}\cdots\mathbf{o}_T$:

$$\beta_t(i) = P(\mathbf{o}_{t+1}\mathbf{o}_{t+2}\cdots\mathbf{o}_T|q_t = i,\lambda) \tag{1.18}$$

$$= \begin{cases} 1 & ,t = T \\ \sum_j \{a_{ij} b_j(\mathbf{o}_{t+1}) \beta_{t+1}(j)\} & ,t = T-1, T-2, \cdots, 1 \end{cases} . \tag{1.19}$$

We, now, define as $\xi_t(i,j)$ the probability that the HMM is in state i for time t and in state j for time $t+1$, and as $\gamma_t(i)$ the probability that the HMM is at state i for time t. It can be proved [6] that

$$\xi_t(i,j) = \frac{\alpha_t(i) a_{ij} b_j(\mathbf{o}_{t+1}) \beta_{t+1}(j)}{\sum_i \sum_j \alpha_t(i) a_{ij} b_j(\mathbf{o}_{t+1}) \beta_{t+1}(j)}\,, \tag{1.20}$$

$$\gamma_t(i) = \sum_j \xi_t(i,j) \tag{1.21}$$

$$= \frac{\alpha_t(i)\beta_t(i)}{\sum_i \alpha_t(i)\beta_t(i)}\,. \tag{1.22}$$

Equations (1.20) and (1.21) consist the expectation step in the EM algorithm. Based on those, maximization is the step where essentially the model parameters are estimated according to the Maximum Likelihood (ML) criterion:

$$\begin{aligned} \hat{\pi}_i &= \text{expected number of times the HMM is in state } i \text{ for time } t = 1 \\ &= \gamma_1(i)\,, \end{aligned} \tag{1.23}$$

$$\begin{aligned} \hat{a}_{ij} &= \frac{\text{expected number of transitions from state } i \text{ to state } j}{\text{expected number of transitions from state } i} \\ &= \frac{\sum_{t=1}^{T-1} \xi_t(i,j)}{\sum_{t=1}^{T-1} \gamma_t(i)}\,. \end{aligned} \tag{1.24}$$

For completeness, but also to make easier the generalization in the continuous case, we give the formula to compute the parameters $b_j(k)$ for discrete probability distributions:

$$\begin{aligned}\hat{b}_j(k) &= \frac{\text{expected number of times at state } j \text{ with observation } \mathbf{v}_k}{\text{expected number of times at state } j} \\ &= \frac{\sum\limits_{t=1 \text{ s.t. } \mathbf{o}_t=\mathbf{v}_k}^{T} \gamma_t(j)}{\sum\limits_{t=1}^{T} \gamma_t(j)} .\end{aligned} \tag{1.25}$$

For GMMs, which are used in practice, we generalize formula so that it expresses the probability that HMM is at state i for time t, with m-th gaussian giving the observation $\mathbf{o}_t$:

$$\gamma_t(i,m) = \frac{\alpha_t(i)\beta_t(i)}{\sum\limits_{i} \alpha_t(i)\beta_t(i)} \cdot \frac{c_{im}\mathcal{N}(\mathbf{x}|\boldsymbol{\mu}_{im}, \boldsymbol{\Sigma}_{im})}{\sum\limits_{m=1}^{M_i} c_{im}\mathcal{N}(\mathbf{x}|\boldsymbol{\mu}_{im}, \boldsymbol{\Sigma}_{im})} . \tag{1.26}$$

We note that equation (1.22) does not hold only for the case of discrete distributions, but also in the continuous case, when, for example, we choose to model the probability by only one gaussian.

The parameters are now updated according to the following formulas (maximization steps):

$$\begin{aligned}\hat{c}_{im} &= \frac{\text{expected number of times at state } j \text{ with the } m\text{-th gaussian}}{\text{expected number of times at state } j} \\ &= \frac{\sum\limits_{t=1}^{T} \gamma_t(i,m)}{\sum\limits_{t=1}^{T}\sum\limits_{m=1}^{M_i} \gamma_t(i,m)} \triangleq \frac{N_{i,m}}{\sum\limits_{m=1}^{M_i} N_{i,m}} .\end{aligned} \tag{1.27}$$

Using the maximum likelihood criterion [8], we also get

$$\hat{\boldsymbol{\mu}}_{im} = \frac{1}{N_{i,m}} \sum_{t=1}^{T} \gamma_t(i,m)\mathbf{o}_t = \frac{\sum\limits_{t=1}^{T} \gamma_t(i,m)\mathbf{o}_t}{\sum\limits_{t=1}^{T} \gamma_t(i,m)} , \tag{1.28}$$

$$\hat{\boldsymbol{\Sigma}}_{im} = \frac{1}{N_{i,m}} \sum_{t=1}^{T} \gamma_t(i,m)(\mathbf{o}_t - \boldsymbol{\mu}_{im})(\mathbf{o}_t - \boldsymbol{\mu}_{im})^T = \frac{\sum\limits_{t=1}^{T} \gamma_t(i,m)(\mathbf{o}_t - \boldsymbol{\mu}_{im})(\mathbf{o}_t - \boldsymbol{\mu}_{im})^T}{\sum\limits_{t=1}^{T} \gamma_t(i,m)} . \tag{1.29}$$

EM algorithm is iterative with expectation and maximization steps coming one after the other. It can be proved [8] that every time the parameters of the model are updated from λ_{old} to λ_{new}

$$P(O|\lambda_{new}) \geq P(O|\lambda_{old}) \,, \tag{1.30}$$

so it is guaranteed that the algorithm will converge at some local (not necessarily global) maximum, after enough iterations.

1.2.4 Tied States and Decision Trees

The use of triphones as SUs, despite the advantageous effects it has to the final performance of the recognizer, is connected to a main "disadvantage": there are too many triphones. Assuming a language with 40 phonemes - a very realistic assumption - we have $40^3 = 64000$ triphones. Thus, apart from the complexity they insert to the problem, there is always the danger that some triphones are so rare that there are not enough representatives in the training set and the algorithm cannot accurately estimate the parameters related to the corresponding HMMs. It is even possible that there are not representatives at all in the training set, but they are encountered in the test set, that is in the word sequences to be recognized.

In order to aleviate the aforementioned problem, speech recognition systems in practice use the notion of parameter sharing between HMMs and HMM states [9]. A first idea is the usage of a common set of a specific number of gaussian distributions, which are used by all the states of all the HMMs and are combined with appropriate weights, so that each state is modeled by a suitable GMM [10]. In theory, parameter sharing can happen at any level. For instance, parameters may be shared between states of the same SU (usually triphone) or different SUs. Additionally, the entire state (that is the GMM) may be shared, or only the individual gaussians, or only specific parameters (e.g. the variances are shared but the means are different). Here, we will present a technique which is found in practice in modern systems, the tied-state HMMs, using decision trees [11].

A tied-state HMM aims at providing guarantees that there are enough training data so that all the required parameters are accurately estimated, but, at the same time, all the context-dependent differences between the phonemes are preserved. With this method, it is possible even to estimate parameters for HMMs corresponding to triphones which have never been encountered during training.

To describe the process, there is the initial assumption that all the triphones are modeled by HMMs with the same number of states N. For each one of those states and for every phoneme, a decision tree is constructed; thus, there will be totally $N \cdot P$ decision trees, where P is the number of phonemes in the language. All the triphones of the same central phoneme share the same transition matrix A, so the decision trees are used to cluster the probability distributions of the observations of HMM states.

Initially, all the states to be clustered by one decision tree (for example the second state of the HMMs of all the triphones having x as their central phoneme) are put at the root of the tree and are considered to be tied, that is they share the same parameters, while the corresponding probability distribution is modelled by only one gaussian. Let S be the set of those states and F the corresponding training frames. Then, we calculate the log likelihood $L(S)$ that F are produced from S and we choose the question which will split the root into two nodes in a way that will lead to the maximum increase of the log likelihood. The same process is repeated iteratively until the log likelihood reaches a minimum threshold,

while not too many leaves have been generated, so that it is guaranteed that all the leaves are related to a sufficient number of training samples. All the questions are of the form "*Is the left or right phoneme member of the set X?*". An example of such a decision tree is given in Figure 1.2.

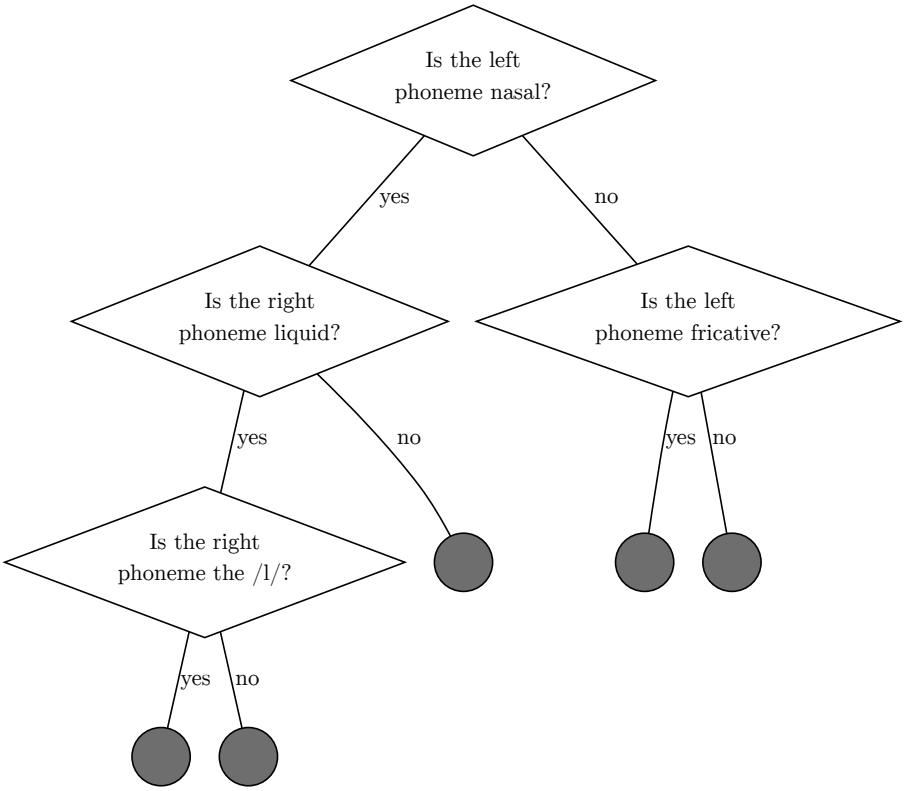


Figure 1.2: Example of decision tree to cluster triphones. The input is the x-th state of the HMMs of all the triphones with the same central phoneme. In this example, 5 clusters of triphones are generated, where the states in each cluster are tied, that is they share the same parameters.

The last stage is to split the gaussian distributions. As mentioned, each state is initially modeled by a single gaussian. After the construction of the decision trees, the parameters of the probabililty distributions are iteratively retrained, splitting in every iteration one or more gaussian distributions until we reach the desired number of gaussians from which the GMMs will be composed.

The total number of states, that is the total number of all the leaves of all the decision trees, is usually predefined and is usually in the range of a few thousand to a few tens of thousands, with each state corresponding to a GMM of 16 to 64 gaussians. Those numbers are empirical and depend on the volume of the available training data.

According to the technique described, triphone models are actually never trained or stored. While, during decoding, a specific triphone model is needed, the corresponding HMM is composed based on the appropriate decision trees. That way, the required models for all the triphones can be composed, even for those which the system did not see during training.

1.2.5 Forced Alignment

The training of the acoustic model is based on the available recordings and the corresponding transcripts. However, in spoken language there are some elements which, in general, are not available int the written text of transcripts. The main such elements are the possible silences between the words and the different pronunciation that every speaker uses to utter a specific word.

To aleviate those problems, and since the transcrpits are usually available at the word level and not at the phoneme level (with the exception of a few databeses), a phonetic alignment step is necessary. Even though this is by itself a field of research and different approaches have been proposed [12], for the puproses of speech recognition the technique of forced alignment, through the Viterbi algorithm, is commonly used.

First of all, we assume that there is a given dictionary which contains all the possible pronunciations of the words and that silences optionally exist after every word, so that pauses can be modeled between consecutive spoken words [2]. It is noted that usually the silence phones are modeled by a more complex topology compared to the rest. For example, in the Kaldi speech recognition toolkit[3], silences are modeled by default by a 5-state left-rigth HMM where transitions between various states are valid, as in Figure 1.1ii. Additionally, the spectral properties of silence are not affected by neighboring phonems, so context-free, monophone models are typically trained.

Based on those hypotheses, we construct, for each sentence, HMMs with parallel paths for every possible pronunciation of a word and with all the additional silence phones, as shown in the example of Figure 1.3. Viterbi algorithm can now be applied on those HMMs. Since thei algorithm can find the most probable state sequence on the HMM it is applied to, the pronunciation used, as well as the time boundaries between the words, can be found from that sequence. The term "forced" alignment stems from the fact that Viterbi algorithm is forced to find the optimal alignment under specific restrictions imposed by the HMM structure.

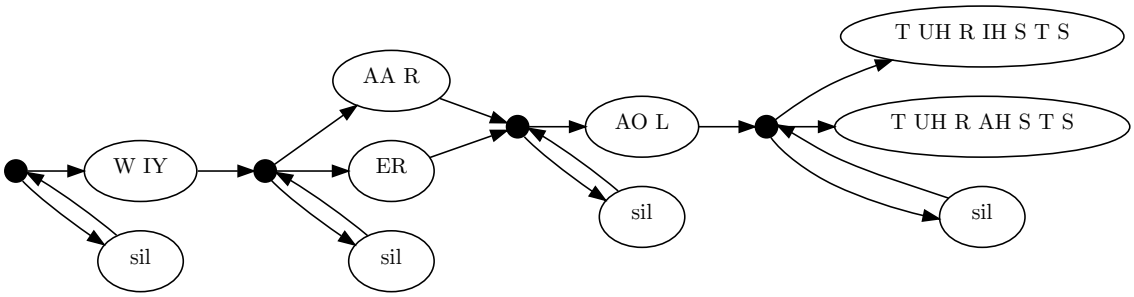


Figure 1.3: Simplified HMM modeling the phrase *we are all tourists*. There are two alternative pronunciations for two words of the phrase, and pauses may occur between any two words. Every ellipse here corresponds to a smaller constituent HMM.

1.3 Language Model

The performance of a speech recognizer is drastically improved when a reliable language model is used. With that, ungrammatical and non probable sentences can be rejected,

[3] http://kaldi-asr.org

leading to a significant reduction of errors [4].

The most widely used models for Large Vocabulary Continuous Speech Recognition (LVCSR) are n-grams, which are going to be presented in detail. For applications where we know in advance a finite, limited set of valid phrases, such as the spoken command recognition [13], Finite State Grammars (FSGs), which are defined a priori by the designer of the system, can be used. An example is shown in Figure 1.4.

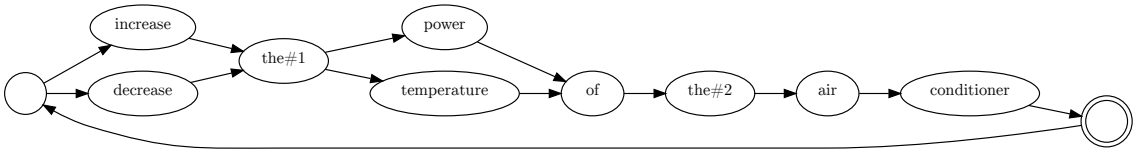


Figure 1.4: Example of a Finite State Grammar. the#1 and the#2 correspond to the same word but different states.

1.3.1 Statistical Language Modelling with n-gram Models

By the term "n-gram" we refer to a subsequence of n similar elementss of a sequence. We usually refer to n-grams of words, but, depending on the application, we may have n-grams of characters, of phonemes, etc. One of the first times where the term n-gram was used is in Shannon's work on Information Theory [14], who supported that a synthetic text which is constructed based on n-gram modeling becomes more intelligible as the order n of the model is increased.

Denoting a sequence of M words as $W = w_1, w_2, \cdots, w_M \triangleq w_1^M$, we know that

$$P(W) = P(w_1)P(w_2|w_1)\cdots P\left(w_M|w_1^{M-1}\right) = P(w_1)\prod_{m=2}^{M} P\left(w_m|w_1^{m-1}\right) . \tag{1.31}$$

According to an n-gram model, instead of computing the probability of a word given its entire history, this history is approximated only by the n previous words:

$$P(W) \approx \prod_{m=1}^{M} P\left(w_m|w_{m-n+1}^{m-1}\right) = \prod_{m=1}^{M} P\left(w_m|w_{m-n+1}, w_{m-n+2}, \cdots, w_{m-1}\right) . \tag{1.32}$$

This model is equivalent to a Markov model of order $n-1$, since every state depends only on the history up to $n-1$ "time levels" back.

In the notation used in (1.32), in order for negative indices to make sense, that is in order to model the cntext of the first words, special characters (`<s>`) need to be inserted at the beginning of each sentence of the corpus used. For instance, if $m = 1$, $n = 2$ (bigram model), we would get $P(w_1|w_0) = P(w_1|\texttt{<s>})$. Furthermore, special charachters are inserted at the end of each sentence as well (`<\s>`). For example, if the available corpus contained the sentence *I play*, to train a bigram model, we should estimate the probabilities P(I—`<s>`), P(play—I) and P(`<\s>`—play).

An n-gram model is trained on the available text data, which is called training corpus. For Automatic Speech Recognition, this corpus is composed by the transcriptions of the speech data. The required probabilities can be simply estimated using Maximum

Likelihood Estimation (MLE):

$$P\left(w_m|w_{m-n+1}^{m-1}\right) = \frac{C\left(w_{m-n+1}^{n}\right)}{C\left(w_{m-n+1}^{n-1}\right)}\,, \tag{1.33}$$

where $C(s)$ is the number of times that the sequence s is encountered in the corpus.

As the order of the model gets bigger, the results are expected to become better and better. However, a very "powerful" language model can be very biased on the the training set without having good generalization performance. The most common models are 3-grams, with the big, modern systems using up to 5-grams.

The intuitive improvement of a recognition system as the order of the model gets bigger can be formally expressed through the notion of perplexity [2]. The perplexity (PP) of a sequence W with M words is defined as

$$PP(W) = P\left(W\right)^{-M} = \sqrt[M]{\frac{1}{P(w_1, w_2, \cdots, w_M)}}\,. \tag{1.34}$$

Combining equations (1.34) and (1.32), we have that, using an n-gram model, the perplexity is

$$PP(W) = \sqrt[M]{\prod_{m=1}^{M} \frac{1}{P\left(w_m|w_{m-n+1}^{m-1}\right)}}\,. \tag{1.35}$$

As we can see, minimizing the perplexity is equivalent to maximizing the probability of sequence W, according to the model used.

Here, it is worth noting a practical issue related to n-gram models and has to do with the relationship between closed and open vocabulary. In the second case, it is possible that the training set contains words which do not exist in the vocabulary used (for example, there are not corresponding entries in the pronunciation dictionary, so we do not know how to map them into a phoneme sequence). Then, a text normalization step needs to precede the modeling, where all those words, known as OOV (Out-Of-Vocabulary) words, are substituted by a common, predefined "word", which is usualy the symbol `<UNK>`. For the purposes of n-gram modeling, this "word" is treated like any other word of the training corpus.

1.3.2 Smoothing Using the Witten-Bell Method

One obvious problem when using MLE according to equation (1.33) is that some n-grams may appear very rarely or not at all in the training corpus, without that necessarily meaining that those n-grams cannot be encountered during testing. Therefore, we should not give them a zero (or almost zero) probability. The process during which a probability mass is subtracted by the very frequent sequences (discounting) and is shared between sequences with zero or very few occurencies (backoff) is called smoothing. In this Subsection, we will describe one of the various methods proposed for discounting, known as Witten-Bell method [15, 16].

The definition of the new probability of a specific n-gram is recursive:

$$P^*\left(w_m|w_{m-n+1}^{m-1}\right) = \lambda_{w_{m-n+1}^{m-1}} P\left(w_m|w_{m-n+1}^{m-1}\right) + \left(1 - \lambda_{w_{m-n+1}^{m-1}}\right) P^*\left(w_m|w_{m-n+2}^{m-1}\right)\,, \tag{1.36}$$

where the probability $P\left(w_m|w_{m-n+1}^{m-1}\right)$ is computed according to (1.33). To estimate the parameters $\lambda_{w_{m-n+1}^{m-1}}$ we define the auxiliary function $N_{1+}\left(w_{m-n+1}^{m-1}\bullet\right)$ as the number of different n-grams which appear in the training set and start by the $(n-1)$-gram w_{m-n+1}^{m-1}:

$$N_{1+}\left(w_{m-n+1}^{m-1}\bullet\right) = \left|\left\{w_m : C\left(w_{m-n+1}^{m-1}w_m\right) > 0\right\}\right| , \tag{1.37}$$

where $|\cdot|$ denotes the cardinality of a set. We, now, want to satisfy

$$1 - \lambda_{w_{m-n+1}^{m-1}} = \frac{N_{1+}\left(w_{m-n+1}^{m-1}\bullet\right)}{N_{1+}\left(w_{m-n+1}^{m-1}\bullet\right) + \sum_{w_m} C\left(w_{m-n+1}^{m}\right)} . \tag{1.38}$$

Using the equations (1.36), (1.33), and (1.38), we get the final equation for computing the occurence probabilities of n-grams, according to the Witten-Bell method:

$$P^*\left(w_m|w_{m-n+1}^{m-1}\right) = \frac{C\left(w_{m-n+1}^{m}\right) + N_{1+}\left(w_{m-n+1}^{m-1}\bullet\right) P^*\left(w_m|w_{m-n+2}^{m-1}\right)}{\sum_{w_m} C\left(w_{m-n+1}^{m}\right) + N_{1+}\left(w_{m-n+1}^{m-1}\bullet\right)} . \tag{1.39}$$

1.4 Search and Decoding

As already mentioned, what we want in Automatic Speech Recognition, according to the framework we have introduced, is to find the most probable word sequence $\hat{W}$ given the observation sequence O, as shown in (1.3), where an observation is a feature vector extracted by the signal.

In practice, during the process described, the acoustic likelihood is underestimated, since during the construction of the acoustic model, where a GMM simply estimates the occurence probability of an observation given a state, the context is not taken into account [2]. Additionally, there is a significant difference between the dynamic ranges resulting from the acoustic and the language model. For that reason, an extra parameter, known as Language Model Scaling Factor (LMSF), which reduces the effect of the language model, is introduced:

$$\hat{W} = \underset{W\in\mathcal{W}}{\operatorname{argmax}}\, p(O|W)P(W)^{LMSF} . \tag{1.40}$$

Of course, since the probabilities take real values less than one, to get the desired result, we need $LMSF > 1$.

For reasons related to numerical stability computational complexity, we are usually working in the logarithmic field[4]. So

$$\hat{W} = \underset{W\in\mathcal{W}}{\operatorname{argmax}}\, \{\log p(O|W) + LMSF \cdot \log P(W)\} . \tag{1.41}$$

Under this viewpoint LMSF is just a weighting factor of the language model, when the corresponding weighting factor of the acoustic model is fixed and equal to one [17].

The language model, however, provides a way of modeling the transition probability between consecutive words and thus inserts a penalty whenever new words are inserted. So, reducing the effect of the language model, we are actually decreasing the insertion probabilities or, equivalently, we increase the word insertion penalty. In other words, the recognizer shows a preference towards a few big words rather than many smaller ones

[4] Unless stated otherwise, we are going to use the natural logarithm (base-e).

[2]. To balance this negative effect of LMSF, a separate Word Insertion Penalty (WIP) is introduced:

$$\hat{W} = \underset{W \in \mathcal{W}}{\operatorname{argmax}}\, p(O|W) P(W)^{LMSF} WIP^{N(W)} \tag{1.42}$$

$$= \underset{W \in \mathcal{W}}{\operatorname{argmax}} \left\{ \log p(O|W) + LMSF \cdot \log P(W) + N(W) \cdot \log WIP \right\} , \tag{1.43}$$

where $N(W)$ is the number of words in the sequence W.

Finding the most probable word $\hat{W}$ is the job of the decoder. To make the entire process more easily understandable, let's consider a simple example. Let the working vocabulary be very limited and contain only the words *yes* and *no*. Speech is continuous and an unlimited number of those two words can be uttered consecutively. According to the pronunciation dictionary, every word has a unique phonetic representation:

$$\text{yes} \rightarrow \texttt{Y EH S}$$
$$\text{no} \rightarrow \texttt{N OW}$$

Let every phoneme be modelled by a three-state HMM with the topology shown in Figure 1.1iv. Furthermore, let a bigram model which gives the transition probabilities between consecutive words. According to what has been presented up to that point, a graph is constructed, which is called search network and takes the form of Figure 1.5, where, for simplicity, the states modeling the potential silence between words have been omitted.

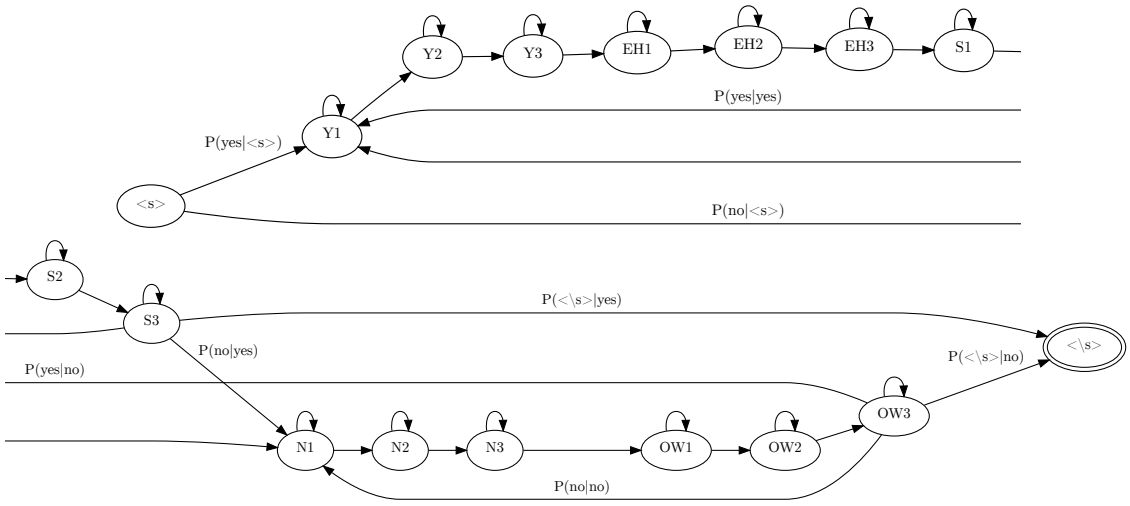


Figure 1.5: Example of a search network for decoding. The vocabulary contains only the words *yes* and *no*, evry phoneme is modeled by a 3-state HMM and a bigram model is used. For better visuzlization, the network is split into two parts.

Inside this search network, the decoder has to find the path which, given the observation sequence, is the most probable. To achieve that, decoders are based on the dynamic Viterbi algorithm. Assuming the entire search network as a single HMM with parameters λ and a set Q of N states, Viterbi algorithm Viterbi looks for the optimal state sequence Q^* which will give the Viterbi score of equation (1.8) and works as follows [6]:

- Initialization

$$\begin{aligned} \delta_1(i) &= \pi_i b_i(\mathbf{o}_1) && ,1 \leq i \leq N \\ \psi_1(i) &= 0 && ,1 \leq i \leq N \end{aligned}$$

- Recursion

$$\begin{aligned} \delta_t(j) &= \max_{1 \leq i \leq N} \left\{ \delta_{t-1}(i) a_{ij} \right\} b_j(\mathbf{o}_t) && ,2 \leq t \leq T \,, 1 \leq j \leq N \\ \psi_t(j) &= \operatorname*{argmax}_{1 \leq i \leq N} \left\{ \delta_{t-1}(i) a_{ij} \right\} && ,2 \leq t \leq T \,, 1 \leq j \leq N \end{aligned}$$

- Termination

$$\begin{aligned} P_V &= \max_{1 \leq i \leq N} \left\{ \delta_T(i) \right\} \\ q_T^* &= \operatorname*{argmax}_{1 \leq i \leq N} \left\{ \delta_T(i) \right\} \end{aligned}$$

- Backtracking

$$q_t^* = \psi_{t+1} \left(q_{t+1}^* \right), t = T-1, T-2, \cdots, 1$$

By T we denote the last frame of the last word of the sequence to be decoded.

The algorithm is based on the recursive function δ which is defined as

$$\delta_t(i) = \max_{q_1, q_2, \cdots, q_{t-1}} P\left(q_1, q_2, \cdots, q_{t-1}, q_t = i, \mathbf{o}_1 \mathbf{o}_2 \cdots \mathbf{o}_t | \lambda\right) \tag{1.44}$$

and on keeping back pointers through the function ψ in order for the final backtracking to be possible. According to the principles of dynamic programming, if the optimal move is taken at every step, then the final path will be the optimal one. It is noted that, as usually, the calculations are done in the logarithmic field, where, equivalently, we use the function

$$\tilde{\delta}_t(j) = \log \delta_t(j) = \max_{1 \leq i \leq N} \left\{ \tilde{\delta}_{t-1}(i) + \tilde{a}_{ij} \right\} + \tilde{b}_j(\mathbf{o}_t) \,. \tag{1.45}$$

Viterbi algorithm - which is not guaranteed to give the optimal decoding result - has computational complexity $O(N^2 T)$, which makes it much faster than the exponential time required to examine all the possible word sequences in order to choose the most suitable according to the available observations. However, as shown even by the toy example in Figure 1.5, the number of states N can be very large in practical applications. Thus, in order to further reduce the computational complexity, we often use a variation of the Viterbi algorithm that utilizes the idea of beam searching, which is why it is called Viterbi Beam Search (or also Time Synchronous Viterbi Beam Search) [4].

According to that method, the Viterbi score, say $L(t)$, of the state sequence which would be the optimal one if decoding terminated at time step t is computed for every t. This value is used to prune all the paths with score less than $\theta \cdot L(t)$, where θ is a constant between 0 and 1 called beam width. Since we are working in the logarighmic field, the threshold for pruning is $(\log L(t) - \eta)$, where $\eta = -\log \theta$. Alternatively, we can define in advance a fixed number K with the best (most probable) states to be kept at every step, or even combine those two approaches. In any case, it is necessary to store at every time

step the ative states, that is the states which have not been pruned yet, in queues. The beam width is a tradeoff decision between the desired accuracy of the recognition and the small decoding time. As the beam gets narrower, the decoding becomes faster, but, at the same time, the errors due to pruning are more probable, so the recognition accuracy is decreased.

Often, we don't just need the optimal sequence, but a set of say N optimal sequences (N-best search) [4]. For instance, during a multi-pass decoding, after a set of hypotheses has been constructed, the final result is found after rescoring. So, during the first decoding pass we may use a simple language model (e.g. bigram or trigram) and subsequently rescore the network based on a more complex model. Furthermore, during rescoring, different combinations for the values of LMSF andWIP may be used, which have probably not been taken into account during the first pass.

The representation of those hypotheses is done by a word lattice, which is in fact a directed acyclic graph, where each edge corresponds to a probable word, together with its score, and every node corresponds to the word boudaries and contains the relevant time information for forced alignment. This lattice can be easily constructed through a variation of the Viterbi algorithm. At every time step we need to keep a set of back pointers from active states. That way, at the last step of the algorithm, multiple probable sequences can be generated through backtracking. Each one of those sequences is a path in the word lattice.

1.5 Recognition Evaluation

A speech recognition system is trained on a specific training set and evaluated on a test set, while there is probably an additional validation set for the optimization of certain parameters. The most widely used metric for the avaluation of Automatic Speech Recognition is known as Word Error Rate (WER) and is given by the formula (1.46).

$$\mathrm{WER} = \frac{w_i \cdot \#\text{insertions} + w_s \cdot \#\text{substitutions} + w_d \cdot \#\text{deletions}}{\text{total number of words in transcripts}} \tag{1.46}$$

Setting all the weights in (1.46) equal to 1, which is the most common approach, we get

$$\mathrm{WER} = \frac{\#\text{insertions} + \#\text{substitutions} + \#\text{deletions}}{\text{total number of words in transcripts}}\,. \tag{1.47}$$

The total number of insertions, substitutions, and deletions is such that the Levenshtein distance between the constructed text and the given transcripts is minimized [2].

Alternatively, we can use the equivalent metric of Word Accuracy (WACC) [4], defined as

$$\mathrm{WACC} = 100\% - \mathrm{WER}\,. \tag{1.48}$$

Of course, for a successful recognition, we need as high WACC or as low WER as possible.

Chapter 2

Weighted Finite-State Transducers

2.1 Main Definitions

The Weighted Finite-State Transducers (WFSTs) are the more general case of Finite-State Automata or simply Finite Automata (FA). In its simplest form, a FA is a Finite-State Acceptor (FSA), which is formally defined as a 5-tuple $(Q, \Sigma, \delta, q_0, F)$ [18], where

- Q is a finite set of states,
- Σ is a finite set of symbols, called the alphabet,
- $\delta : Q \times \Sigma \to Q$ is the transition function which, based on the current state and the current symbol, determines the next state,
- $q_0 \in Q$ is the initial state, and
- $F \subseteq Q$ is a set of final states, or accept states.

In place of the function δ, we can instead define a multi-set of valid transitions $E \subseteq Q \times \Sigma \times Q$ and similarly, the initial state can be generalized to a set $I \subseteq Q$ of initial states. Moreover, the above definition does not use the empty symbol ϵ, which means that any transition requires a spesific symbol from the alphabet. Additionally, since δ is presented as a function from $(Q \times \Sigma)$ to Q, it is assumed that we refer to a Deterministic FA (DFA). If more than one transitions from the same state and with the same symbol are valid, then we refer to Non-deterministic FA (NFA). However, in the case of FSAs, it can be proved that for any NFA there is an equivalent DFA, but also that for any ϵ-NFA (where transitions without any symbol from the alphabet) there is an equivalent NFA [18].

Job of an FSA, as its name suggests, is to accept or reject an input string which is comprised of symbols belonging in the alphabet Σ. The set of strings that the FSA accepts, that is the set of strings for which there are allowed transitions that lead from an initial state to a final state, consist the language that the FSA recognizes. Any language that can be represented by an FSA is called a regular language.

On the other hand, the job of a Finite-State Transducer (FST) is to transform a representation into another one, by taking as input a string and producing, instead of a binary acceptance/rejection output, a new string [2]. Any such relation that can be represented by an FST is called a rational relation. It is noted that an acceptor can be viewed as a special case of a transducer where the output string is identical to the input

string. If we further introduce the notion of the weights, that is if each transition form a state to another is connected to some weight, which can be related to the transition probability or some cost that the particular transition incurs, then we get the WFSTs (and similarly the Weighted Finite-State Acceptors (WFSAs)). Any relation that can be represented by a WFST is called rational power series.

Before we give the formal definition of a WFSA or WFST, we should define the abstract algebraic structures of monoid and semiring [19].

A monoid, denoted as $< \mathbb{M}, \circ, \bar{1} >$, consists of a set $\mathbb{M}$, an associative binary operation $\circ$ on $\mathbb{M}$ and an identity element $\bar{1}$ such that $\bar{1} \circ a = a \circ \bar{1} = a \; \forall a \in \mathbb{M}$. If also it holds that $a \circ b = b \circ a \; \forall a \in \mathbb{M}, \forall b \in \mathbb{M}$ then the monoid is called commutative.

A semiring, denoted as $< \mathbb{A}, \oplus, \otimes, \bar{0}, \bar{1} >$, consists of a set $\mathbb{A}$ with two binary operations $\oplus$ and $\otimes$ and two constants $\bar{0}$ and $\bar{1}$, such that the following axioms are satisfied:

(i) the $< \mathbb{A}, \oplus, \bar{0} >$ is a commutative monoid,

(ii) the $< \mathbb{A}, \otimes, \bar{1} >$ is a monoid,

(iii) the distributive property holds:
$a \otimes (b \oplus c) = (a \otimes b) \oplus (a \otimes c)$ and $(a \oplus b) \otimes c = (a \otimes c) \oplus (b \otimes c) \; \forall a \in \mathbb{A}, \forall b \in \mathbb{A}, \forall c \in \mathbb{A}$,

(iv) $\bar{0} \otimes a = a \otimes \bar{0} = \bar{0} \; \forall a \in \mathbb{A}$

A WFSA on a set $\mathbb{W}$ of a semiring is defined as the 7-tuple $(Q, \Sigma, I, F, E, \lambda, \rho)$ [20, 4], where

- Q is a finite set of states,
- Σ is a finite set of symbols, called the alphabet,
- $I \subseteq Q$ is a set of initial states,
- $F \subseteq Q$ is a set of final states,
- $E \subseteq Q \times (\Sigma \cup \{\epsilon\}) \times \mathbb{W} \times Q$ is a finite multi-set of valid transitions,
- $\lambda : I \to \mathbb{W}$ is a function giving a weight to each initial state, and
- $\rho : F \to \mathbb{W}$ is a function giving a weight to each final state.

A path π is a sequence of finite (say n) successive transitions $t_0, t_1, \cdots, t_n$, where $t_i = (p(t_i), l(t_i), w(t_i), n(t_i)) \in E \; , i = 1, 2, \cdots, n$ and $p(t_{i+1}) = n(t_i)$. If $p(t_0) \in I$ and $n(t_n) \in F$, then we say that the WFSA accepts the string $l(t_0), l(t_1), \cdots, l(t_n)$ with a cost

$$w(\pi) = \lambda(p(t_0)) \otimes w(t_0) \otimes w(t_1) \otimes \cdots \otimes w(t_n) \otimes \rho(n(t_n)) \, . \tag{2.1}$$

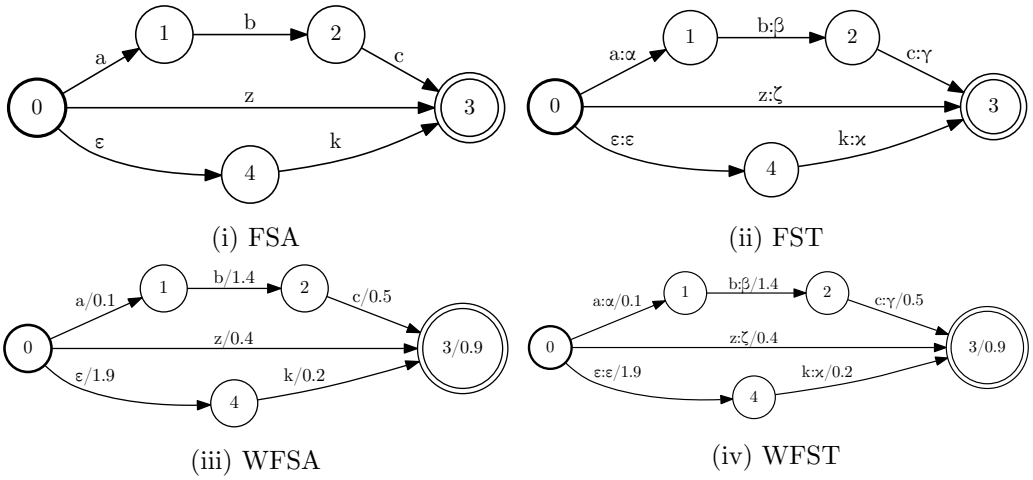
Therefore, a WFSA can be viewed as a mapping between strings and weights. It is noted that HMMs can be viewed as a special case of WFSAs [20].

Similarly, a WFST on the set $\mathbb{W}$ of a semiring is defined in the more general case as the 8-tuple $(Q, \Sigma, \Delta, I, F, E, \lambda, \rho)$ [20, 4], where

- Q is a finite set of states,
- Σ is a finite set of labels, called the input alphabet,

- Δ s a finite set of labels, called the output alphabet,
- $I \subseteq Q$ is a set of initial states,
- $F \subseteq Q$ is a set of final states,
- $E \subseteq Q \times (\Sigma \cup \{\epsilon\}) \times (\Delta \cup \{\epsilon\}) \times \mathbb{W} \times Q$ is a finite multi-set of valid transitions,
- $\lambda : I \to \mathbb{W}$ is a function giving a weight to each initial state[1], and
- $\rho : F \to \mathbb{W}$ is a function giving a weight to each final state.

FA are represented as directed graphs, where each state is denoted by a circular node, while the final states are denoted by double border circular nodes. Each node is labeled with a unique number. The edges of the graph represent the transitions between states. In WFSTs each edge is labeled as $l_i(t) : l_o(t)/w(t)$, where $l_i(t)$ is the input label, $l_o(t)$ is the output label and $w(t)$ is the weight. In case $w(t)$ does not appear, it is assumed to be equal to $\bar{1}$, while when all the edges are labeled as $l(t)/w(t)$, we have a WFSA. The initial and final nodes are also labeled with their weights (if it is not equal to $\bar{1}$). Various FA examples are shown in Figure 2.1.



(i) FSA (ii) FST

(iii) WFSA (iv) WFST

Figure 2.1: Various Finite Automata examples. (i), (ii) and (iii) can be viewed as special cases of a WFST.

Often, for the task of speech recognition, weights play the role of probabilities; therefore the suitable semiring would be the probability semiring $< [0, 1], +, \cdot, 0, 1 >$. For numerical stability reasons, logarithmic comutations are commonly used, so the costs are the negative logarithms of the probabilities. Since the goal is to find the most probable word sequence (using the Viterbi algorithm), the most suitable semiring for speech recognition is the tropical semiring $< \mathbb{R}_+ \cup \{\infty\}, \min, +, \infty, 0 >$. Sometimes, the log semiring $< \mathbb{R}_+ \cup \{\infty\}, \oplus_{\log}, +, \infty, 0 >$ is also used, where $x \oplus_{\log} y = -\log(e^{-x} + e^{-y})$.

[1]Without loss of generality, the Weighted FA (WFA) can be limited so that only an initial weighted state $\bar{1}$ is allowed. This is a convention followed in practice by systems that implement WFA for simplicity [21, 22].

2.2 Basic Operations

The true strength of WFSTs comes form the existence of various unary and binary operations which enable their efficient manipulation and transformation. This section is an overview of the most basic operations.

2.2.1 Rational Operations

According to Theory of Computation, three regular operations are defines on langauges; union, concatenation, andKleene closure (or Kleene star) [18]: If L_1 and L_2 are regular languages, then

- the union of L_1, L_2 is $L_1 \cup L_2 = \{x : x \in L_1 \text{ or } x \in L_2\}$,
- the concatenation of L_1, L_2 is $L_1 \cdot L_2 = \{xy : x \in L_1 \text{ and } y \in L_2\}$,
- the Kleene closure of L_1 is $L_1^* = \{x_1 x_2 \cdots x_k : k \geq 0 \text{ and } x_i \in L_1 \forall i\}$.

Since every regular language can be represented by the FSA that accepts it, those operations can be defined in the FSA framework, as well.

Similarly, the following three rational operations are defined for WFSTs [23], where by $T(x, y)$ we denote the total weight for the transformation of string x to string y through the transducer T:

- the union (or sum) of two WFSTs T_1 and T_2

$$(T_1 \cup T_2)(x, y) = (T_1 \oplus T_2)(x, y) = T_1(x, y) \oplus T_2(x, y) \quad , \forall (x, y) \in \Sigma^* \times \Delta^* , \tag{2.2}$$

- the concatenation (or product) of two WFSTs T_1 and T_2

$$(T_1 \cdot T_2)(x, y) = (T_1 \otimes T_2)(x, y) = \bigoplus_{\substack{x=x_1x_2 \\ y=y_1y_2}} T_1(x_1, y_1) \otimes T_2(x_2, y_2) \quad , \forall (x, y) \in \Sigma^* \times \Delta^* , \tag{2.3}$$

  where the summation is over all the possible ways of splitting string x into x_1 and x_2 and similarly for string y,

- the Kleene closure (or just closure) of a WFST

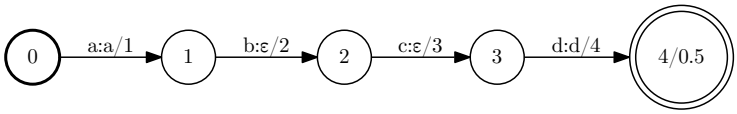
$$T^*(x, y) = \bigoplus_{n=0}^{+\infty} T^n(x, y) \quad , \forall (x, y) \in \Sigma^* \times \Delta^* , \tag{2.4}$$

  where

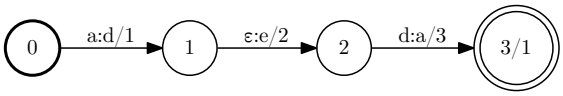
$$T^n(x, y) = \begin{cases} \overbrace{(T \otimes \cdots \otimes T)}^{\text{n times}}(x, y) & , n > 0 \\ \begin{cases} \bar{1} & , (x, y) = (\epsilon, \epsilon) \\ \bar{0} & , \text{otherwise} \end{cases} & , n = 0 \end{cases} . \tag{2.5}$$

An overview of the defined rational operations is given in Figure 2.2.

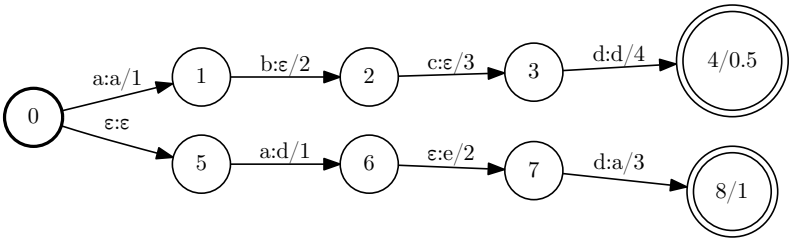
[2] From now on, the tropical semiring is the one to be used, unless it is stated otherwise.



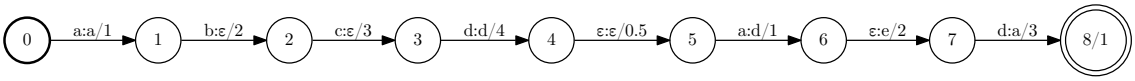
(i) WFST T_1



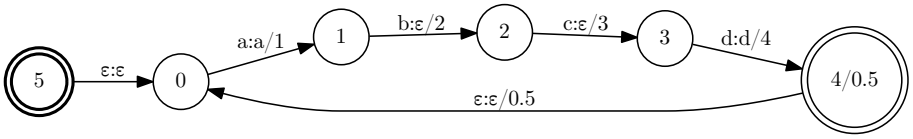
(ii) WFST T_2



(iii) Union $T_1 \cup T_2$



(iv) Concatenation $T_1 \cdot T_2$



(v) Kleene closure T_1^*

Figure 2.2: Rational operations defined WFSTs, assuming tropical semiring[2].

2.2.2 Projection, Inversion, and Composition

Some other important operations defined on a WFST projection, inversion, and composition [4]. Projection is the procedure during which a transducer is mapped to an acceptor, the transitions of which are labeled with either the input labels (upper or first projection), or the output labels (lower or second projection) of the corresponding transducer's transitions [2]. Formally, the first and second projections are respectively defined as [23]

$$\downarrow T(x) = \bigoplus_{y} T(x, y), \tag{2.6}$$

$$T(x) \downarrow = \bigoplus_{x} T(x, y)\,. \tag{2.7}$$

Inversion simply inverts the input and output labels of a transducer:

$$T^{-1}(x, y) = T(y, x)\,. \tag{2.8}$$

During composition, given two WFSTs, say $T_1 = (Q_1, \Sigma_1, \Delta_1, I_1, F_1, E_1, \lambda_1, \rho_1)$ and $T_2 = (Q_2, \Delta_1, \Delta_2, I_2, F_2, E_2, \lambda_2, \rho_2)$, a new WFST $T = (Q, \Sigma_1, \Delta_2, I, F, E, \lambda, \rho) = T_1 \circ T_2$ is

produced, where, if string x is transformed into z by the transducer T_1 and z is transformed into y by the transducer T_2, then T transforms x into y [2]. Formally,

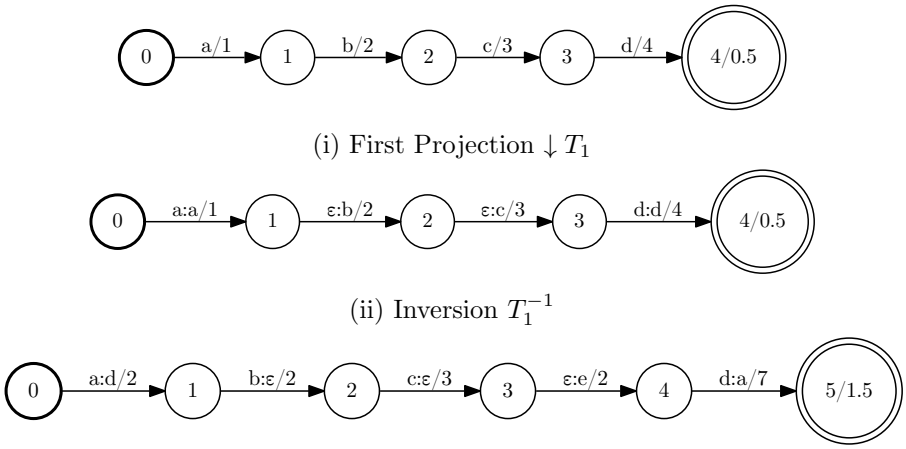
$$T_1 \circ T_2(x, y) = \bigoplus_{z \in \Delta_1^*} T_1(x, z) \otimes T_2(z, y)\,. \tag{2.9}$$

It is noted that in case of acceptors, composition is reduced into intersection.

For the algorithmic computation of composition [4], every state $q \in Q$ in T can be viewd as a pair $q = (q_1, q_2), \in Q_1 \times Q_2$. If in T_1 there exists the transitiuon t_1 from q_1 to q_1' with the label $l_i(t_1) : l_0(t_1)/w(t_1)$ and in T_2 there exists the transition t_2 from q_2 to q_2' with the label $l_i(t_2) : l_0(t_2)/q(t_2)$, then in T there exists the transition t from (q_1, q_2) to (q_1', q_2') with the label $l_i(t_1) : l_o(t_2)/(w(t_1) \otimes w(t_2))$. Additionally, any initial state (i_1, i_2) has weight $\lambda(i_1) \otimes \lambda(i_2)$ and, similarly, any final state (f_1, f_2) has weight $\rho(f_1) \otimes \rho(f_2)$.

For this analysis, however, it was assumed that no state in T_1 has ϵ-output and no state in T_2 has ϵ-input. In the more general case, a first step is required where T_1 and T_2 are transformed into T_1' and T_2', respectively, and the described algorithm computes $T_1' \circ T_2'$. During this first step, T_1' is computed by T_1 replacing ϵ in the ϵ-outputs by a new symbol, say ϵ_o, while T_2' is computed by T_2 replacing ϵ in the ϵ-inputs by a new symbol, say ϵ_i. Moreover, self-loops with labels $\epsilon : \epsilon\_i$ are introduced in all the states of T_1' and self-loops with labels $\epsilon\_o : \epsilon$ are introduced in all the states of T_2' [4].

An overview of the defined basic operations is given in Figure 2.3.



(i) First Projection $\downarrow T_1$

(ii) Inversion T_1^{-1}

(iii) Composition $T_1 \circ T_2$

Figure 2.3: Projection, inversion, and composition on WFSTs. T_1 and T_2 are defined in Figure 2.2.

2.3 Optimization Operations

The optimization operations that can be applied to a WFST aim at changing their structure in a way that their manipulation is more efficient in terms of time as well as of memory. Obviously, the output of this tranformation has to be an equivalent WFST. Two WFSTs are called equivalent if they tranform the same input string to the same output string with the same total weight [20].

Maybe the most importat optimization operation on a WFST is its transformation to a deterministic one (determinization). A transducer is called deterministic or sequential if from each state there is one and only one transition given an input label and in addition no state has an ϵ-input [20]. Contrary to the classic theory of FA that deals with automata without weights, according to which for each NFA there is an equivalent DFA, this does not hold for automata with weights. However, almost all the WFSTs used for speech recognition can be transformed into deterministic, either directly or after some auxiliary transformations. For example, every acyclic automaton with weights has a deterministic equivalent [20].

The algorithm to find the deterministic equivalent of a WFST assumes that the semiring being used is weakly left-divisible, which means that for every x and y in the set $\mathbb{A}$ of the semiring, such as $x \oplus y \neq \bar{0}$, there is at least one $z \in \mathbb{A}$ such that $x = (x \oplus y) \otimes z$ [4]. Tropical semiring is weakly left-divisible.

If the determinization is applied to an ϵ-NFA, vieweing ϵ as a usual symbol from the alphabet, then the produced FA will continue to have ϵ-transitions, which means it will not be deterministic. The operation during which the ϵ-transitions are removed from an automaton is called ϵ-removal [4]. It is an important operation, since the existence of ϵs intorduces time bottlenecks in lots of applications. The produced WFST does not contain new states, but is does contain new transitions, which, however, do not alter the relation that the initial WFST represents. The transitions removed from an FST with ϵ-removal are only the ones labeled as $\epsilon : \epsilon$, which means that after this operation, it is possible that there still exist ϵs in input or output.

To remove as many ϵs as possible, another operation, called synchronization, takes place as a previous step. During synchronization, given a WFST T, an equivalent WFST T' is computed, which is synchronized [23]. A WFST is called synchronized if the delay of every seccessful path is 0 or changes striclty monotonically. The delay $d(\pi)$ of a path π is defined as the difference between the length of the output string and the length of the input string of the path. The algorithm which is used for this operation only requires that the WFST on which it is applied has vounded delays. A necessary and sufficient codnition is that the delay of any circle of the WFST is equal to 0. Intuitively, during synchronization, transitions of the form$\epsilon : x$ and $x : \epsilon$ are reduced and transitions of the form $x : x$, with no ϵs, and of the form $\epsilon : \epsilon$, which can be deleted during ϵ-removal, are increased [4].

Synchronization leads to a more efficient distribution of ϵs in a WFST. On the other hand, weight pushing leads to a more efficient distribution of the weights. Specifically, during weight pushing the weights "are pushed" towards more initial states of the WFST. That way, the search of shortest paths (meaning paths with minimum weight in a tropical semiring) is substantially accelerated, since we can reject paths which seem to be associated with big weights from early on [4].

The corresponding algorithm operates in two steps. First, a potential $V(q)$ is computed for each state q of a WFST T. Denoting as $\Pi(q, F)$ the set of the paths from state q to a final state $q' \in F$, $V(q)$ is defined as the shortest path from q to F:

$$V(q) = \bigoplus_{\pi \in \Pi(q,F)} \{w(\pi) \otimes \rho(n(\pi))\}, \tag{2.10}$$

where $w(\pi)$ is the total weight of the path and $n(\pi)$ is the final state of the path. After that, denoting the beginning of a transition e as $p(e)$ and the end as $n(e)$, we re-calculate

the weights as:

$$w(e) = V(p(e))^{-1} \otimes w(e) \otimes V(p(e)), \qquad \forall e : V(p(e)) \neq \bar{0} \tag{2.11}$$

$$\lambda(q) = \lambda(q) \otimes V(q), \qquad \forall q \in I \tag{2.12}$$

$$\rho(q) = V(q)^{-1} \otimes \rho(q), \qquad \forall q \in F : V(q) \neq \bar{0} \tag{2.13}$$

The algorithm assumes that the semiring in use is weakly left-divisible, k-closed and zero-sum free. A semiring on the set $\mathbb{A}$ is called k-closed if there exists $k \geq 0$ such that $\bigoplus_{n=0}^{k+1} x^n = \bigoplus_{n=0}^{k} x^n$ for any $x \in \mathbb{A}$, and is called zero-sum free if $x \oplus y = \bar{0} \Rightarrow x = y = \bar{0}$ for any $x, y \in \mathbb{A}$. Tropical semiring satisfies all three assumptions, (and it is 0-closed).

Finally, to avoid resundancies and for a better memory usage, another operation of major importance is the minimization of an FA, that is the construction of an equivalent FA with the minimum number of states. For the minimization of a WFST [23], it is enough to first apply the weight pushing algorithm and then a common minimization algorithm, as it would be applied to an FSA, vieweing the label $l_i(e) : l_o(e)/w(e)$ of a transition e as a single input label, as if it were an FSA. It is noted that the common minimization algorithms assume DFA and not NFA, so a first determinization of the WFST is is necessary.

During the minimization of a DFA all the unreachable states are removed and then all the equivalent states are merged. Two states are called equivalnet if they are not k-distinguishable for any k. The definition of k-distinguishability is recursive: Two states are 0-distinguishable if one is final and the other is not, while they are called $(i+1)$-distinguishable if there is a symbol (label) with which they lead to i-distinguishable states.

Some examples of applying the optimization operations that were analyzed are shown in Figure 2.4. We should note that the sequence of applying the optimization algorithms on a WFST is of crucial importance. For example, in Figure 2.4vi a non-deterministic WFST is produced, although the determinization of the WFST in Figure 2.4i has been preceded. It is also observed that during the minimization of the WFST in Figure 2.4ii into the WFST in Figure 2.4vii a weight pushing is also applied, as expected according to the described algorithm.

2.4 WFSTs and Speech Recognition

WFSTs offer a unifying framework, both for the language and acoustic models, as well as for other sources of information in a speech recognition system, such as the pronunciation lexicon. From this point of view, we finally have in our disposal a unique static WFST which has been constructed by the composition of component WFSTs and which directly creates a search network for the mapping of a sequence of acoustic features, or observations as had been defined in the HMM-framework, into a sequence of words. That way, the decoding becomes faster and also various redundancies are removed through optimization techniques, as described in Section 2.3.

Ignoring, for simplicity, the parameters LMFS and WIP, introduced in Section 1.4, the probabilistc representation of a speech recognition model is given by the relation (1.3). In fact, many times the lexicon does not contain only one mapping between words and pronunciations, that is between sequences of letters and sequencies of phonemes, for the construction of a word, but it also contains the information of how probable the occurence of a phoneme sequence V is, given the word W [4]. Therefore, relation (1.3) is transformed

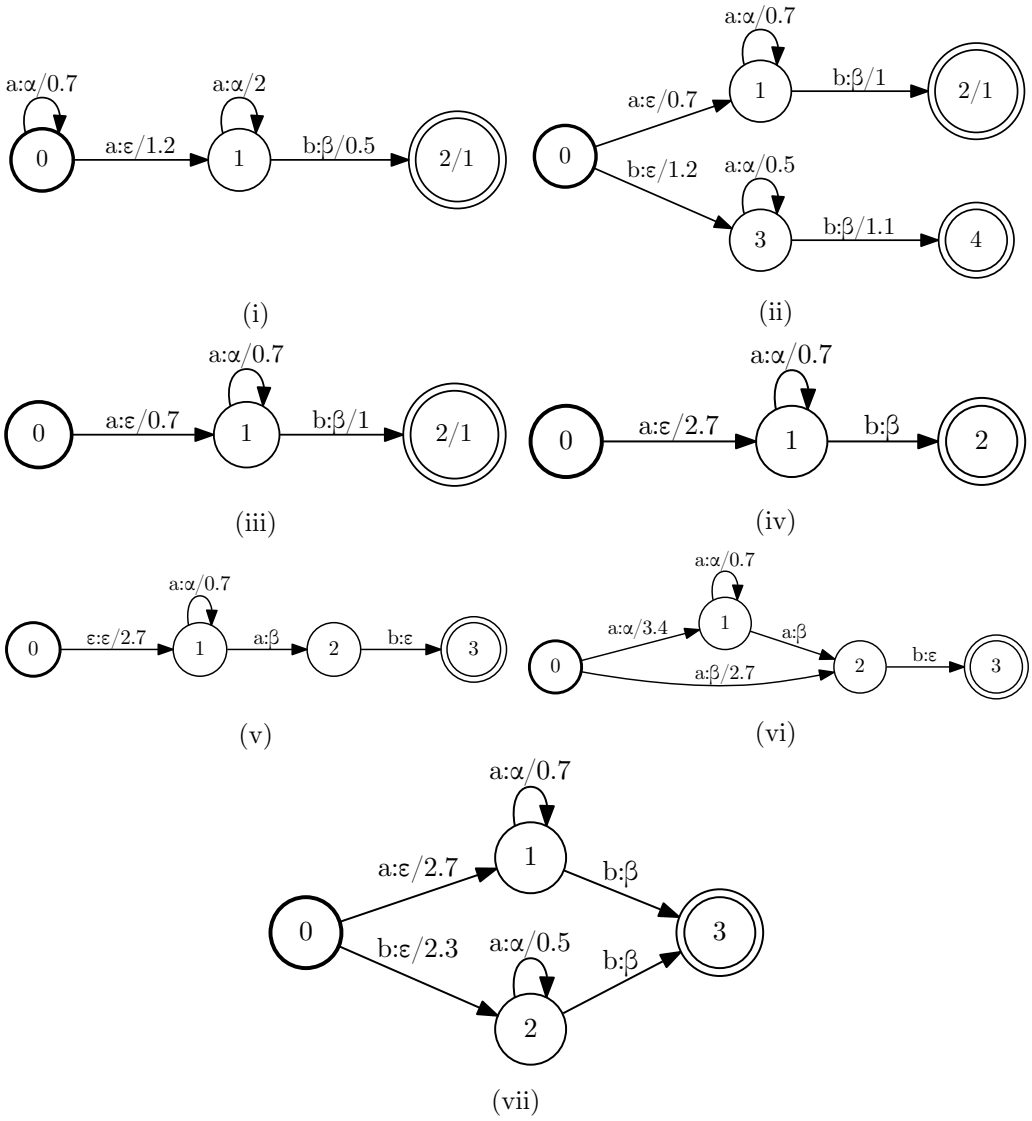


Figure 2.4: Optimization Operations on WFSTs. (i) Non-deterministic WFST. (ii) Non-minimum WFST. (iii) Deterministic WFST, equivalent to (i). (iv) Equivalent WFST to (iii) after weight pushing. (v) Equivalent WFST to (iv) after synchronization. (vi) Equivalent WFST to (v) after ϵ-removal. (vii) Minimum WFST, equivalent to (ii).

into the relation

$$\hat{W} = \underset{W \in \mathcal{W}}{\operatorname{argmax}} \left\{ \sum_{V \in R(W)} p(O|V,W)P(V|W)P(W) \right\} \tag{2.14}$$

$$\approx \underset{W \in \mathcal{W}}{\operatorname{argmax}} \left\{ \sum_{V \in R(W)} p(O|V)P(V|W)P(W) \right\}, \tag{2.15}$$

where $\mathcal{W}$ is the set of possible sequences of words and $R(W)$ is the set of the possible sequences of phonemes for the word W.

As has been explained, Viterbi algorithm is used for decoding, which always underestimates, but is computationally tractable and is based on the replacement of the summation in the last equation by a max function. Therefore, we actually get the formula

$$\hat{W} \approx \underset{W \in \mathcal{W}}{\operatorname{argmax}} \left\{ \max_{V \in R(W)} p(O|V) P(V|W) P(W) \right\} \tag{2.16}$$

and, using logarithms, the formula

$$\hat{W} \approx \underset{W \in \mathcal{W}}{\operatorname{argmax}} \left\{ \max_{V \in R(W)} \{\log p(O|V) + \log P(V|W) + \log P(W)\} \right\} . \tag{2.17}$$

In the language of WFSTs, we have the WFST H which transforms a sequence of acoustic features O into a sequence of phonemes V with a weight $w_H(O \to V) = -\log P(O|V)$, the WFST L which transforms a sequence of phonemes V into a sequence of words W with a weight $w_L(V \to W) = -\log P(V|W)$ and WFSA G which accepts a sequence of words W with a weight $w_G(W) = -\log P(W)$. Those automata are composed into a final WFST N:

$$N = H \circ L \circ G \,. \tag{2.18}$$

Thus, the problem of speech recognition is reduced to a problem of finding the shortest path of the WFST given a sequence O:

$$\hat{W} \approx \underset{W \in \mathcal{W}}{\operatorname{argmin}} \left\{ \min_{V \in R(W)} \{(-\log p(O|V)) + (-\log P(V|W)) + (-\log P(W))\} \right\} \tag{2.19}$$

$$= \underset{W \in \mathcal{W}}{\operatorname{argmin}} \left\{ \min_{V \in R(W)} \{w_H(O \to V) \otimes w_L(V \to W) \otimes w_G(W)\} \right\} \tag{2.20}$$

$$= \underset{W \in \mathcal{W}}{\operatorname{argmin}} \, w_N(O \to W) \,, \tag{2.21}$$

according to the operations defines on the semiring.

When using triphone models, it is necessary to have one more WFST C, which transforms a sequence of triphones into a sequence of phonemes, where each phoneme is context-independent and is identical to the central phoneme of the corresponing triphone. After this addition, the final WFST can be written as

$$N = H \circ C \circ L \circ G \,. \tag{2.22}$$

2.4.1 Construction of the Components

The transducer H transforms a sequence of acoustic features (or observations) O into a sequence of phonemes or in general into a sequence of SUs, no matter which SUs have been chosen. Since usually the SUs used in practice are triphones, those are the ones we will consider. H, therefore, can be viewed as the set of all the elementary HMMs that model all the triphones, which are unified into a single WFST through the rational operations of union and Kleene closure [20].

However, the possible observations in each HMM state are non-quantized vectors. Thus, an HMM cannot be directly represented as FA, since the definition of the latter requires a finite alphabet, both for the input labels and the output labels (in the case of transducers). For that reason, the information carried by the set of HMMs can be split,

in a way that we get one WFST that carries the HMM toppology information and one transducer, which is out of the strict scope of the WFST definition, which models the acoustic matching probability. To better illustrate this procedure we will use an example [4].

Let x be a meta-symbol representing any possible feature vector, which means any possible observation in an HMM. Moreover, let all the triphones be modeled by left-right 3-state HMMs, where each state is denoted by S_i, $i = 1, 2, \cdots$. Every state S_i comes from a predefined, finite set of tied states, produced after the procedure described in Subsection 1.2.4. The closure of the union of two such HMMs with totally 4 tied states is illustrated in Figure 2.5i. Both the transition probabilities a_{ij} and the probabilities $b_{S_i}(x)$ of an observation x from a state S_i are generated during training the acoustic model, as described in Subsection 1.2.3. Using the tropical (or log) semiring, we get as transition weights the negative log-probabilities such that $w(x|S_i) = -\log b_{S_i}(x)$.

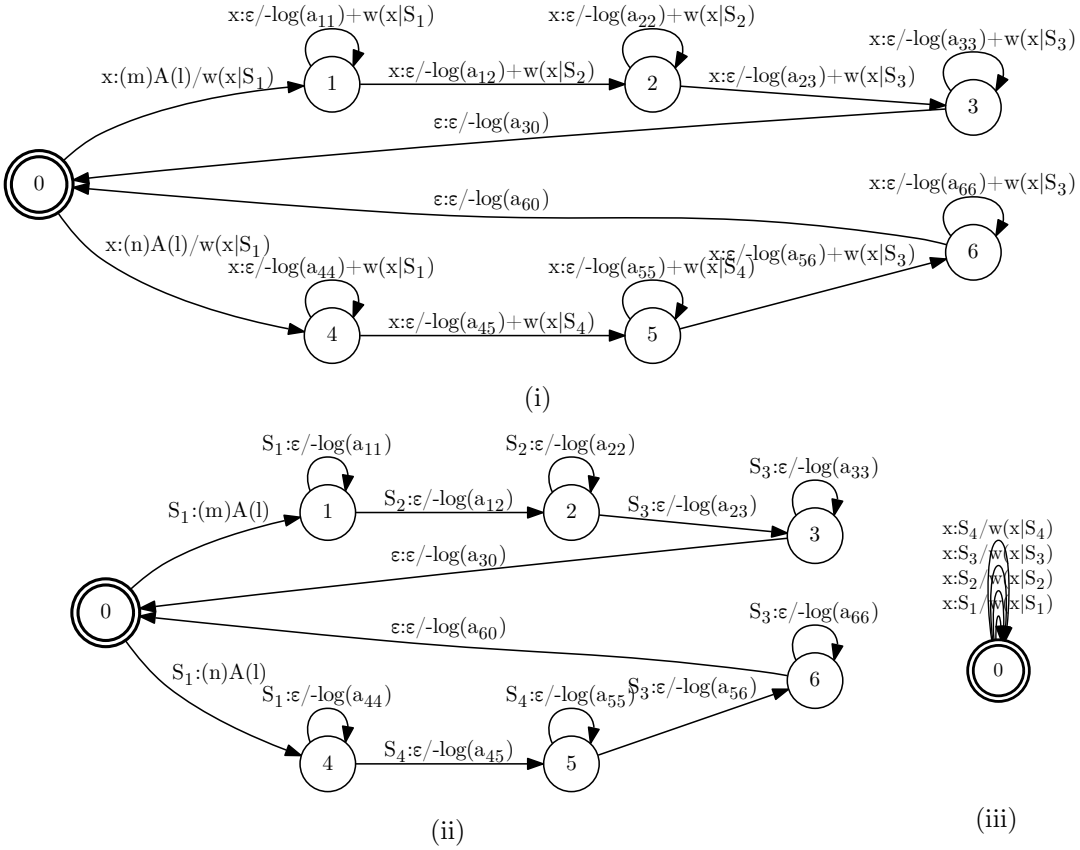


Figure 2.5: Splitting a transducer from acoustic observations to triphones into two component transducers. (i) Tranducer that transforms a sequence of acoustic observations to a sequence of triphones. It is the Kleene closure of the union of the HMMs that model the triphones $(m)A(l)$ and $(n)A(l)$. (ii) WFST that transforms a sequence of HMM states into a sequence of triphones. (iii) Transducer that transforms a sequence of acoustic features into a sequence of HMM staes.

The transducer in Figure 2.5i can be split into the WFST in Figure 2.5ii and the transducer in Figure 2.5iii. The former is actually the automaton H which is used for the construction of the WFST in equation (2.22) and converts a sequence of HMM states into a sequence of triphones (or of any context-dependent SUs chosen). The latter, which

transforms a sequence of acoustic observations into a sequence of HMM states, is directly used during decoding. It is noted that in practice, in order for N to be more efficient in terms of size, H does not contain self-loops. Those are simulated during decoding [20].

As far as the WFST C is regarded, which transforms a sequence of triphones (or more generally context-dependent SUs) into a sequence of phonemes (or more generally context-independent SUs), let's consider a simple example with just two phonemes, A and B. The WFST C' which transforms a sequence of phonemes into a sequence of triphones is more easily digestible. C is just the inversion of C' [24, 20]. Thus, C' transforms, for instance, the sequence $A/B/A$ into the sequence $(\epsilon)A(B)/(A)B(A)/(B)A(\epsilon)$. Taking into consideration all the possible combinations, the corresponsing WFST is illustrated in Figure 2.6. Since there is a big number of triphones which never occur, there is no reason to use the entire network C. Instead, C can be generated in memory exactly when needed, integrating into N only parts of it which are really required.

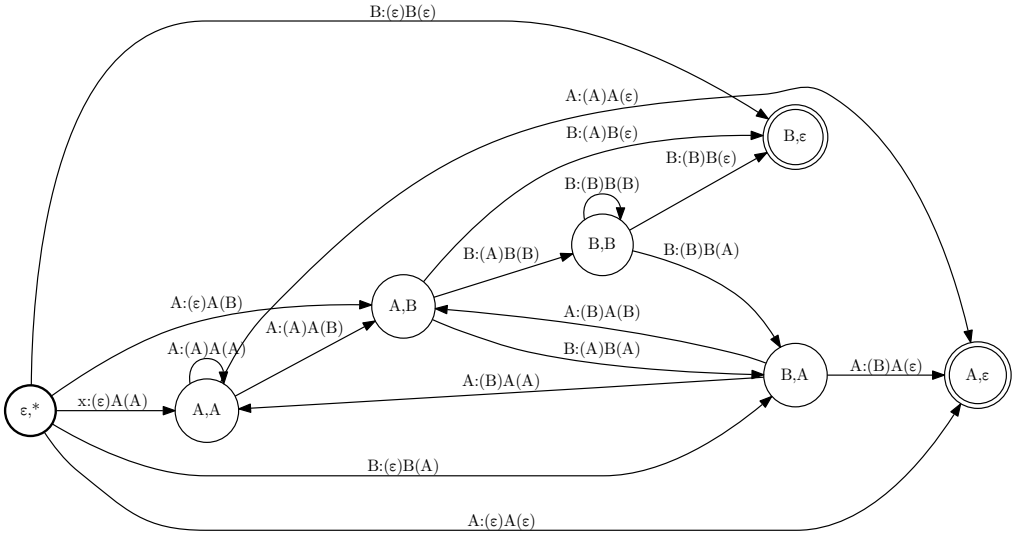


Figure 2.6: WFST which transforms a sequence of context-independent phonemes into a sequence of triphones. It is considered that only the phonemes A and B exist.

The pronunciation lexicon L, to convert a sequence of phonemes into a sequence of words, is constructed as follows. First, one FST for each instance of the lexicon in use is generated. In the general case where the lexicon also contains the probabilistic information for each pronunciation, that is how probable it is for a word to be pronounced in a specific way, then WFSTs are generated. Then, the union of all those WFSTs and the Kleene closure of the union is computed [20]. For example, considering the elementary lexicon of page 19, the procedure followed to generate L is illustrated in Figure 2.7. We can directly extend L in order to accomodate possible pauses between successive words, by adding a transition from every final state to the initial state labeled as $sil : \epsilon$.

Finally, the WFSA G, which accepts a sequence of words with some cost, is nothing but the language model, as it has been described in Section 1.3. In case of an n-gram model, which is the most common one, the representation by a WFSA is straightforward, since it is in fact a Markov model of order $n-1$ [4]. The representation of an FSG by an FSA is straightforward as well, with the sole modification being the fact that the words of the grammar should label the edges of the network instead of the states. For instance,

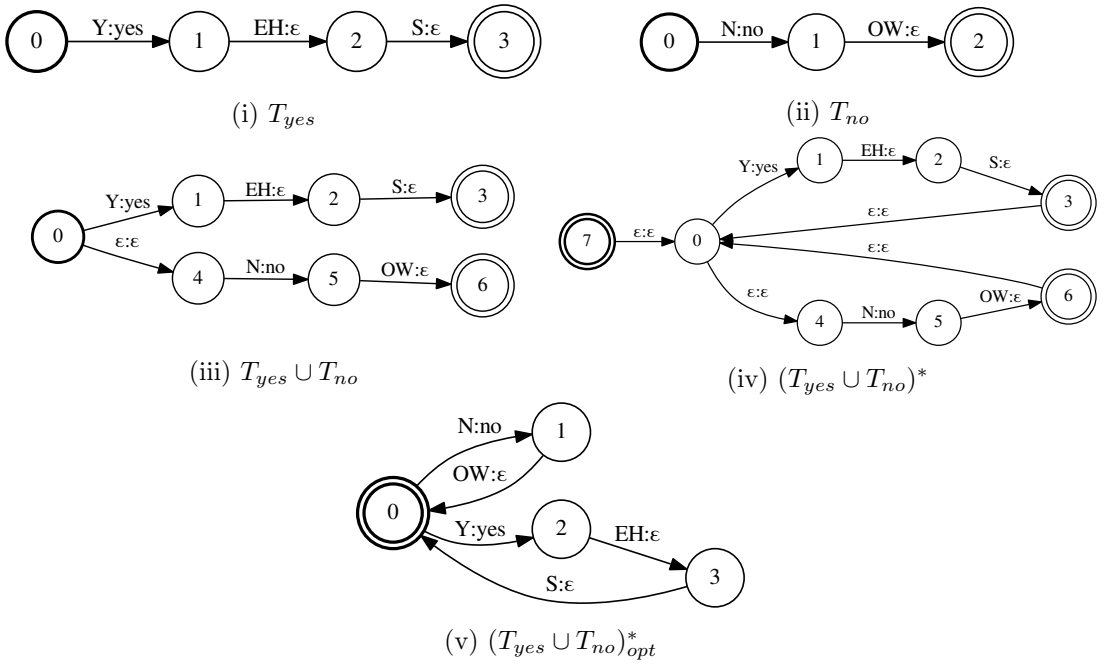


Figure 2.7: Generation of the WFST which models the pronunciation lexicon L. (i) FST for the pronunciation of the word *yes*. (ii) FST for the pronunciation of the word *no*. (iii) Union of T_{yes} and T_{no}. (iv) Kleene closure of the union of T_{yes} and T_{no}. (v) Equivalent WFST to (iv) after ϵ-removal and minimization.

the FSG of Figure 1.4 can be represented by an FSA as illustrated in Figure 2.8.

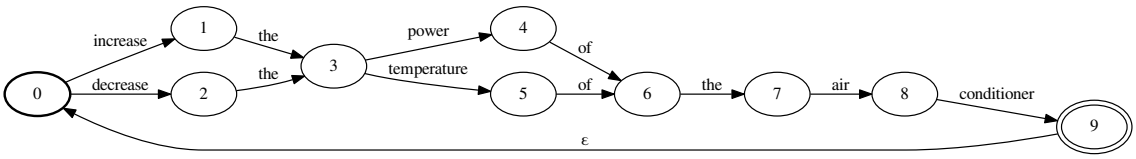


Figure 2.8: Example of FSG represented by an FSA. It is the FSG in Figure 1.4.

2.4.2 Composition and Optimization

Having constructed the required WFSTs H, C, L, and G, those can be composed into a unified WFST N, according to the equation (2.22). However, in order to avoid a huge N, gradual optimization operations, like the ones presented in Section 2.3, are required. This need becomes apparent from the WFST illustrated in Figure 2.7v, which is equivalent to that in Figure 2.7iv modeling an elementary pronunciation lexicon, but after some optimization operations having taken place.

First of all, we should make sure that any intermediate step generates a deterministic WFST [20]. Assuming that G is indeed deterministic[3], we should guarantee the deterministic nature $L \circ G$. This property often does not hold because of the potential existence of homophones. For that reason, the lexicon is extended so that at the end of each sequence

[3]Any ϵs in G are viewed as regular symbols from the alphabet. If G remains non-deterministic, its determinization is required.

of phonemes there is a special symbol (for instance, #1, #2, etc.). Even in the cases of sequence of phonemes uniquely representing a word, it is a good practice to add a special symbol at the end (for instance #1), in order to avoid errors in case of homophone phrases which consist of non-homophone words [4]. For example, *ice cream* cannot be distinguished from *I scream* (the) based on the pronunciation lexicon unless those special symbols are added. Thus, in the first case we have the sequence of phonemes {`AY #1 S K R IY M #1`}, while in the second one we get the sequence {`AY S #1 K R IY M #1`}. That way, the modified WFST $\tilde{L}$ is generated, which is then composed with G and the result of the composition can be transformed into a deterministic WFST, denoted as LG:

$$LG = \det(\tilde{L} \circ G) \tag{2.23}$$

After this modification of L, though, paths with symbols not originally generated by C are created. Therefore, C should be also transformed into $\tilde{C}$, adding self-loop transitions in each state[4], such that any auxiliary symbol can be added at any pronunciation of any word [20]. If P is the maximum number of words that can be uttered with the same pronunciation, then P self-loops should be added in each state with the labels $\#1 : \#1, \#2 : \#2, \cdots \#P : \#P$. The produced WFST $\tilde{C}$ can now be composed with LG and be determinized:

$$CLG = \det(\tilde{C} \circ LG) \tag{2.24}$$

Following a similar line of thought, P self-loops are added in the initial state of H, where the initial state is considered to be in the boundary between any two SUs, as depicted in Figure 2.5ii. Thus, the next step is the composition of $\tilde{H}$ which was just generated with the CLG and yet another determinization:

$$HCLG = \det(\tilde{H} \circ CLG) \tag{2.25}$$

$HCLG$ is then minimized and all the auxiliary symbols which have been added are replaced by the empty symbol ϵ, with the corresponding operation denoted as π_ϵ, so that we get the WFST

$$N = \pi_\epsilon(\min(HCLG)) = \pi_\epsilon(\min(\det(\tilde{H} \circ \det(\tilde{C} \circ \det(\tilde{L} \circ G))))) \,. \tag{2.26}$$

The first step of the minimization is, as already described, the weigth pushing. The choice of the semiring to be used for the weight pushing turns out to play an important role, with log semiring having substantial advantages versus the tropical one [20]. In both cases, the result of the minimization in terms of the final number of states and transitions is identical, but the weight distribution is different. It has been observed that the distribution that results after weight pushing on the log semiring makes pruning during decoding with Viterbi beam search more efficient.

The main difference as far as the weight pushing algorithm is concerned is related to the potential $V(q)$ corresponding to each state q of the WFST, according to the equation (2.10). Specifically, when using tropical semiring we get the equation

$$V(q) = \min_{\pi \in \Pi(q,F)} \{w(\pi) + \rho(n(\pi))\} \,, \tag{2.27}$$

[4] In practice, $(\det(C^{-1}))^{-1}$ is what is used here instead of C [4], because C is not deterministic and also a deterministic equivalent is not straightforward to find.

which can be computed with a classic shortest path algorithm. On the other hand, the usage of log semiring leads to a potential

$$V(q) = -\log \sum_{\pi \in \Pi(q,F)} \left\{ e^{w(\pi)+\rho(n(\pi))} \right\} , \tag{2.28}$$

which is equal to the negative logarithm of the total probability of all the paths from state q to some final state, since the corresponding weights are equal to the log-probabilities of transitions. Using such a potential function guarantees that the produced WFST will retain the usual normalization in HMMs, where the sum of all the "weights" of all the outcoming transitions from any state is required to be equal to 1.

Finally, the above steps can be succeeded by a procedure known as factoring [20]. During factoring, any chained transition is replaced by a single transition, which is labelled with the concatenation of all the labels of the component transitions in the chain, while the weight assigned is equal to the product[5] of the component weights. We define as chain a path all the states of which, apart from the first and the last ones, have only one incoming and one outcoming transition.

Since $\tilde{H}$ does not contain, as has already been mentioned, any self-loops, but those are simulated during decoding, it is expected that the final WFST N, which transforms a sequence of HMM states to a word sequence, contain multiple chains. Out of them, however, not all the chains are replaced. Instead, only the replacements which are going to reduce the size of the transducer take place. At any case, factoring does not affect the decoding run time, but only the size of the final transducer.

The decision of whether a chained transition is going to be merged into a single transition can be based on a gain function. Let a sequence of HMM states $\sigma = S_k, S_{k+1}, \cdots$ and let $C(N)$ the set of chained transitions in WFST . Then, denoting as $l_i(\pi)$ and $l_o(\pi)$ the input and output strings of a path π, respectively, the gain $G(\sigma)$ of the sequence σ is defined as

$$G(\sigma) = \sum_{\substack{\pi \in C(N) \\ l_i(\pi)=\sigma}} \{|\sigma| - |l_o(\pi)| - 1\} , \tag{2.29}$$

where $|x|$ is the length of the string x. The replacement of the string σ during factoring reduces the trancducer's size only if $G(\sigma) > 0$.

Thus, during factoring, sequences of input labels that correspond to the HMM state identities are replaced by a single string corresponding to the identity of an n-state HMM, where n is the number of transitions in the chain to be replaced. A relevant example is illustrated in Figure 2.9. Therefore, factoring results in splitting N as follows:

$$N = H' \circ F , \tag{2.30}$$

where, if we denote as fact$(\cdot)$ the factoring operation,

$$F = \text{fact}(N) = \text{fact}(\pi_\epsilon(\min(\det(\tilde{H} \circ \det(\tilde{C} \circ \det(\tilde{L} \circ G)))))) \tag{2.31}$$

and H' is a transducer that maps sequencies of n HMM states into n-state HMMs. The WFST which is actually used for the recognition task is F, since H' can be directly simulated during decoding. It is noted that alternatively, factoring can in practice precede minimization [20].

[5] For a tropical semiring the "product" is equal to the algebraic operation of summation.

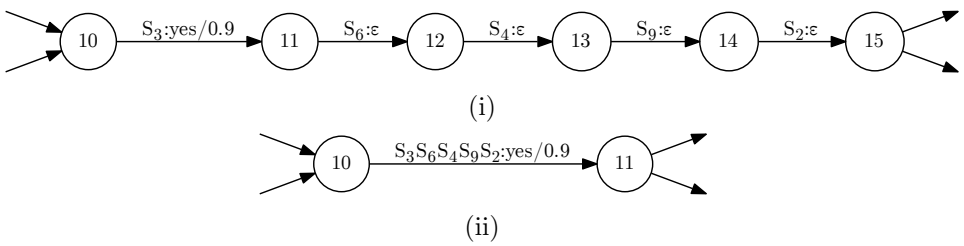


Figure 2.9: Example of factoring part of a WFST. (i) 5-transition chain. (ii) Replacement of the chained transitions by a single transition. In this example, the corresponding part of the transducer H' in formula (2.30) should map the sequence S_3, S_6, S_4, S_9, S_2 to the string $S_3S_6S_4S_9S_2$.

Chapter 3

Feature Sets for ASR

3.1 Mel Frequency Cepstrum Coefficients (MFCCs)

With no doubt, the most widely used feature sets used in practical applications are MFCCs (Mel-Frequency Cepstrum Coefficients) [25]. In this Section, we will describe how those features are extracted and we will present a few simple, common transformations applied to them.

3.1.1 Extraction of MFCCs

The preprossecing steps consist of passing the signal through a pre-emphasis system with transfer function

$$H_{preemph}(z) = 1 - \tilde{a}z^{-1}, \tilde{a} \in (0.9, 1) \tag{3.1}$$

and segmenting it in - usually overlapping - frames. In order to minimize the discontinuities in the frame boundaries and the leakage efferct between neighboring frames, Hamming windows (or of similar shape) are usually applied. A Hamming window if length L is defined as

$$w(n) = 0.54 - 0.46 \cos\left(\frac{2\pi n}{L-1}\right) . \tag{3.2}$$

Pre-emphasis is necessary to amplify the high frequencies, where usually the spectrum values of the speech signal are small and more easily affected by any noise. Windowing is necessary for speech recognition, as well as in many other applications of speech processing, because the statistical properties of the signal change over time, but can be considered to be fixed over small time segments (of about $10 - 30msec$) [3], so that the common Fourier analysis techniques can be applied. Because of the small duration of the windows used, MFCCs belong to a broad class of feature sets called short-term.

MFCCs are based on the real cepstrum of the windowed signal, derived through the Fourier transform of the signal. For their extraction, a non-linear frequency scale is used, which approximates the behavior of human hearing and has been proved to have significant advantages in the field of speech recognition. The entire idea is founded on the fact that humans perceive sound changes in low frequencies more easily, compared to changes in high frequencies.

The *mel* unit is defined in a way that pairs of sounds which have the same "distance" according to the human perception, are separated by the same number of *mels*. The

formal mapping between a frequency expressed in Hz and the corresponding frequency in mel is not unique. A widely used formula is the following [26]:

$$m = B(f) = 2595 \log_{10}\left(1 + \frac{f}{700}\right), \tag{3.3}$$

$$f = B^{-1}(m) = 700\left(10^{m/2595} - 1\right) \tag{3.4}$$

An alternative formula, which first appeared in [27], defines that the mapping between $mels$ and Hz is linear until $1000Hz$ and continues as logarithmic:

$$m = \begin{cases} \dfrac{3f}{200} & , f < 1000Hz \\ 1000 + \dfrac{\log \frac{f}{1000}}{\log 1.0711703} & , f \geq 1000Hz \end{cases} \tag{3.5}$$

Those two approaches are depicted in Figure 3.1.

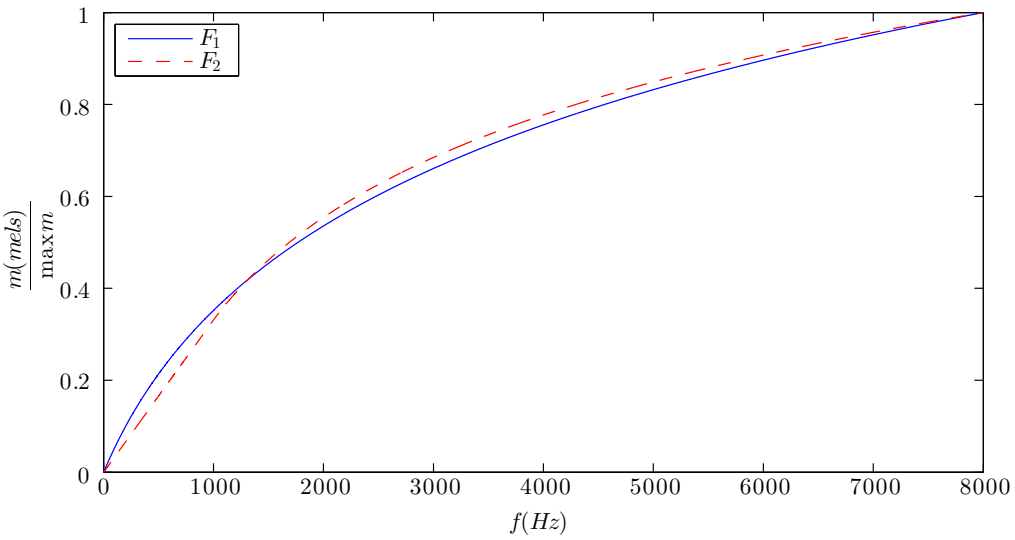


Figure 3.1: Mapping frequencies from Hz to mel.F_1 corresponds to formula (3.3), while F_2 corresponds to formula (3.5).

The idea, thus, is to use a filterbank, where each filter captures a certain frequency band, with the filterbank being more dense in the low frequencies and more sparse in the high frequencies, where human hearing is less sensitive.

The filterbank used for MFCC extraction is composed of $Q(\approx 20-40)$ triangular filters H^j, each one of which has a bandwidth such that its cutoff frequencies are the central frequencies of its two neighboring filters (in mel scale). Denoting the central frequency of the j-th filter as f_c^j, where f_c^0 and f_c^{Q+1} are the low and high cutoff frequencies of the first and last filter, respectively, and additionally assuming that $H^j(f_c^j) = 1$, the j-th filter is

described by the equation

$$H^j[k] = \begin{cases} 0 & , k < f_c^{j-1} \\ \dfrac{k - f_c^{j-1}}{f_c^j - f_c^{j-1}} & , f_c^{j-1} \leq k \leq f_c^j \\ \dfrac{f_c^{j+1} - k}{f_c^{j+1} - f_c^j} & , f_c^j \leq k \leq f_c^{j+1} \\ 0 & , k > f_c^{j+1} \end{cases} . \tag{3.6}$$

The central frequencies are uniformly distributed in the *mel* scale. Thus, denoting for simplicity as f_h and f_l the highest and lowest frequencies of the filterbank, the central frequencies are given as

$$f_c^j = \frac{N}{F_s} B^{-1}\left(B(f_l) + j\frac{B(f_h) - B(f_l)}{Q+1}\right) , \tag{3.7}$$

where N is the length of the Discrete Fourier Transform (DFT) and F_s is the sampling frequency.

Such a filterbank is illustrated in Figure 3.2, for 24 triangular filters and sampling frequency $Fs = 16kHz$. For this example, we have $f_c^0 = 0$ and $f_c^{Q+1} = Fs/2$. This is, however, not necessary, since the very low or very high frequencies are often ignored, in order to remove low- or high-frequency noise.

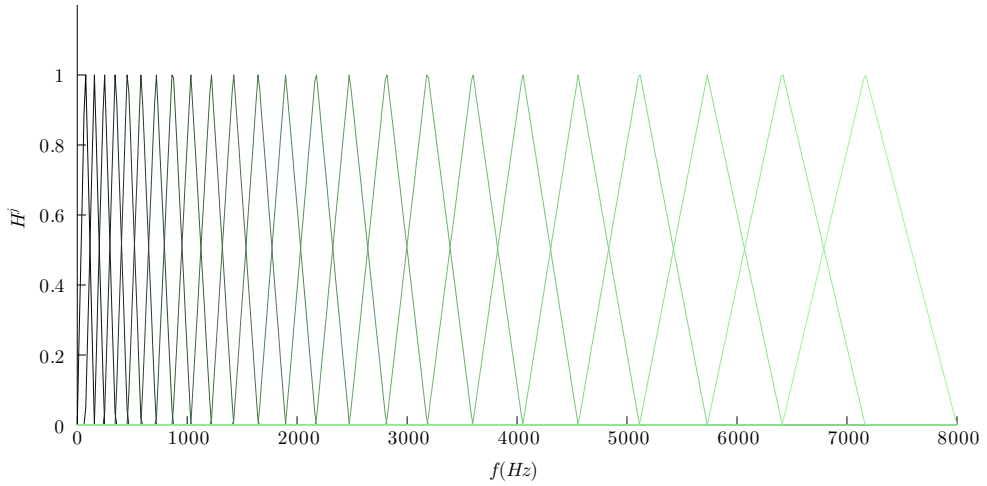


Figure 3.2: Filterbank used to extract MFCCs (24 filters).

Instead of normalizing the filters with respect to height, they can be normalized with respect to area. In that case, for each filter we have

$$\sum_{k=0}^{N-1} H^j[k] = 1 . \tag{3.8}$$

and the j-th filter is described by the equation

$$H^j[k] = \begin{cases} 0 & , k < f_c^{j-1} \\ \dfrac{2\left(k - f_c^{j-1}\right)}{\left(f_c^j - f_c^{j-1}\right)\left(f_c^{j+1} - f_c^{j-1}\right)} & , f_c^{j-1} \leq k \leq f_c^j \\ \dfrac{2\left(f_c^{j+1} - k\right)}{\left(f_c^{j+1} - f_c^j\right)\left(f_c^{j+1} - f_c^{j-1}\right)} & , f_c^j \leq k \leq f_c^{j+1} \\ 0 & , k > f_c^{j+1} \end{cases} . \tag{3.9}$$

The next step is to compute the logarithm of the energy, say $G_i(j)$ of the response of the j-th filter when the input is the windowed frame $s_i[n]$. We know that the output of a system in the frequency domain is equal to the product between the system's frequency response and the input's Fourier transformation. Having that in mind, and using Parseval's theorem which relates the squared signal to the power spectrum according to the equation

$$\sum_{n=0}^{N-1} s[n]^2 = \frac{1}{N} \sum_{k=0}^{N-1} |S[k]|^2 , \tag{3.10}$$

we directly concude that

$$G_i(j) = \log\left\{ \frac{1}{N} \sum_{k=0}^{N-1} |S_i[k] \cdot H^j[k]|^2 \right\} = \log\left\{ \frac{2}{N} \sum_{k=0}^{N/2} |S_i[k] \cdot H^j[k]|^2 \right\} , \tag{3.11}$$

where $S_i[k]$ is the N-point DFT of $s_i[n]$. The additive constant $\log\{2/N\}$ is omitted, while the following modified formula is also frequently used:

$$G_i(j) = \log\left\{ \sum_{k=0}^{N/2} |S_i[k]|^2 \cdot |H^j[k]| \right\} . \tag{3.12}$$

Finally, MFCCs for the i-th frame are the first N_c coefficients of the Discrete Cosine Transform (DCT) of the energy $\{G_i(j)\}_{j=1}^{Q}$. A typical choice for speech recognition is $N_c = 13$. More precisely, the coefficients $2 - 13$ are typically used. On those 12 features, the squared energy of the signal is added, given by the equation (3.10), expressed in the logarithmic domain.

The idea of using cepstral characteristics is based on the fact that for successful recognition we need features able to distinguish the different phonemes as much as possible. Those features are related to the position of the various articulators of the vocal tract or, in other words, to the formants of the "filter", and not to the excitation source (e.g. vocal cords vibrating). As we know, cepstral analysis is a reliable way to distinguish the source spectrum from the filter spectrum [3].

On the other hand, DCT is used in order to decorrelate and compress the data (since we keep only the first few coefficients). In particular, for speech signals, DCT approximates very well the Karhunen-Loève Transform (KLT), which is optimal with respect to the minimization of the energy of the compression error [28]. Additionally, DCT outputs real-valued vectors, somthing which facilitates their efficient storage and processing.

Two practical details on the aforementioned procedure are the techiques of dithering liftering. Dithering is the process of adding 1-bit random noise bit to the waveform, which is almost equivalent to adding a constant to the spectrum. This takes place in order to eliminate the existence of zero values in the spectrum, something which could lead to problems during the computation of the corresponding logarithmic values.

As far as liftering is regarded, it is a process during which each MFCC x_i, is multiplied with a weight w_i, so that a new feature y_i [29] is produced. It is, thus, a linear transformation, which is described by the relationship

$$\mathbf{y} = \mathbf{W}\mathbf{x}\,, \tag{3.13}$$

where $\mathbf{W} = \text{diag}\{w_1, w_2, \cdots, w_{N_c}\}$. The diagonal elements of $\mathbf{W}$ are chosen in such a way that all the final MFCCs have a similar dynamic range. This was of great importance during the first days of speech recognition, when it was based on the Dynamic Time Warping (DTW) algorithm and on the euclidean distances between the feature vectors. Nowadays, liftering has limited pratical value, but is often used for historical and compatibility reasons.

3.1.2 Cepstral Mean (& Variance) Normalization

A very simple technique proposed to aleviate the negative effects of noise is the so-called Cepstral Mean Normalization (CMN), also known as Cepstral Mean Subtraction (CMS). Initially, it was used as a solution to the problems imposed because of recordings with different microphones, but it was shown that this technique can improve the performance even when the communication or recording channel doesn't change [30].

In general, CMN helps a lot when there is convolutive noise, including distortions because of microphone-specific characteristics and distorions due to reverberation, where the clean speech signal is convlolved with the impulse response of the microphone or the room, respectively. Having in mind the properties of the Fourier Transform, as well as the identity $\log(x \cdot y) = \log x + \log y$, we directly get that convolution in time domain is equivalent to summation in quefrency domain, which is the domain where the cepstral features are estimated. Formally, if $y(n) = x(n) * h(n)$, then $Y(q) = X(q) + H(q)$, where the quefrency is denoted by q. Assuming that the impulse response $h(n)$ is fixed throughout the utterance, for the i-th frame (out of totally F frames) we 'll have $Y_i(q) = X_i(q) + H(q)$. Subtracting the arithmetic mean of all the frames we get

$$\begin{aligned} Y_i'(q) &= Y_i(q) - \frac{1}{F}\sum_{i=1}^{F} Y_i(q) \\ &= X_i(q) - \frac{1}{F}\sum_{i=1}^{F} X_i(q)\,, \end{aligned}$$

so it becomes clear that the effect of $h(n)$ has been eliminated. If we additionally normalize with respect to the standard deviation, then the resulting transformation is called Cepstral Mean and Variance Normalization (CMVN).

For multiple-speaker speech recognition applications, it is often suggested that CMVN is done per speaker and not per utterance, so that the sufficient statistics are collected from all the frames of all the utterances of a specific speaker (of course, this assumes that the speaker information is available). That way, any variations in the cepstral features due to speaker-specific characteristics are eliminated.

3.1.3 Derivatives of MFCCs

For time windows shorter than $100msec$, human capability to distinguish different speech characteristics is not satisfactory [31]. Thus, since for short-term features, the frame length is less than $\sim 30msec$, it seems that we need a way to include information about a longer time interval in the feature set.

The simplest way to deal with the aforementioned issue is to augment the i-th feature vector so that it contains features which correspond not only to the i-th frame, but also to its neighboring frames, or its relationship with them. Here we will mention the most widely used approach, first proposed in [32], which is the augmentation of the MFCCs vector with dynamic features which capture the temporal behavior of the MFCCs. It is noted that, even though we refer to MFCCs, the method is much more general.

The first-order synamic features, often called velocity coefficients and denoted as Δs, are estimated as the first-order orthogonal polynomial coefficients. Considering the consecutive frames $\{\cdots, i-2, i-1, i, i+1, i+2, \cdots\}$, and assuming that 13 MFCCs $x(k),\ k = 1, 2, \cdots, 13$ have been extracted for each frame, the velocity coefficients for the i-th frame are stimated as

$$\Delta x_i(k) = \frac{\sum_{m=-M}^{M} m \cdot x_{i+m}(k)}{\sum_{m=-M}^{M} m^2}. \tag{3.14}$$

Constant M denotes the context of the derivation, that is how many neighboring frames are taken into consideration during the computation of the dynamic features. M is usually in the range $[1, 10]$ [32].

Equation (3.14) can be re-used to get second-order (acceleration or $\Delta\Delta$), third-order, ... coefficients. Fr example, if a recognizer uses 13 MFCCs, together with the corresponding Δ and $\Delta\Delta$ coefficients, then the length of the feature vector is $3 \cdot 13 = 39$.

3.1.4 Delta-Spectral Cepstal Coefficients

An idea proposed for robust speech recognition has to do with the computation of the dynamic features in the spectral, instead of the cepstral, domain, with the resulting features known as Delta-Spectral Cepstral Coefficients (DSCCs) [33].

The idea is based on the fact that the spectral characteristics of human speech change much faster than the spectral characteristics of the noisy background. It is supported that this different behavior between speech and background can be better reflected directly in the spectral domain, that is before the non-linearity of the logarithm and the DCT take place. Again, the approach is general, but we are going to describe it when applied to and combined with MFCCs.

Figure 3.3 presents the workflow followed for the DSCC extraction, and is compared to the extraction of the usual Δ and $\Delta\Delta$ coefficients. DSCCs are typically combined with the 13 static MFCCs, generating again a vector with 39 features.

Histogram normalization takes place because computing the derivative in the spectral domain results in features with a very large dynamic range - something which is not desired by itself - but most of the coefficients have values around the neihborhood of 0.

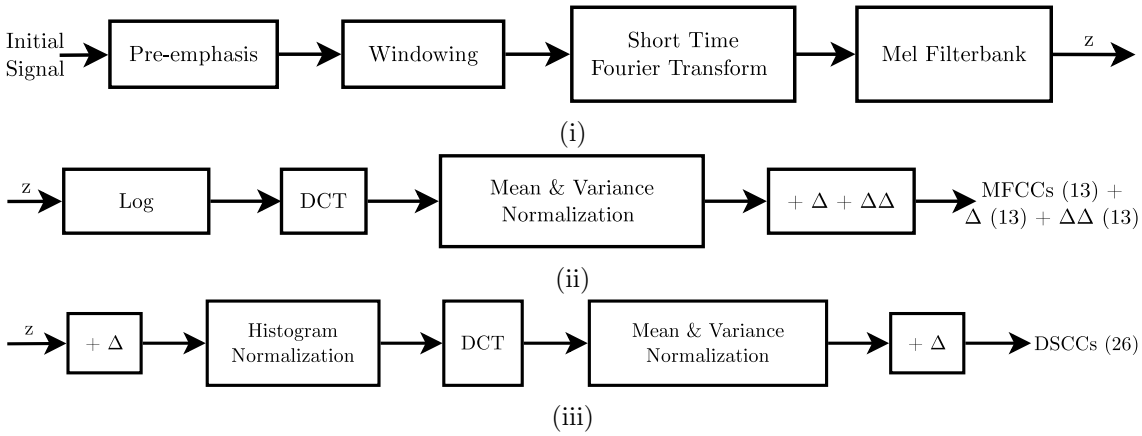


Figure 3.3: Workflow followed to extract MFCCs and DSCCs. (i) Initial stages of the process. (ii) MFCC and standard Δ and $\Delta\Delta$ coefficients extraction. (iii) DSCC extraction. [*image adapted from [33]*]

After this normalization, which is applied per utterance, the values of the coefficients are following a normal distribution with zero mean and unit standard deviation. The effect of this gaussianization is visually depicted in Figure 3.4.

It is noted that instead of (3.14), the Δ coefficients for DSCCs are extracted using the formula

$$\Delta x_i(k) = x_{i+M} - x_{i-M}\,. \tag{3.15}$$

3.2 Perceptual Linear Prediction (PLP) and RASTA Analysis

3.2.1 PLP Analysis and Feature Extraction

Another widely used feature set in the short-term family is the one steming from the Perceptual Linear Prediction (PLP). We refer to the resulting features as [34]. PLPs are based on the same fundamental principles as MFCCs, but in PLP analysis, there is an effort for more precise estimation of the human autitory characteristics.

The starting point for PLP was linear prediction, which was used during the first years of speech recognition. Linear prediction provides a tool for satisfactory estimation of the poles of the transfer function which describes the human speech production system. Those poles correspond to the so-called formants, which are the characteristic resonanses of the vocal tract. Formants change through time, depending on the shape of the vocal tract for the production of the different sounds; thus, estimating the formants we can in fact estimate the sound to be recognized. One of the main disadvantages of the linear prediction, however, is that the resulting model approximates the speech spectrum with the same accuracy at every frequency. However, the sensitivity of human hearing decreases as the frequency increases, especially above $800Hz$.

PLP, thus, tries to aleviate the aforementioned disadvantage. Moreover, according to [34], linear prediction will give different results for two utterances with different spectra, but with the same linguistic information, while PLP will give similar results, something with obvious advantages for the final goal of speech recognition.

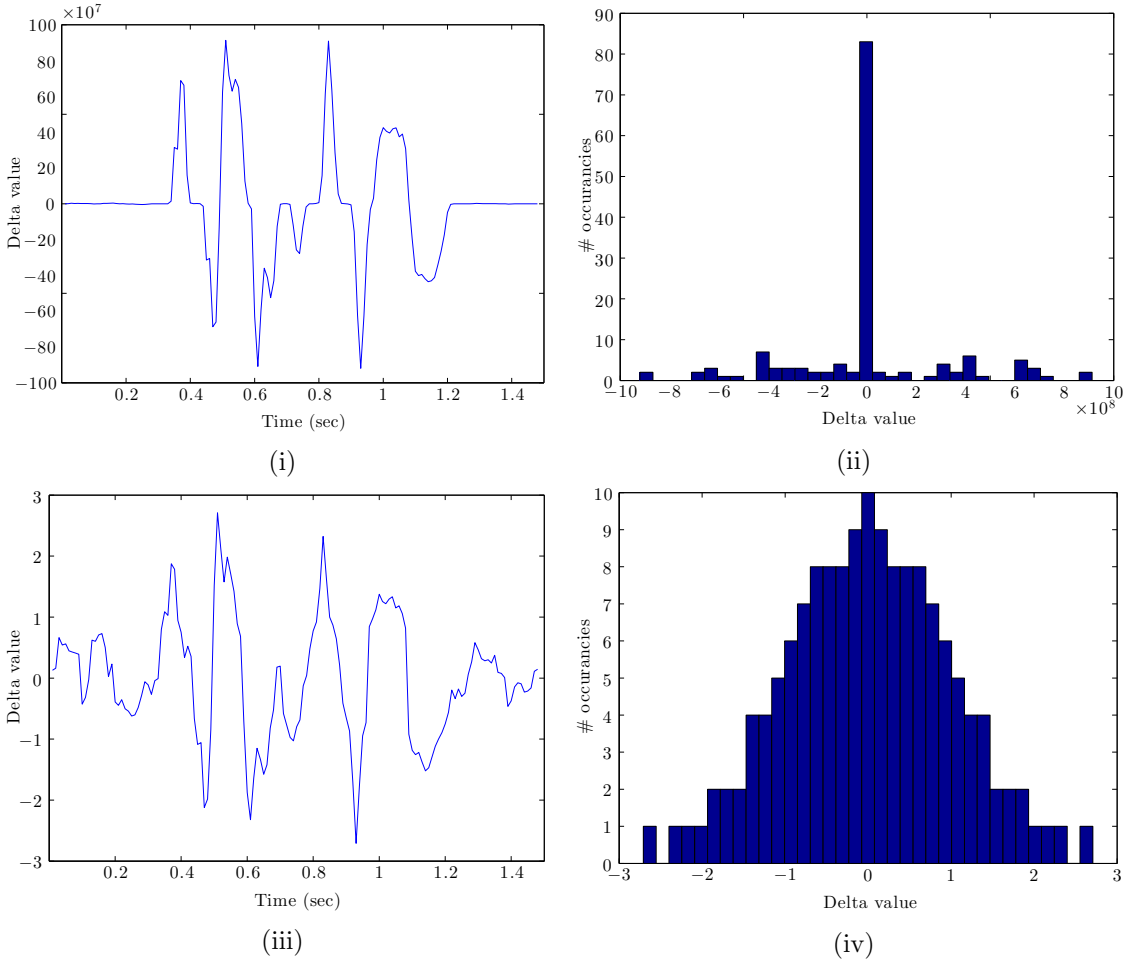


Figure 3.4: Effect of histogram normalization during DSCC extraction. (i) DSCC trajectory for the 10th filter of the bank before the normalization. (ii) Histograms of the values in (i). (iii) DSCC trajectory for the 10th filter of the bank before the normalization. (iv) Histogram of the values in (iii). We plot the first-order coefficients for a speech signal recorded with a close-talk microphone.

As is the case during the MFCC extraction, the signal is first windowed into overlapping frames using a Hamming (or Hamming-like) window and then the power spectrum is computed through DFT. Without passing to the logarithmic domain, we conclude to the equation (3.16) (similarly to (3.12)).

$$G_i(j) = \sum_{k=0}^{N/2} \left\{ |S_i[k]|^2 \cdot |H^j[k]| \right\} \tag{3.16}$$

For PLPs, however, the non-linear psycho-acoustic scale used is *Bark* and not *mel*. The formal mapping between a frequency expressed in Hz and the corresponding frequency in $Barks$ is not unique, but the one used in [34] is given by the equations (3.17)-(3.18),

and the corresponing result is illustrated in Figure 3.5.

$$B = 6 \operatorname{arcsinh}\left(\frac{f}{600}\right) = 6 \log\left\{\frac{f}{600} + \sqrt{\left(\frac{f}{600}\right)^2 + 1}\right\} \tag{3.17}$$

$$f = 600 \sinh\left(\frac{B}{6}\right) \tag{3.18}$$

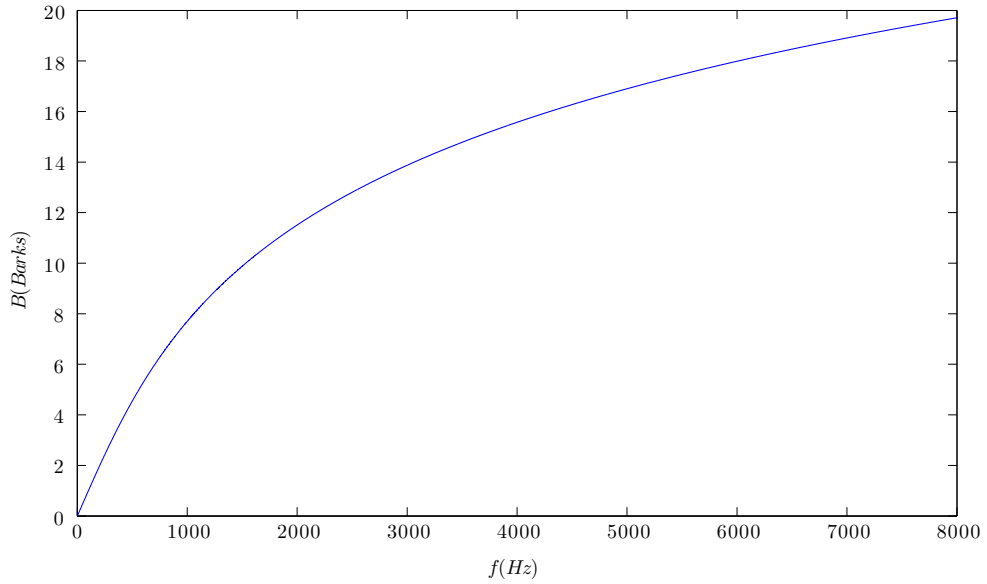


Figure 3.5: Mapping frequencies from Hz to $Bark$.

The central frequencies of the filters are uniformly distributed in the $Bark$ scale, with the central frequency of the first filter being equal to $0Hz$ and the distince between any two consecutive filters being $1Bark$. So, if, for example, the sampling frequency is $16kHz$, 21 filters will be needed, the last one of which will have a central frequency equal to $8398Hz$, that is a bit larger than the Nyquist frequency. All the filters have the same shape in the $Bark$ scale, which is described by the equation (3.19), where the central frequency of the j-th filter is denoted as B_c^j.

$$H^j(B) = \begin{cases} 0 & , B - B_c^j < -1.3 \\ 10^{2.5(B+0.5)} & , -1.3 \leq B - B_c^j \leq -0.5 \\ 1 & , -0.5 \leq B - B_c^j \leq 0.5 \\ 10^{-(B-0.5)} & , 0.5 \leq B - B_c^j \leq 2.5 \\ 0 & , B - B_c^j > 2.5 \end{cases} \tag{3.19}$$

According to this equation, the filters have a trapezoid shape, where the slope towards the low frequencies ($10dB/Bark$) os significantly smoother than the slope towards the higher frequencies ($25db/Bark$) [35]. The filterbank is visually depicted in Figure 3.6 for sampling frequency $F_s = 16kHz$.

Through such a filterbank, we say that the power spectrum of the signal $s_i(n)$ is decomposed into critical bands. For deterministic signals, the power spectrumis equal to

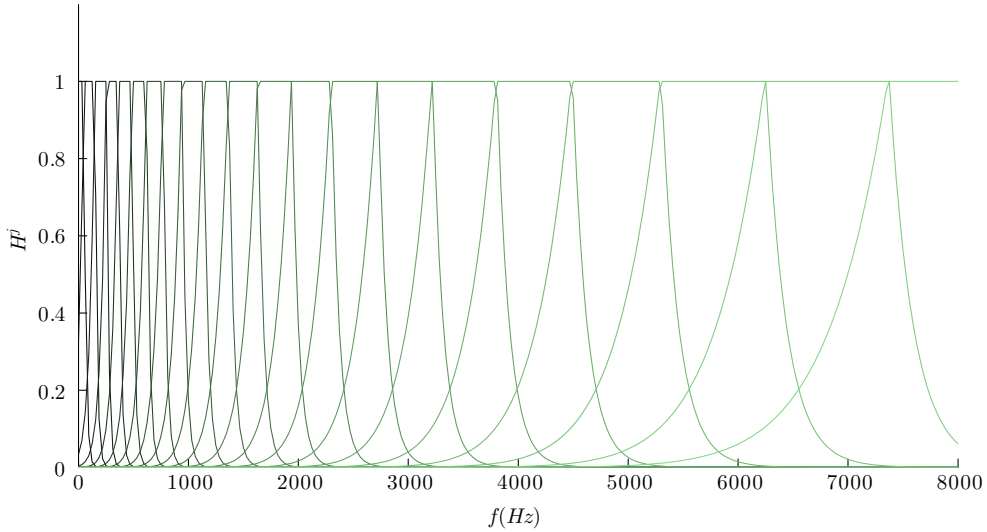


Figure 3.6: Filterbank to extract PLPs. We assume a sampling frequency equal to $16kHz$, so the filterbank is composed of 21 filters.

the squared Fourier Transform of the signal [36], as used in equation (3.16). In acoustics, a critical band is defined as the range of frequencies perceived as the same frequency by the human ear, because they activate the same region of the basilar membrane [37]. Under this viewpoint, human cochlea acts based on a set of filters called auditory filters.

The next step is to pass the output of (3.16) through a pre-emphasis filter aiming at simulating the non-uniform perception of loudness in different frequencies, as this is reflected by the equal-loudness-level contours. An equal-loudness-level contour [38] contains acoustic pressure points which are perceived as equally loud under different frequencies by a human listener. The unit of measurement of loudness is *phon* and, by definition, two sinusoids of equal *phons* are equally loud. Equal-loudness-level contours, according to ISO 226:2003 [39], are illustrated in Figure 3.7.

The introduction of the equal-loudness-level contours during audio feature extraction is of great importance when trying to find features simulating the human auditory system, since those contours are considered to reflect the frequency characteristics of this system [38]. The filter in use simulates the sinsitivity of human hearing at about $40dB$ and is described, in the frequency domain, by the equation

$$E(\omega) = \begin{cases} \dfrac{\omega^4\left(\omega^2 + 56.8\cdot 10^6\right)}{\left(\omega^2 + 6.3\cdot 10^6\right)^2\left(\omega^2 + 0.38\cdot 10^9\right)} & , Fs \leq 10kHz \\ \dfrac{\omega^4\left(\omega^2 + 56.8\cdot 10^6\right)}{\left(\omega^2 + 6.3\cdot 10^6\right)^2\left(\omega^2 + 0.38\cdot 10^9\right)\left(\omega^6 + 9.58\cdot 10^{26}\right)} & , Fs > 10kHz \end{cases}, \tag{3.20}$$

where $\omega = 2\pi f$.

The weights introduced by this pre-emphasis filter can be incorporated into the weights existing becuase of the filterbank. Intuitively, the result is a new filterbank, where the shape of each filter is as depicted in Figure 3.6, but the height is increased as the frequency increases. Additionally, the values for the first and last filters are set equal to their

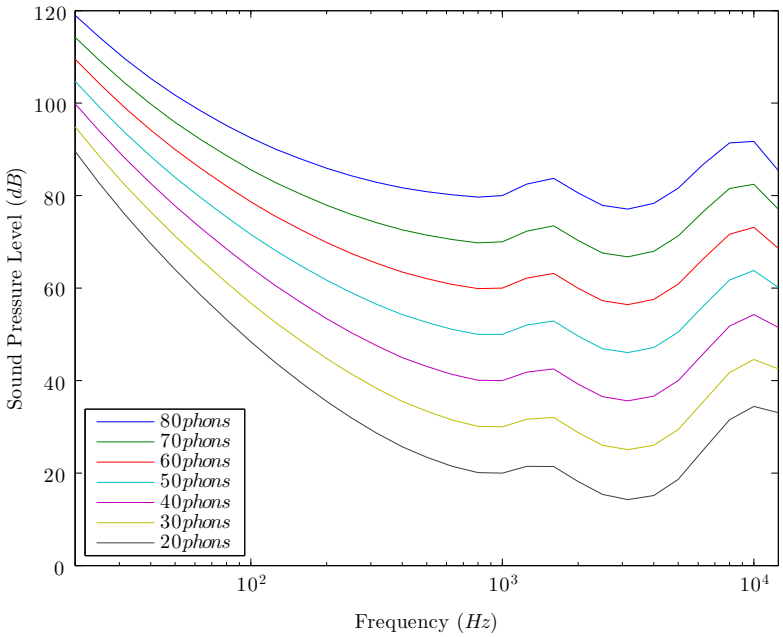


Figure 3.7: Equal-loudness-level contours.

neighboring ones. So, equation (3.16) is now transformed into equation (3.21).

$$\tilde{G}_i(j) = \begin{cases} \tilde{G}_i(2) & , j = 1 \\ \sum_{k=0}^{N/2} \left\{ |S_i[k]|^2 \cdot |E[k]H^j[k]| \right\} & , 2 \leq j \leq Q-1 \\ \tilde{G}_i(Q-1) & , j = Q \end{cases} \tag{3.21}$$

Apart from the variable sensitivity of human ear depending on the frequency, there are psychophysical laws which relate the real intensity of a sound source with its loudness as perceived by a human. Those laws are part of a general group of emperical laws that relate an activation with the psychological intensity that this activation triggers. [40]. As far as the sound intensity is concerned, the corresponding psychophysical law is expressed thorugh an exponential function where the exponent is equal to about 0.3. This perceived loudness is measured with the so-called *sones*. To approximate this law during PLP extraction the coefficients $\tilde{G}_i(j)$ are converted to $\Phi_i(j)$:

$$\Phi_i(j) = \left(\tilde{G}_i(j) \right)^{0.33} \tag{3.22}$$

Finally, we need to estimate the linear prediction coefficients for the vector Φ_i. To this end, the autocorrelation method, and specifically the Levinson-Durbin [3] is used. Thus, the autocorrelation needs to be calculated.

As already stated, Φ_i is estimated as the power spectrum of the signal $s_i(n)$, after it has been decomposed into critical bands and it has been further processed to simulate specific characteristics of the human auditory system. But since the highest frequency of the filterbank is the Nyquist frequency, and because of the symmetry of DFT, the entire

power spectrum $\tilde{\Phi}_i$ is equal to:

$$\tilde{\Phi}_i = [\Phi_i(1), \Phi_i(2), \cdots, \Phi_i(Q-1), \Phi_i(Q), \Phi_i(Q-1), \cdots, \Phi_i(2)]$$

We note that because of (3.21), we have that $\Phi_i(2) = \Phi_i(1)$ and $\Phi_i(Q) = \Phi_i(Q-1)$.

Now, according to the Wiener-Khinchin Theorem [41], the power spectrum of a function is is equal to the Fourier Transfer of the autocorrelation of the function. Therefore, in order to calculate the desired autocorrelation, we can just use the IDFT of $\tilde{\Phi}_i$.

Let the LPC modelling be of order p. Fomally that means that the speech signal $s_i(n)$, as perceived by a human, is generated by the difference equation (3.23), where $u_i(n)$ is the excitation source and where the prediction coefficients $a_{i,k}$ are estimated through the Levinson-Durbin algorithm.

$$s_i(n) = \sum_{k=1}^{p} a_{i,k} s_i(n-k) + G_i u_i(n) \tag{3.23}$$

Those coefficients can easily be converted to a set of cepstral coefficients $\{\hat{\phi}_i(j)\}, j = 0, 1, \cdots, p$ which is exactly the set of PLPs for the signal s_i. To that end, the recursive equation (3.24) is used [3].

$$\hat{\phi}_i(j) = \begin{cases} \log G_i & , j = 0 \\ a_{i,j} + \sum_{k=1}^{j-1} \left(\frac{k}{j}\right) \hat{\phi}_i(k) a_{i,j-k} & , 1 \leq j \leq p \end{cases} \tag{3.24}$$

3.2.2 Robust Features Using RASTA Analysis

As we have already seen, the right spectral analysis of a speech signal can lead to the successful recognition of the linguistic information carried by the signal because the resulting spectral features reflect the shape of the vocal tract and, thus, the sounds produced through it. In order to move towards the filed of robust speech recognition, we need to study the unique characteristics of the spectral features of speech, when compared to spectral features extracted by different sounds (such as noise). The idea of RelAtive SpecTrAl (RASTA) analysis [42] is moving exaxtly towards this direction.

RASTA analysis is a precursor of features based on the Modulation Frequency (MF) and the Modulation Spectrum, ideas we are going to see again in section **??**. MF is the frequency with which the spectral characteristics of the signal are changing through time, an important notion for speech recognition, since speech perception depends on the spectral variations, that is on how the spectral features of a sound change compared to its neighbors [42]. Human hearing is more sensitive into changes around $4Hz$, while modulation frequencies above $16Hz$ are almost negligible for speech intelligibility [43]. The above, in combination with the fact that articulators are moving in a rate lying in the range $[1Hz, 13Hz]$ [44], give direct clues for the range of modulation frequencies on which we should focus.

The idea of RASTA analysis is based on filtering the signal in a suitable domain so that only the necessery for recognition parts of the signal pass through the filter, while those that vary slower or faster than the typical speech variations are filtered out. When RASTA analysis is combined with the PLPs, as suggested in [42], the resulting features are called RASTA-PLPs. In that case, RASTA analysis takes place after the spectral coefficients of

the critical bands in the Bark scale have been extracted, according to (3.16), and before the application of the pre-emphasis filter to simulate the equal-loudness-level contours. It is comprised of three steps: the non-linear transformation of the spectral coefficients into a new domain (let the transformation T), their filtering so that the frequency bands of low interest are filtered out, and a new transformation of the result through the inverse T^{-1} (or a similar transformation). It is noted that RASTA analysis is general and does not strictly depend on the specific features used. For instance, RASTA filtering can be applied during the MFCC extraction, before we pass to the cepstral domain, that is after the step described by equation (3.12).

We will assume for the rest of our analysis that the distance between consecutive windows is equal to $10msec$, which means that the windowing rate is ${}^{1}/_{10msec} = 100Hz$. Therefore, we examine modulation frequencies in the range $[0Hz, 50Hz]$. The transfer function of the filter proposed in [42] is

$$H(z) = 0.1z^4 \frac{2 + z^{-1} - z^{-3} - 2z^{-4}}{1 - 0.94z^{-1}}\,. \tag{3.25}$$

It is interesting to notice the strong relationship between RASTA analysis and the computation of the Δ coefficients, since the numerator in (3.25) closely matches the transfer funciton of the filter that computes the Δ coefficients. Specifically, for $M = 2$, using (3.14) we get

$$\begin{aligned} \Delta x(k) &= \frac{1}{10}[-2x(k-2) - x(k-1) + x(k+1) + 2x(k+2)] \\ \Rightarrow \Delta X(z) &= 0.1[-2z^{-2}X(z) - z^{-1}X(z) + zX(z) + 2z^2X(z)] \\ \Rightarrow \tilde{H}(z) &= \frac{\Delta X(z)}{X(z)} = 0.1z^2[-2 - z^{-1} + z^{-3} + 2z^{-4}]\,. \end{aligned} \tag{3.26}$$

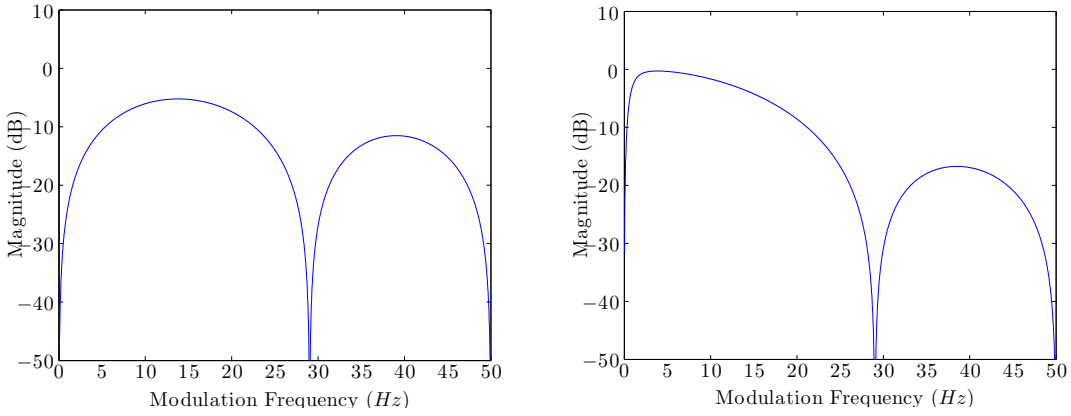
In fact, the motivation behind RASTA analysis were indeed the Δ coefficients and their ability to partially eliminate the negative effects of the convolutive distortions [42]. However, Δ coefficients cannot successfully be used on their own; so they are used in combination with the initial, static coefficients, which, as we have seen, are sensitive to distortions. The frequency responses of $H(z)$ and $\tilde{H}(z)$ are illustrated in Figure 3.8.

We can observe that the filter for RASTA analysis has a relatively fixed response in the range $[1Hz, 10Hz]$, with the maximum being close to $4Hz$, being in accordance with the acoustic theory analyzed above. In constrast, during the extraction of the Δ coefficients, it seems that a few modulation frequencies are favored, which are not close to the ones that characterize human speech. As a result, the linguistic content of the signal may be distorted.

The question that has to be answered is which is a suitable transformation T. In the case of convolutive noise, (e.g. due to reverberation or to channel/microphone variabilities), then a reasonable option is to filter in the logarithmic domain. There, those distortions appear as additive constants; thus it is easy to eliminate their effect. Therefore, every spectral coefficient is passed through a log function, RASTA filtering is applied and finally we apply the inverse logarithic function, which is just the exponential $\exp(\cdot)$.

If, however, there is also additive noise, then the logarithmic domain is not suitable. In that case the suggested transformation is

$$y = \log(1 + Jx)\,, \tag{3.27}$$



(i) (ii)

Figure 3.8: (i)Frequency response for the filter $\tilde{H}(z)$ used to estimate the Δ coefficients. (ii)Frequency response for the filter $H(z)$ used for RASTA analysis.

where J is a constant. This domain behaves as linear when $J \ll 1$ and as logarithmic when $J \gg 1$. The optimal choice for J is the one that assigns the largest possible part of the clean signal to the logarithmic part of the non-linearity and the largest part of the noise to the linear part. The inverse of (3.27) is

$$x = \frac{e^y - 1}{J}\,. \tag{3.28}$$

However, this function is not guaranteed to always being positive. So, in practice, the inverse transform is approximated by

$$x = \frac{e^y}{J}\,. \tag{3.29}$$

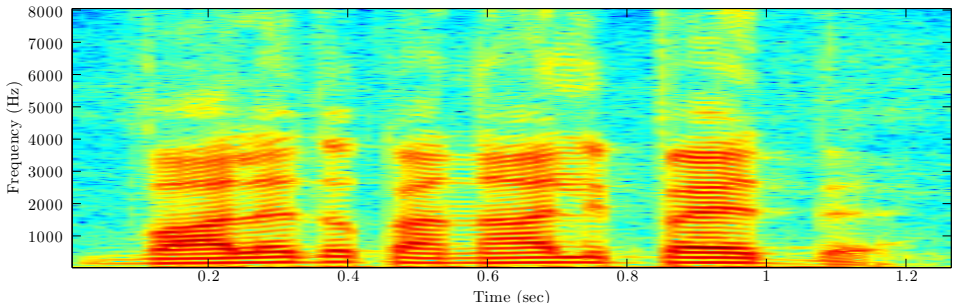
When the equations (3.27), (3.29) are used for RASTA filtering, then the process is called J-RASTA or Lin-Log RASTA.

The effect of PLP and RASTA-PLP analysis is obvious in Figure 3.9, where the initial spectrogram of a signal, together with the corresponding spectrograms after PLP and RASTA-PLP (of course before the conversion of the spectral features to the cepstral domain), are illustrated. First of all, it is apparent that going from *Hertz* scale to *Bark* scale, "more space" is devoted to the low frequencies. Additionally, we can see that after RASTA filtering the temporal evolution is much smoother with fewer details. The amount information that has remained after filtering is expected to capture everything needed for the recognition of the linguistic content.

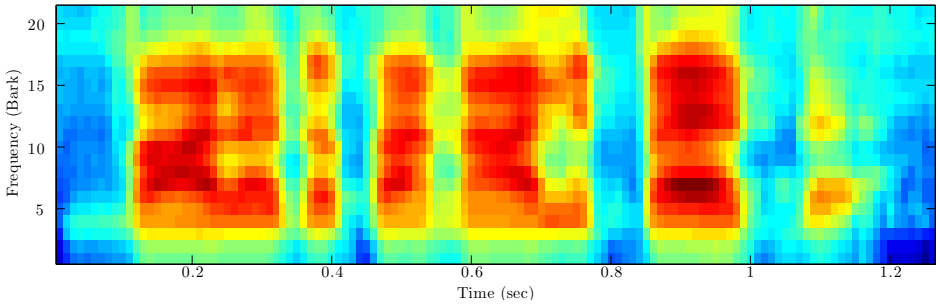
3.3 Power Normalized Cepstral Coefficients (PNCCs)

One of the numerous ideas suggested for robust speech recognition has led to a feature set known as Power Normalized Cepstral Coefficients (PNCCs) [45]. The analysis follwed here is mainly based on [46].

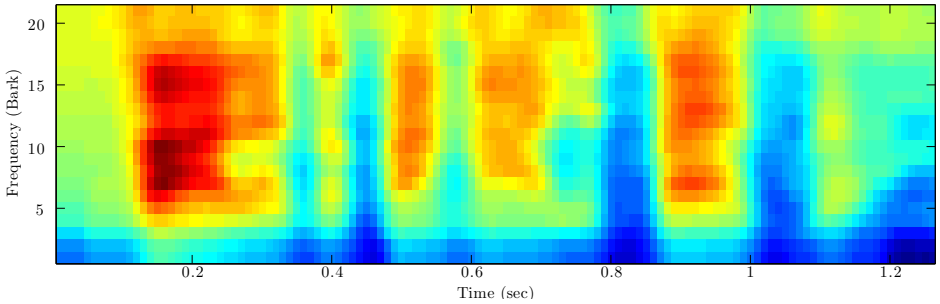
The extraction of PNCCs is a procedure similar to the MFCC extraction; it has, however, a few fundamental differences. Initially, the desired signal passes through a proemphasis system with transfer function given by (3.1) (where $\tilde{a} = 0.97$) and is segmented into



(i) .



(ii) PLP .



(iii) RASTA-PLP .

Figure 3.9: Spectrogram of a speech signal before and after PLP and RASTA-PLP analysis.

overlapped frames using Hamming windows. For each frame, say $s_i[n]$, we get, through N-point DFT, the spectrum $S_i[k]$, which is analyzed by a filterbank $H^j[k]$, $j = 1, \cdots, Q$, to get the coefficients $G_i(j)$:

$$G_i(j) = \sum_{k=0}^{N/2} \left\{ \left| S_i[k] \cdot H^j[k] \right|^2 \right\} \tag{3.30}$$

Instead of the triangular filters used for MFCCs or the trapezoid filters used for PLPs, here we use a filterbank of gammatone filters, which was propesed in [47] to model the human cochlea. The impulse response $g(t)$ of a gammatone filter is generally given as

$$g(t) = at^{n-1} e^{-2\pi bt} \cos\left(2\pi f_c t + \phi\right), \tag{3.31}$$

where f_c is the central frequency of the filter. The parameter b defines the length of the impulse response of the filter, so its bandwidth as well, while n is the order of the filter

and defines the slope of its "tails". In the particular case, the values $n = 4$ and $b = 1.019ERB(f_c)$ are used. ERB (Equivalent Rectangular Bandwidth) has been introduced to estimate the bandwidth of a non-symmetric IIR filter and is approximated as

$$ERB(f_c) = 24.7\frac{4.37f_c}{1000} + 1\,. \tag{3.32}$$

The generated filterbank, where filters are linearly distributed in the ERB scale, is illustrated in Figure 3.10, for 24 filters and sampling frequency $F_s = 16kHz$, when the filters are normalized with respect to their height.

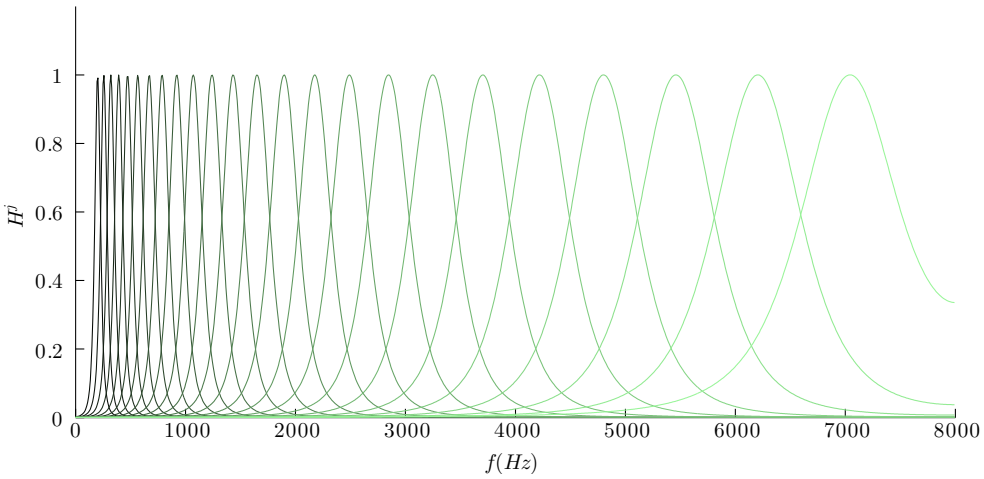


Figure 3.10: Gammatone filterbank to extract PNCCs. We assume a sampling frequency equal to $16kHz$, while the filterbank is composed of 24 filters. The central frequency of the first filter is $200Hz$.

The coefficients $G_i(j)$ are normalized with respect to G_{peak}, which is equal to the 95th percentile of the elements $\left\{\sum_j G_i(j)\right\}_i$:

$$\tilde{G}_i(j) = g_0\frac{G_i(j)}{G_{peak}}\,, \tag{3.33}$$

where g_0 is a constant used to have a reasonable dynamic range.

The next stages comprise a process called Power Bias Subtraction (PBS). PBS aims at the maximization of the sharpness of the power spectrum in different frequencies (after it is passed through the filterbank), since human hearing is more sensitive to spectral changes, when compared to the relatively stationary spectral background. If PBS doesn't take place, the resulting features are called Simple PNCCs (SPNCCs) [48]. The first step of PBS is to find the mean-duration power through the running average of $\tilde{G}_i(j)$:

$$Q_i(j) = \frac{1}{2C+1}\sum_{j'=j-C}^{j+C}\tilde{G}_i(j') \tag{3.34}$$

In order to reduce the effect of the spectral background, a constant q_0 is subtracted from the coefficients $Q_i(j)$. However, to avoid very small values, there is a threshold q_f. Thus, we get the coefficients

$$\tilde{Q}_i(j) = \max\left\{Q_i(j) - q_0, q_f\right\}\,. \tag{3.35}$$

The exact values of q_0 and q_f are analytically estimated through the proposed searching algorithm in [46], which is based on the ratio of the arithmetic mean over the geometric mean of the power coefficients $Q_i(j)$. A final smoothing of the initial spectral values takes place:

$$P_i(j) = \left(\frac{1}{2L+1} \sum_{j'=j-L}^{j+L} \frac{\tilde{Q}_i(j')}{Q_i(j')} \right) \tilde{G}_i(j) \tag{3.36}$$

Finally, after the coefficients $P_i(j)$ (which are identical to $\tilde{G}_i(j)$ for the case of SPNCCs) pass through a non-linearity, DCT is applied and the first N_c coefficients are used, where typically $N_c = 13$, like in the MFCC case. The non-linear function used is not the logarithm, like in MFCCs, but an exponential function where the exponent is in the range $(0, 1)$. This exponent, which for PLPs was equal to 0.33, is here defined equal to $^1/_{15}$.

Bibliography

[1] F. Rumsey and T. McCormick, *Sound and Recording.* Focal Press, 2009.

[2] D. Jurafsky and J. Martin, *Speech and Language Processing.* Prentice Hall, 2008.

[3] L. R. Rabiner and R. W. Schafer, *Digital Processing of Speech Signals.* Prentice Hall, 1978.

[4] T. Hori and A. Nakamura, *Speech Recognition Algorithms Using Weighted Finite-State Transducers.* Morgan & Claypool, 2013.

[5] G. A. Fink, *Markov models for pattern recognition: from theory to applications.* Springer, 2008.

[6] L. Rabiner and B.-H. Juang, *Fundamentals of Speech Recognition.* Prentice Hall, 1993.

[7] J. Pinto, B. Yegnanarayana, H. Hermansky, and M. M. Doss, "Exploiting contextual information for improved phoneme recognition," in *Acoustics, Speech and Signal Processing, 2008. ICASSP 2008. IEEE International Conference on*, pp. 4449–4452, IEEE, 2008.

[8] C. M. Bishop, *Pattern recognition and machine learning.* Springer, 2006.

[9] S. J. Young, "The general use of tying in phoneme-based HMM speech recognisers," in *Acoustics, Speech, and Signal Processing, 1992. ICASSP-92., 1992 IEEE International Conference on*, vol. 1, pp. 569–572, IEEE, 1992.

[10] X. D. Huang and M. A. Jack, "Semi-continuous hidden markov models for speech signals," *Computer Speech & Language*, vol. 3, no. 3, pp. 239–251, 1989.

[11] S. J. Young, J. J. Odell, and P. C. Woodland, "Tree-based state tying for high accuracy acoustic modelling," in *Proceedings of the workshop on Human Language Technology*, pp. 307–312, Association for Computational Linguistics, 1994.

[12] J.-P. Hosom, *Automatic Time Alignment of Phonemes Using Acoustic-Phonetic nformation.* PhD thesis, Oregon Graduate Institute of Science and Technology, 2000.

[13] A. Katsamanis, I. Rodomagoulakis, G. Potamianos, P. Maragos, and A. Tsiami, "Robust far-field spoken command recognition for home automation combining adaptation and multichannel processing," in *Acoustics, Speech and Signal Processing (ICASSP), 2014 IEEE International Conference on*, pp. 5547–5551, IEEE, 2014.

[14] C. E. Shannon, "A mathematical theory of communication," *Bell System Technical Journal*, vol. 27, pp. 379–423, 623–656, 1948.

[15] I. H. Witten and T. C. Bell, "The zero-frequency problem: Estimating the probabilities of novel events in adaptive text compression," *Information Theory, IEEE Transactions on*, vol. 37, no. 4, pp. 1085–1094, 1991.

[16] S. F. Chen and J. Goodman, "An empirical study of smoothing techniques for language modeling," in *Proceedings of the 34th annual meeting on Association for Computational Linguistics*, pp. 310–318, Association for Computational Linguistics, 1996.

[17] G. Donaj and Z. Kačič, "The use of several language models and its impact on word insertion penalty in lvcsr," in *Speech and Computer*, pp. 354–361, Springer, 2013.

[18] M. Sipser, *Introduction to the Theory of Computation*. Thomson Course Technology, 2006.

[19] W. Kuich and A. Salomaa, *Semirings, automata, languages*. Springer Verlag, 1986.

[20] M. Mohri, F. Pereira, and M. Riley, "Weighted finite-state transducers in speech recognition," *Computer Speech & Language*, vol. 16, no. 1, pp. 69–88, 2002.

[21] M. Mohri, F. Pereira, and M. Riley, "The design principles of a weighted finite-state transducer library," *Theoretical Computer Science*, vol. 231, no. 1, pp. 17–32, 2000.

[22] C. Allauzen, M. Riley, J. Schalkwyk, W. Skut, and M. Mohri, "Openfst: A general and efficient weighted finite-state transducer library," in *Implementation and Application of Automata*, pp. 11–23, Springer, 2007.

[23] M. Mohri, "Weighted automata algorithms," in *Handbook of weighted automata*, pp. 213–254, Springer, 2009.

[24] M. Riley, F. Pereira, and M. Mohri, "Transducer composition for context-dependent network expansion.," in *EUROSPEECH*, pp. 1427–1430, 1997.

[25] S. B. Davis and P. Mermelstein, "Comparison of parametric representations for monosyllabic word recognition in continuously spoken sentences," *Acoustics, Speech and Signal Processing, IEEE Transactions on*, vol. 28, no. 4, pp. 357–366, 1980.

[26] T. Ganchev, N. Fakotakis, and G. Kokkinakis, "Comparative evaluation of various mfcc implementations on the speaker verification task," in *Proceedings of the SPECOM*, vol. 1, pp. 191–194, 2005.

[27] M. Slaney, "Auditory toolbox version 2. interval research corporation," *Indiana: Purdue University*, vol. 2010, pp. 1998–010, 1998.

[28] M. Unser, "On the approximation of the discrete Karhunen-Love transform for stationary processes," *Signal Processing*, vol. 7, no. 3, pp. 231–249, 1984.

[29] K. K. Paliwal, "Decorrelated and liftered filter-bank energies for robust speech recognition.," in *Eurospeech*, vol. 99, pp. 85–88, 1999.

[30] R. Schwartz, T. Anastasakos, F. Kubala, J. Makhoul, L. Nguyen, and G. Zavaliagkos, "Comparative experiments on large vocabulary speech recognition," in *Proceedings of the workshop on Human Language Technology*, pp. 75–80, Association for Computational Linguistics, 1993.

[31] M. Wölfel and J. McDonough, *Distant Speech Recognition*. Wiley, 2009.

[32] S. Furui, "Speaker-independent isolated word recognition using dynamic features of speech spectrum," *Acoustics, Speech and Signal Processing, IEEE Transactions on*, vol. 34, no. 1, pp. 52–59, 1986.

[33] K. Kumar, C. Kim, and R. M. Stern, "Delta-spectral cepstral coefficients for robust speech recognition," in *Acoustics, Speech and Signal Processing (ICASSP), 2011 IEEE International Conference on*, pp. 4784–4787, IEEE, 2011.

[34] H. Hermansky, "Perceptual linear predictive (plp) analysis of speech," *the Journal of the Acoustical Society of America*, vol. 87, no. 4, pp. 1738–1752, 1990.

[35] H. Hermansky, "Should recognizers have ears?," *Speech communication*, vol. 25, no. 1, pp. 3–27, 1998.

[36] P. Stoica and R. L. Moses, *Spectral analysis of signals*. Pearson/Prentice Hall Upper Saddle River, NJ, 2005.

[37] S. A. Gelfand, *Hearing: An introduction to psychological and physiological acoustics*. CRC Press, 2009.

[38] Y. Suzuki and H. Takeshima, "Equal-loudness-level contours for pure tones," *The Journal of the Acoustical Society of America*, vol. 116, no. 2, pp. 918–933, 2004.

[39] ISO, "226: 2003: Acoustics–normal equal-loudness-level contours," *International Organization for Standardization*, 2003.

[40] S. S. Stevens, "On the psychophysical law.," *Psychological review*, vol. 64, no. 3, p. 153, 1957.

[41] D. W. Ricker, *Echo signal processing*, vol. 725. Springer Science & Business Media, 2012.

[42] H. Hermansky and N. Morgan, "Rasta processing of speech," *Speech and Audio Processing, IEEE Transactions on*, vol. 2, no. 4, pp. 578–589, 1994.

[43] R. Drullman, J. M. Festen, and R. Plomp, "Effect of temporal envelope smearing on speech reception," *The Journal of the Acoustical Society of America*, vol. 95, no. 2, pp. 1053–1064, 1994.

[44] C. L. Smith, C. P. Browman, R. S. McGowan, and B. Kay, "Extracting dynamic parameters from speech movement data," *The Journal of the Acoustical Society of America*, vol. 93, no. 3, pp. 1580–1588, 1993.

[45] C. Kim and R. M. Stern, "Feature extraction for robust speech recognition using a power-law nonlinearity and power-bias subtraction.," in *INTERSPEECH*, pp. 28–31, 2009.

[46] C. Kim and R. M. Stern, "Feature extraction for robust speech recognition based on maximizing the sharpness of the power distribution and on power flooring," in *Acoustics Speech and Signal Processing (ICASSP), 2010 IEEE International Conference on*, pp. 4574–4577, IEEE, 2010.

[47] R. D. Patterson, K. Robinson, I. Holdsworth, D. McKeown, C. Zhang, and M. Allerhand, "Complex sounds and auditory images," in *Auditory Physiology and Perception: Proceedings of the 9th International Symposium on Hearing Held in Carcens, France on 9-14 June 1991*, no. 83, p. 429, Pergamon, 1992.

[48] C. Kim and R. M. Stern, "Power-normalized cepstral coefficients (pncc) for robust speech recognition," *IEEE/ACM Transactions on Audio, Speech and Language Processing (TASLP)*, vol. 24, no. 7, pp. 1315–1329, 2016.