Deep Neural Network approach to Automatic Speech Recognition

by

Marcin Sikora

Submitted to the The Faculty of Mathematics and Computer Science
in partial fulfillment of the requirements for the degree of

Master of Science in Computer Science

at the

ADAM MICKIEWICZ UNIVERSITY IN POZNAŃ

November 2016

Author . . . .
The Faculty of Mathematics and Computer Science
November 28, 2016

Certified by . . . .
Krzysztof Jassem
Associate Professor
Thesis Supervisor

Zastosowanie głębokich sieci neuronowych w rozpoznawaniu mowy

Marcin Sikora

Przedłożone na Wydziale Matematyki i Informatyki jako jeden z warunków ukończenia studiów i uzyskania tytułu zawodowego

magistra Informatyki

na

UNIWESYTECIE IM. ADAMA MICKIEWICZA W POZNANIU

Październik 2016

Autor...

Wydział Matematyki i Informatyki
28. października 2016. roku

Zatwierdzone przez ....

Krzysztof Jassem
Prof. UAM
Promotor pracy

Deep Neural Network approach to Automatic Speech Recognition

by

Marcin Sikora

Submitted to the The Faculty of Mathematics and Computer Science
on November 28, 2016, in partial fulfillment of the
requirements for the degree of
Master of Science in Computer Science

Abstract

The aim of this master thesis is to study the performance of Speech Recognition systems and in particular, measure the gains obtained from using speed-perturbation, noise addition and LSTM-RNN models. The ASR system has been built using the open-source Kaldi toolkit; four HMM-GMM models have been trained to measure the speed-perturbation folding count impact on the Word Error Rate (WER), another four LSTM-RNN models have been trained to check the impact of SNR range choice of training data on the WER. All systems have been evaluated on separate clean and noisy evaluation corpora. The results have been compared to the state-of-the-art solutions reported in various papers. Additionally, two novel randomization techniques have been suggested and tested with good results - both introduced a measurable WER decrease. They can be easily employed in any future Speech Recognition systems, as they are Acoustic Model agnostic and do not increase the computational complexity of the training process.

Thesis Supervisor: Krzysztof Jassem
Title: Associate Professor

Zastosowanie głębokich sieci neuronowych w rozpoznawaniu mowy

Marcin Sikora

Przedłożone na Wydziale Matematyki i Informatyki
28. października 2016 roku, jako jeden z warunków ukończenia studiów
i uzyskania tytułu zawodowego
magistra inżyniera Informatyki

Streszczenie

Celem tej pracy magisterskiej było zbadanie jakości systemu Rozpoznawania Mowy, a w szczególności, zmierzenie zysków wynikających z użycia perturbacji szybkości, wzbogacania szumem oraz modeli LSTM-RNN. System Automatycznego Rozpoznawania Mowy został zbudowany przy użyciu otwartego zestawu narzędzi Kaldi. Cztery modele HMM-GMM zostały wytrenowane w celu sprawdzenia wpływu wyboru parametrów perturbacji szybkości na metrykę Word Error Rate (WER); kolejne cztery modele LSTM-RNN zostały wytrenowane w celu zmierzenia wpływu wyboru zakresu SNR danych trenujących na WER. Wszystkie wytrenowane systemy zostały zewaluowane w oparciu o czyste, jak i zaszumione korpusy. Wyniki zostały porównane z rozwiązaniami raportowanymi w aktualnych publikacjach. Dodatkowo, zaproponowano dwie nowe techniki związane z wprowadzeniem losowości do zbiorów trenujących - obie przyczyniły się do zauważalnego spadku WER. Techniki te mogą z powodzeniem zostać użyte w każdym przyszłym systemie Rozpoznawania Mowy, jako że nie są zależne od wykorzystywanej architektury Modelu Akustycznego, a ich wykorzystanie nie zwiększa złożoności obliczeniowej procesu trenowania.

Promotor pracy: Prof. UAM Krzysztof Jassem

Acknowledgments

The author would like to thank his thesis supervisor, Prof. Krzysztof Jassem, for all the input on the thesis structure and content; Samsung R&D Institute Poland for providing the access to the training corpora and servers without which this thesis could not be finished; his colleagues from AMU and Samsung R&D Institute Poland for the inspiration in research and words of advice; his wife for the patience and love provided during the periods of intense work on the thesis; his family for enabling this thesis to come into existence in the first place and last, but not least, his friends for kind words of encouragement, when the destination seemed to be too distant to reach.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Problem statement

Out of the number of possible human-computer interfaces, the voice has always been regarded as the most natural form of interaction. Voice communication has several merits, which make it outperform other input methods:

- easy to use - does not require any prior knowledge about the input method,
- does not require touch or visual interaction, leaving the hands and vision free for other tasks,
- fast - an average human can speak 140 words/minute[32], comparing to typing 40 words/minute[14],
- cheap - in it's most basic form, it requires only a microphone and a computing device.

The first stage of all voice interfaces is the Automatic Speech Recognition (ASR) module. Its good performance is substantial to the quality of the voice interface as the whole. Thus, a lot of effort has been put into the research in this area. For a long period of time, the ASR performance was insufficient to build a usable voice interface - either the system latency was too large to use comfortably or the misrecognitions caused the system to fail too often. This has changed recently, when the Neural

Networks became increasingly feasible to train[4]. Several new architectures have been proposed and they have been successfully used in the ASR context[9].

In this thesis, the author has reviewed latest findings on Recurrent Neural Networks and, in particular, Long-Short-Term Memory cells used in ASR systems. A relatively small Polish speech corpus with 200k of utterances has been used as the training data-set. Two data enhancement techniques have been used to overcome the problem of data scarcity. Some additional modifications to these techniques have been evaluated.

1.2 Organisation of thesis

This thesis has been divided into six chapters:

Chapter 2 introduces the basic notions from Speech Acoustics and Digital Signal Processing in reference to the Speech Recognition problem. Several feature processing techniques are described in detail.

Chapter 3 explores the fundamentals of Acoustic Modeling in context of the ASR system. The intuition behind the described Acoustic Model architectures, namely Hidden Markov Model with Gaussian Mixture model and Recurrent Neural Network with Long-Short-Term Memory, is explained and additional theoretical background is provided.

Chapter 4 describes the experimental setup, which was designed to verify the research theses discussed in this thesis. This includes a breakdown of the developed ASR system components, an overview of the training and evaluation corpora and a short description of the used platform.

Chapter 5 contains the analysis of the results obtained in three different experiments related to the data augmentation and Neural Network training. The system performance is compared to the state-of-the-art solutions. Two novel techniques are evaluated and the results are interpreted.

Chapter 6 summarizes the thesis findings and the possible future research areas. A few insights concerning the training process are delivered as well.

Chapter 2

Introduction to Speech Acoustics

2.1 Speech generation

2.1.1 Sound wave

A *mechanical wave* is defined as a disturbance traveling through a medium. An important implication of this description is that the wave is in fact an energy transfer and no actual mass transport takes place. As such, the deformation of a medium is just a temporary state during the propagation of a wave. When the wave ceases, the particles return to their original position. Of three kinds of mechanical waves, the longitudinal wave is of interest to this thesis[1].

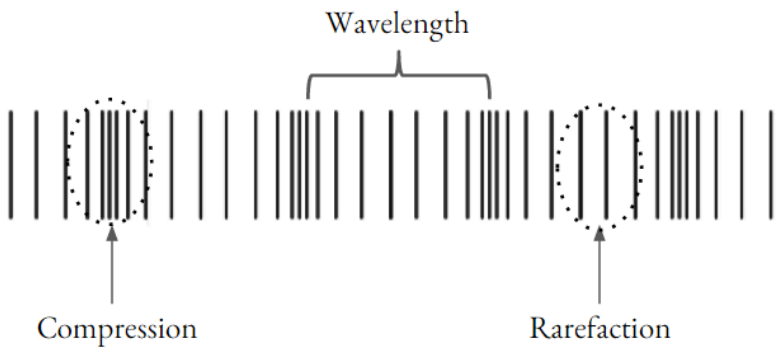

Figure 2-1: One-dimensional longitudinal wave.

[1]The other two are the transverse wave and the surface wave.

In *longitudinal waves*, the particles are displaced in the direction of the energy transfer (that is, parallelly). As the particles are pushed in one direction, they are creating a low density area (rarefaction) in their former, neutral position. Conversely, the area of high density is formed when these particles are being repelled back to their stable location (fig. 2-1). The wave which compresses and decompresses the medium in such a fashion is called the acoustic wave and is described by the following equation:

$$y(x,t) = y_m \cos\left(2\pi f\left(t - \frac{x}{c}\right)\right) \tag{2.1}$$

where:

- y is the displacement of the point on the traveling sound wave,
- x is the distance the point has traveled from the wave's source,
- t is the time elapsed,
- y_m is the amplitude of the oscillations,
- f is the frequency of the wave,
- c is the speed of the wave propagation (dependent on the medium).

When the frequency of the mechanical wave is contained within the human hearing range (approx. 20Hz - 20000Hz) and the medium through which the wave travels is the air, we call such wave the *sound wave.*

Two other parameters of sound wave, which are frequently used, are the *velocity* and the *pressure.* The equation for velocity can be derived from the longitudinal wave equation and the standard kinematics equation in the following way:

$$v(x,t) = \frac{\partial y}{\partial t}(x,t) \tag{2.2}$$

$$v(x,t) = 2\pi f y_m \cos\left(2\pi f\left(t - \frac{x}{c}\right) + \frac{\pi}{2}\right) \tag{2.3}$$

where v is the wave velocity.

The equation for pressure (2.5) can be derived from the Euler equation for the acoustic wave (2.4) [18]:

$$\rho\frac{\partial v}{\partial t} = -\frac{\partial p}{\partial x} \tag{2.4}$$

$$p(x,t) = 2\pi f c\rho y_m \cos\left(2\pi f\left(t-\frac{x}{c}\right)+\frac{\pi}{2}\right) \tag{2.5}$$

The ratio of pressure to velocity is called the *acoustic impedance* Z:

$$Z = \frac{p}{v} = c\rho \tag{2.6}$$

Acoustic impedance is the measure of the opposition of the medium to the applied pressure change.

2.1.2 Vocal tract

From the acoustic point of view, speech is a process, in which the sound wave is generated, filtered and modulated by an ensemble of articulators, folds and cavities, called the *vocal tract.* The position of the most important vocal tract elements is depicted in fig. 2-2.

Most kinds of speech sounds are initially generated by vocal cords[2]. The change of speed of gas caused by muscle action is directly influencing the pressure of moving air stream. This is described by the Bernoulli's equation:

$$p + \frac{\rho v^2}{2} + \rho g h = const \tag{2.7}$$

where:

- p is the pressure in the measured point,
- h is the elevation of the point above the ground,
- g is the gravitational acceleration.

As the density, elevation and gravitational acceleration values do not change during

[2]A notable exception is a “click” sound, which has no airstream mechanism involved. Instead, it is produced by so called *linguar initiation.* This kind of speech sounds are used in some of the African languages

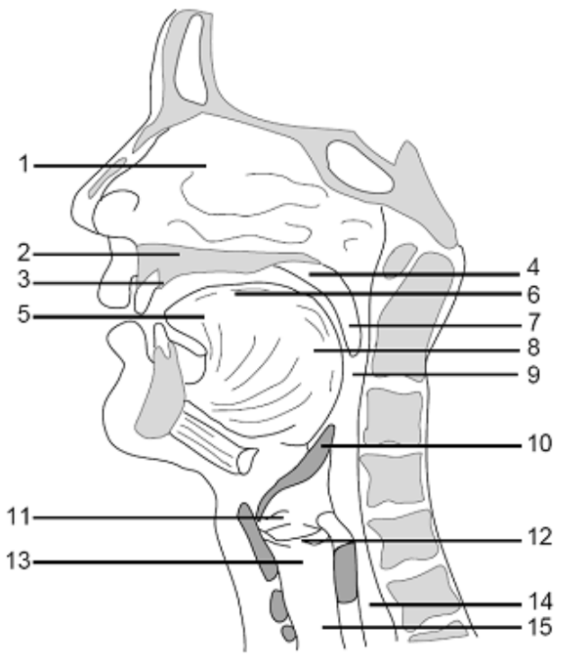

Figure 2-2: Voice tract overview[26]. The following voice tract parts have been marked: (1) nasal cavity, (2) hard palate, (3) gingiva, (4) soft palate, (5) tongue, (6) lingual septum, (7) palatine uvula, (8) lingual tonsils, (9) pharynx (10) epiglottis, (11) vestibular fold, (12) vocal folds, (13) larynx, (14) esophagus, (15) trachea.

speech generation, the equation can be simplified in the following words: an increase in gas velocity is causing a lowering in gas pressure.

While the air flow is being released through the trachea, the velocity of the stream is steadily increasing. This in turn leads to a drop in pressure and, if the vocal cords are flexed, gradual tightening of them, up to a full closure. As an effect, the air flow briefly halts, the velocity and pressure return to the previous values and the vocal cords open. This cycle produces a harmonic oscillation described with equations (2.3) and (2.5). By adjusting the tension of vocal folds, sounds of different pitches (i.e. frequencies) may be generated.

Most periodic sounds consist not only of the single base wave (or fundamental wave), but also of a series of waves called *harmonics*. The frequencies of these oscillations are always a multiple of the fundamental frequency; thus, a complex, harmonic speech sound is described by the following equation:

$$y_H(x,t) = \sum_{k=0}^{\infty} y_k \cos\left(2\pi f_0 k \left(t - \frac{x}{c}\right)\right) \tag{2.8}$$

where:

- f_0 is the fundamental frequency,
- k is the harmonic number.

An example of the complex periodic wave is presented in fig. 2-3.

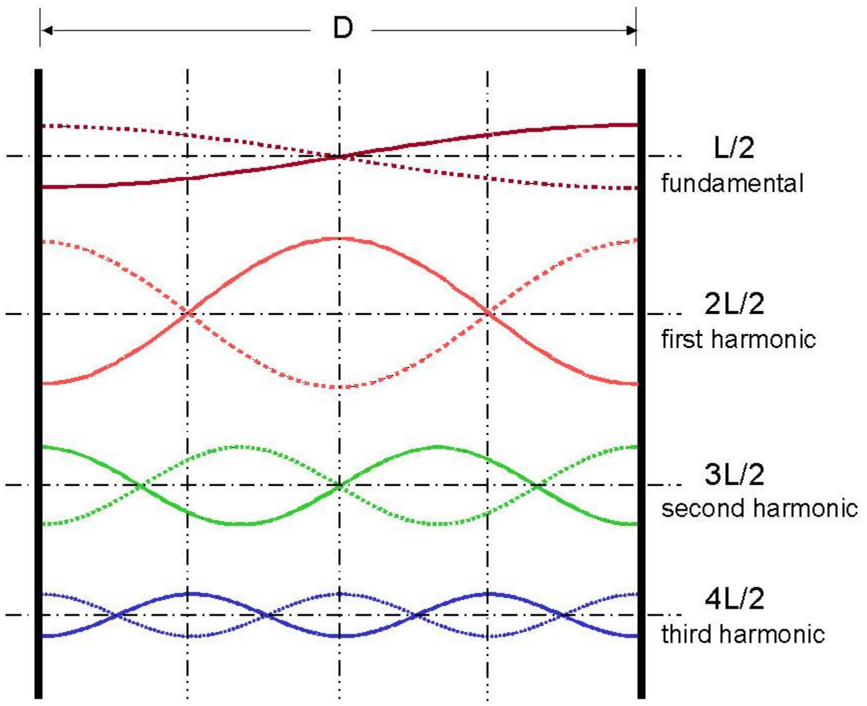

Figure 2-3: A complex periodic wave with base frequency $f_0 = \frac{2}{L}$

Apart from the base, harmonic wave, the *noise wave* is generated through a process called the *turbulent flow*. The laminar-turbulent transition is a process which is not fully understood and therefore, very difficult to describe . In simplification, it is caused by irregularities in medium and discrepancies in flow speed[8]. From the point of view of signal analysis, we can describe a noise wave as a superposition of infinite number of periodic signals, i.e.:

$$y_N(x,t) = \sum_{k=0}^{\infty} y_k \cos\left(2\pi f_k \left(t - \frac{x}{c}\right) + \varphi_k\right) \tag{2.9}$$

where φ_k is a phase shift (fig. 2-4). The noise wave is generated even when the vocal folds are not vibrating. The speech sound generated in that way is called unvoiced speech.

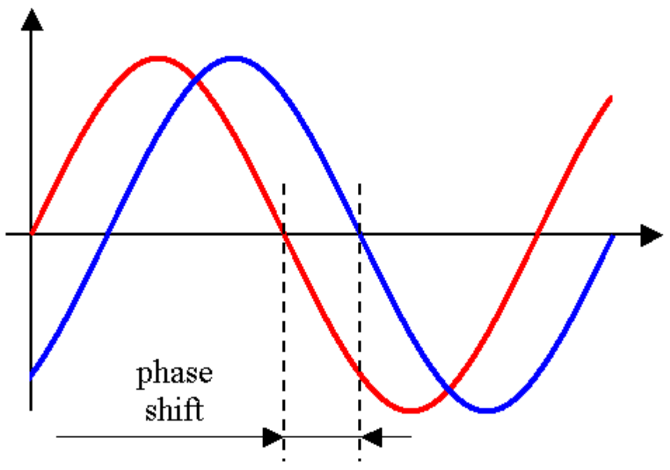

Figure 2-4: Phase shift in a periodic wave.

After the signal is produced, the wave propagates through the rest of vocal tract and is subjected to a number of processes, of which the main are:

- *filtration* - the vocal tract can be perceived as an ensemble of acoustic filters, which attenuates energy of certain frequencies, while amplifying the other,
- *modulation* - movement of the articulators causes a periodic change in filtration parameters: cutoff frequencies, the slope steepness, phase responses,
- *transient state* - a rapid change in the vocal tract state may produce a high-degree of noise components in speech sound.

2.2 Speech as a signal

2.2.1 Basic signal properties

When discussing speech (or, in general, sound) processing, we usually treat the analysed data as a *signal*. A signal is an abstract model of any physical quantity which is changing in time. There exist two main types of signals: analogue (continuous) and digital (discrete). In Digital Signal Processing (DSP), a signal is a discrete-time, quantized representation of the observed, real world process.

A process of converting an analogue signal to the digital form is called *sampling* and is performed by Analogue-Digital Converter (ADC) (fig. 2-5). The quality of sampling is described by various parameters. The *sampling frequency* f_s, also called the Nyquist-Shannon frequency, establish the time resolution of the signal and as a result, the upper bound of registered frequencies f_b:

$$f_b = \frac{f_s}{2} \tag{2.10}$$

Different sampling rates are used for various sound signals, e.g. music is usually sampled at 44.1kHz, while speech signal for speech recognition purposes is registered at 16kHz rate[3].

The other important sampling parameter is *bit depth* - the number of bits used to code each sample. Bit depth, which defines amplitude resolution of the digitized signal, directly influences the dynamic range of the sampled sound.

The recorded and digitized speech signal represents the change of pressure in time. The faintest pressure change that can be perceived by human is as small as $2*10^{-5}Pa$, whereas loud music causes approx. $20Pa$ pressure change. Due to the high range of amplitudes in speech signals, the amplitude is expressed in a value called the *sound*

[3]While speech signal contains frequencies higher than 8kHz, their inclusion does not increase speech recognition systems quality. For that reason, they are usually filtered out to save some data bandwidth.

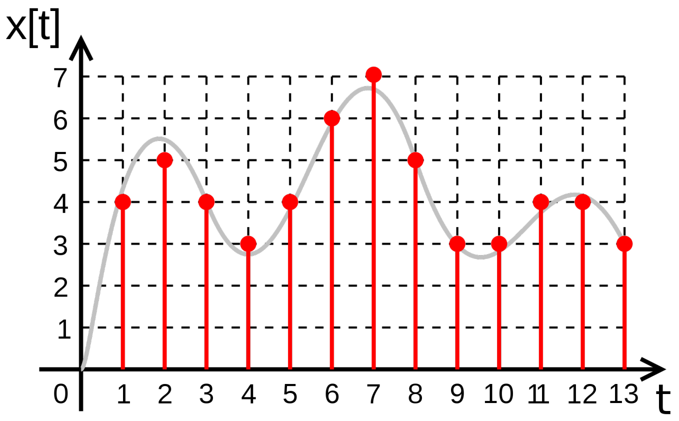

Figure 2-5: A continuous signal $x(t)$ discretized in time and with quantized amplitude values.

pressure level (SPL), measured in decibels (dB). SPL is defined as:

$$L_p = 10 \log \frac{< p^2 >}{p_0} \tag{2.11}$$

where $< p^2 >$ is the average square pressure and p_0 is the pressure of reference equal to the threshold of hearing $(2 * 10^{-5} Pa)$. Thus, the level of threshold of hearing is $0dB$ and SPL for loud music is $120dB$. A typical ADC sampling at 16 bit depth can represent the SPL range of $96dB$.

2.2.2 Short term power spectrum

A useful representation of a sampled speech signal is the plot of the amplitudes of each signal component against the frequency scale. To obtain such a representation, one might use Discrete Fourier Transform (DFT):

$$X[k] = \sum_{n=0}^{N-1} x[n] \exp(-j\frac{2\pi kn}{N}) \tag{2.12}$$

where:

- k is the frequency component,
- N is the number of frequency bands

- j is the imaginary unit[4].

The result of DFT is a series of complex numbers, which, in this form, are not very intuitive to use in DSP. Instead, the *Power Spectrum Density* (PSD) may be computed, which is defined as the squared magnitude of the complex spectrum:

$$P[k] = \frac{\Delta t}{T}(\Re(X[k])^2 + \Im(X[k])^2) \tag{2.13}$$

where:

- $\Re/\Im$ are real/imaginary parts of the complex spectrum,
- Δt is the sampling interval (inverse of the sampling frequency),
- T is the total sampling time.

Most useful information encoded in speech signal is contained in short fragments of a signal, so T is usually restricted to small values (in speech recognition, almost always $25ms$). When defined in this way, PSD introduces artifacts created by windowing with a rectangular window function. In signal processing, a *window* is a function which is used to "cut out" a fragment of signal in time domain. Such an operation has an impact on computed spectrum by deforming the signal in various ways[28]. Usually, a proper windowing function is chosen as a compromise between frequency resolution and the dynamic range. In speech processing, the most frequently used window function is the Hamming window:

$$w[n] = 0.54 - 0.46(\cos\frac{2\pi n}{N-1}) \tag{2.14}$$

When applied to eq. (2.12), we obtain a windowed DFT:

$$X_w[k] = \sum_{n=0}^{N-1} w[n]x[n]\exp(-j\frac{2\pi kn}{N}) \tag{2.15}$$

PSD computed using this modified DFT over a short period of time is called a *Short*

[4]Alternatively, denoted as i.

Term Power Spectrum (STPS). A simplified process of computing STPS is presented in fig. 2-6.

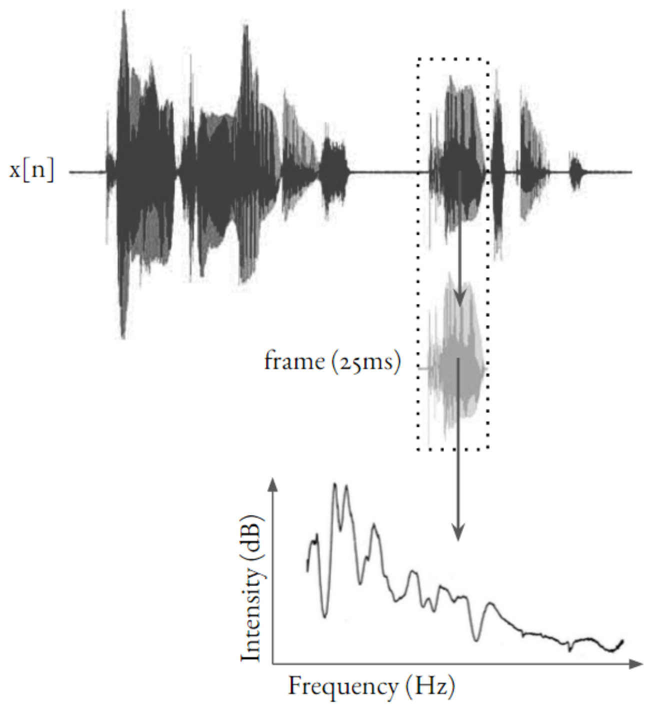

Figure 2-6: Framing a signal to obtain short time power spectrum.

2.2.3 Filterbanks

The idea behind STPS is to mimic the way human cochlea analyses the sound - the signal of each frequency stimulates different hair cells, which in turn send the information to the higher parts of neural system. The frequency resolution of cochlea is high in lower frequencies and decreases as the sound pitch increases. That correlates well with the musical aspect of human hearing - each octave is defined as equidistant in terms of pitch, where in fact octave change is caused by doubling (or halving) the frequency. Moreover, an experiment on human listeners exposed that to produce equal (perceived) pitch increments, the required change of interval is higher, than from following octaves[29]. To describe such phenomena, a special frequency scale called *mel scale* has been introduced. A formula used to map hertz to mel is:

$$m = 1127\ln(1 + \frac{f}{700}) \tag{2.16}$$

Revised frequency scale is presented in fig. 2-7. Note that the plot can be approximated quite accurately with a linear function up to 1000Hz.

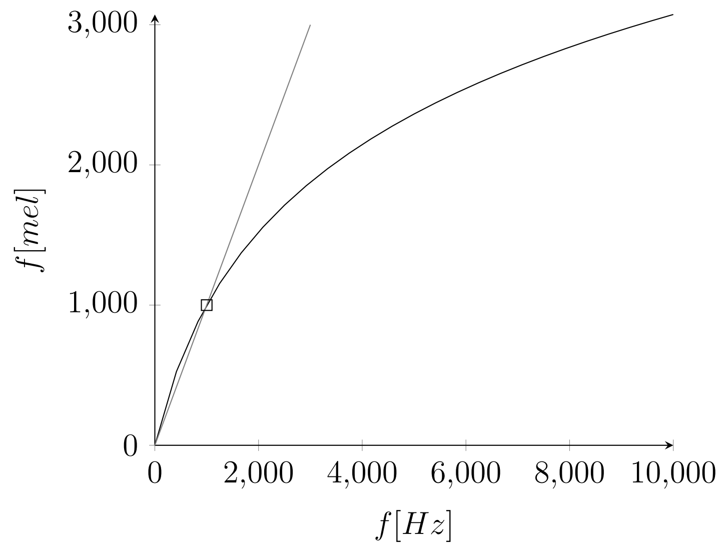

Figure 2-7: Mapping from hertz scale to mel scale. An asymptote has been included for reference.

Another phenomenon which manifests itself in the auditory system is that it does not analyse the sound in a discrete (i.e. separate) way. Rather than that, the signal frequencies which are too close to each other interfere with one another in various ways[7] and are therefore perceived as a single, modulated tone. As these kind of tonal interdependencies are not relevant in speech recognition, it is possible to sum the energy of the signal in several frequ ency bands, giving a very compact representation of the signal. This is usually performed by applying a set of triangular mel-frequency bandpass filters to STPS. The resulting vector of values is called a *filterbank* and is presented in fig. 2-8.

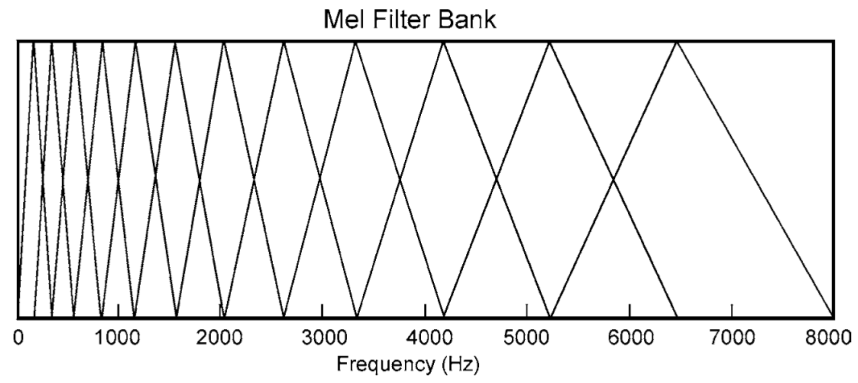

Figure 2-8: Exemplary mel-frequency triangular filterbank.

The other reason for using filterbanks is that the resulting coefficients are in fact the envelope of the spectrum - therefore the pitch of speech is absent. Because of the missing pitch, ASR system will not be sensitive to it - only the timbre of speech will be analysed[5].

SPL has been used to represent the loudness characteristic on a compressed scale, matching closely the actual reaction of human hearing aparatus. A similar philosophy stands behind taking a real logarithm of each filterbank coefficient. Moreover, this operation allows for the separation of the input signal from the channel response in a later step, called Cepstral Mean Normalization (CMN).

2.2.4 Mel-Frequency Cepstral Coefficients

While the log-mel filterbanks contain information about amount of energy in each frequency band in a time frame, another also interesting (from ASR-centric point of view) aspect is the variation between these bands. This quantity can be obtained by performing the Inverse Fourier Transform (IFT) over log-mel filterbanks. The result of this operation is called *cepstrum*[6].

As the full IFT is computationally expensive, we can exploit the fact that the input of transform is a real (as opposed to complex) logarithm. A special case of signal decomposition transform called the Discrete Cosine Transform (DCT) can be

[5]Nevertheless, the pitch of speech is a recommended feature for recognition of tonal languages (e.g. Mandarin).

[6]Spelled /kɛpstrəm/. This has been originally a play on the word “spectrum”.

used for such input:

$$C_m = \sum_{k=1}^{N} \cos(\frac{m(k-0.5)p)}{N})E_k, \quad m = 1, 2, ..., L \tag{2.17}$$

where:

- E_k is the energy of the k_{th} filter band,
- N is the number of filter bands,
- L is the target number of cepstral coefficients.

A feature of the signal computed in such way is called *Mel-Frequency Cepstral Coefficient* (MFCC) and a vector consisting of these features is widely used in speech recognition. An example of cepstrum computation is presented in fig. 2-9.

The cepstral coefficients obtained using DCT have several interesting features. Firstly, as each log-mel triangular filterbank is overlapping with its neighbour, the values of each coefficient tend to be correlated. It means that in future analysis of such a signal the full covariance matrices are to be used during classifier training. Fortunately, a well known property of DCT is that it strongly decorrelates signal[13]. This in turn means, that one can use diagonal covariance matrices, which severely simplify many computations.

Secondly, the input log-mel filterbank vector has $20 - 40$ coefficients (usually 26), but after DCT, it is advised to keep only the lower half (e.g. 12 out of 26). This truncation is performed because higher coefficients represent high frequency oscillations of STPS. It is empirically proved, that various classificators perform better, when these coefficients are discarded[25].

Thirdly, we can perform the aforementioned CMN technique. This is possible because of so called convolution-multiplication property of DFT: Each recorded speech signal $y[n]$ can be represented as a convolution of input signal (glottal excitation) $x[n]$ and channel impulse response $h[n]$:

$$y[n] = x[n] * h[n] \tag{2.18}$$

The h[n] part consists of several components:

- the voice tract filtration,
- the signal radiation
- and the surroundings frequency response.

Each of these components does not contain information about speech, therefore their existence in analysed signal causes deterioration of system's performance and should be, if feasible, removed from the signal. By applying DFT to both sides of the equation, thanks to the convolution-multiplication property, we obtain:

$$Y[f] = X[f] \cdot H[f] \tag{2.19}$$

By taking the logarithm of the spectrum of the signal, we can transform the equation to the following form:

$$\log Y[f] = \log(X[f] \cdot H[f]) = \log(X[f]) + \log(H[f]) \tag{2.20}$$

When averaged over large amount of speech, the $\log(H[f])$ part can be subtracted from the signal, eliminating most of the unwanted impulse response.

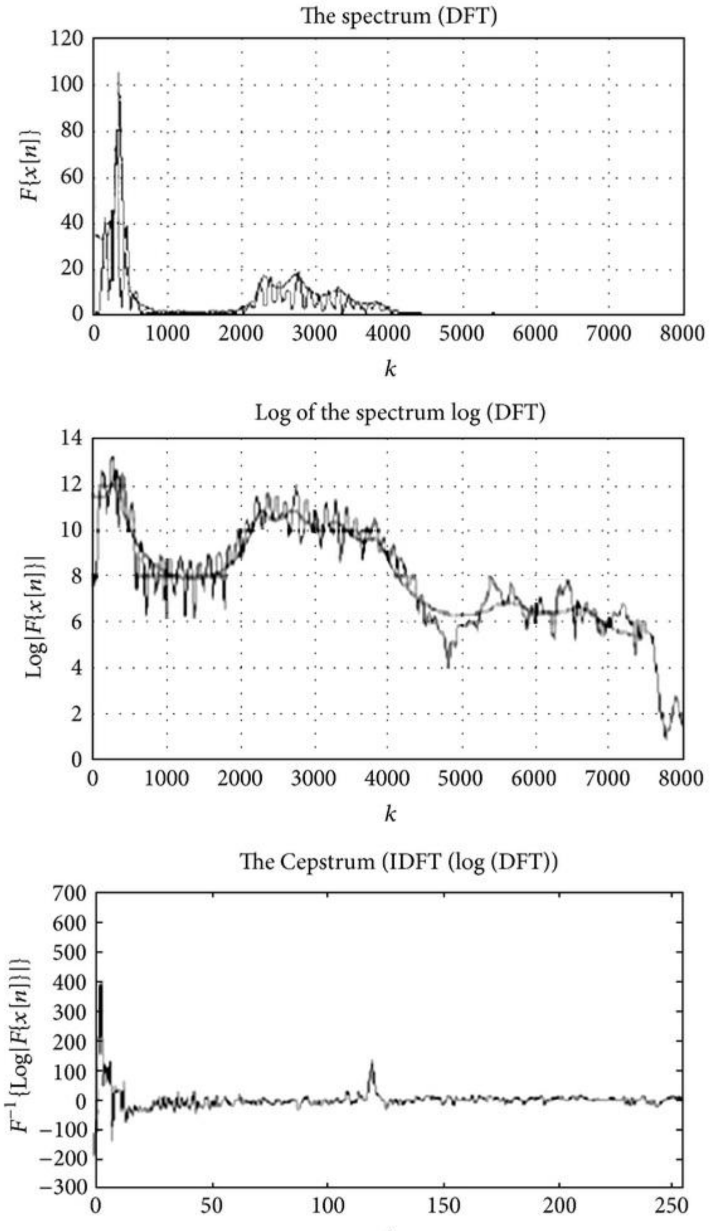

Figure 2-9: The spectrum, log spectrum and cepstrum obtained from the exemplary signal.

Chapter 3

Fundamentals of Acoustic Modeling in Automatic Speech Recognition

3.1 Gaussian Mixture Model

Gaussian (or "normal") distribution is the distribution of choice, if it comes to analysis and modeling of various real-life, physical processes. It is known that physical quantities which are influenced by various independent processes often fit to normal distribution[17]. This is also the case with most acoustic phenomena[1] and, notably, speech processes.

We say that a variable has Gaussian distribution, if its probability density function (pdf) is described by the following equation:

$$p(x|\mu, \sigma^2) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp \frac{-(x-\mu)^2}{2\sigma^2} \tag{3.1}$$

where:

- μ is the mean of the distribution (also, location, expectation, or mode)
- σ^2 is the variance of the distribution (also, dispersion).

[1]Other notable distributions includes γ (used in sound envelope modeling[11]), β (used in ultrasound damage scanning[21]) or exponential (models classical energy dispersion of acoustic wave[23])

When plotted, the *mean* of the Gaussian pdf is equal to the distribution bias, whereas the *variance* is responsible for curve flatness (see fig. 3-1).

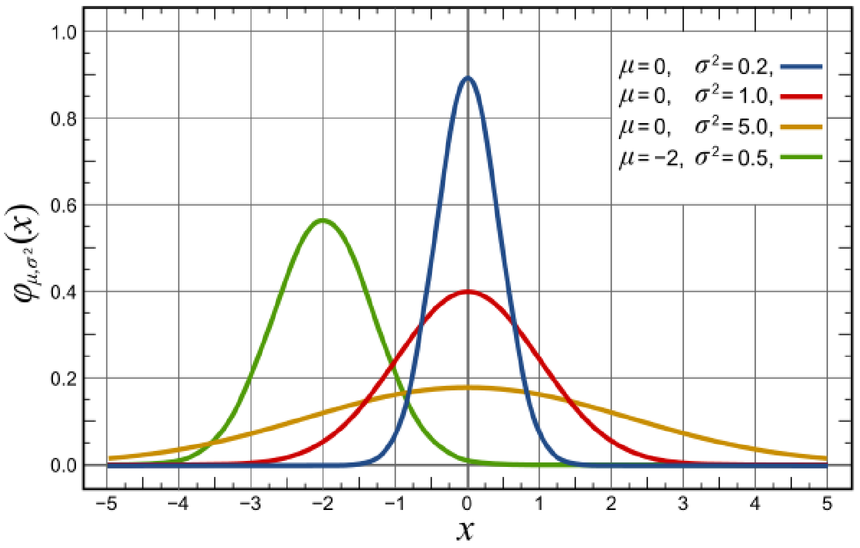

Figure 3-1: Gaussian pdf with various mean and variance values[34].

A generalization of single variable Gaussian distribution into higher dimensions is called a *multivariate Gaussian distribution*. In this case, the mean is represented by a vector $\overline{\mu}$, whereas the variance is replaced by the covariance matrix Σ. The formula for the N-dimensional multivariate Gaussian pdf is following:

$$p(\overline{x}|\overline{\mu},\Sigma) = \frac{1}{(2\pi)^{\frac{N}{2}}\,|\Sigma|^{\frac{1}{2}}} \exp \frac{-(\overline{x}-\overline{\mu})^T\Sigma^{-1}(\overline{x}-\overline{\mu})}{2} \tag{3.2}$$

Each element of the $\overline{\mu}$ vector is simply the mean of respective $\overline{x}$ values and the element of the covariance matrix indexed $[i, j]$ is the covariance of x_i, x_j. It is worth noting, that the diagonal of Σ consists of component variances. The simplest visualization of multivariate Gaussian pdf can be shown for $n = 2$ (see fig. 3-2).

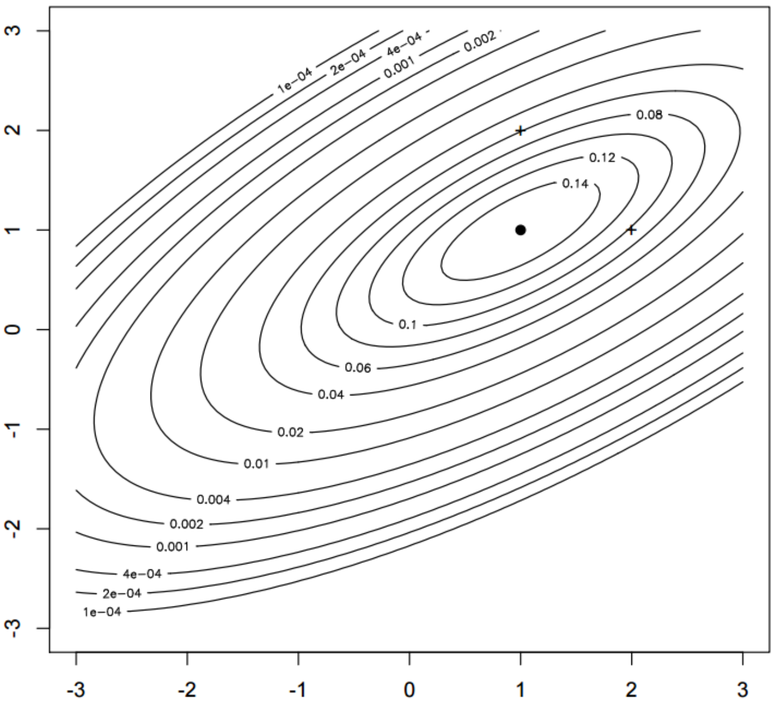

Figure 3-2: Exemplary multivariate Gaussian pdf described by $\overline{\mu} = [1\ 1]^T, \Sigma_1 = [2\ 1], \Sigma_2 = [1\ 1]$. [27]

If we would like to obtain a more generalizable pdf, we can simply create a weighted sum (a linear combination) of more than one pdfs. Such a construct is called a mixture distribution. When all of the components of a given mixture are Gaussian pdfs, a *Gaussian Mixture Model* (GMM) is created. This model is defined by a concise formula:

$$p(x) = p(x|m) \cdot c(m) \tag{3.3}$$

where:

- $p(x|m)$ is a multivariate Gaussian pdf
- $c(m)$ is a mixing parameter (weight).

It is known, that GMMs can approximate any density function, provided that sufficient number of mixture components are used[2]. This is also true for pdfs, whose covariance matrices are diagonal (i.e. only non-zero components of covariance matrix Σ are variances) - so called spherical Gaussian pdfs. This assumption greatly reduces a number of parameters necessary to describe a GMM, as the parameter count of full-rank pdf is equal to the square of the pdf dimension.

Training GMM can be efficiently performed by *expectation-maximization* (EM) algorithm. EM is widely used in computing maximum likelihood parameters of models, which include latent variables. We can define estimating GMM parameters as such a problem, where each observed data point x_i has a hidden variable, associating it with j^{th} mixture component. Therefore, EM algorithm for GMM can be described in the following way:

1. Guess the initial parameters of each mixture (usually using K-means):

$$\theta^g = (c_1^g, c_2^g, ..., c_M^g, \mu_1^g, \mu_2^g, ..., \mu_M^g, \sigma_1^g, \sigma_2^g, ..., \sigma_M^g) \tag{3.4}$$

2. Compute probabilities of each sample x_i belonging to j^{th} mixture (E-step):

$$p(y[i] = j|x[i]; \theta^g) \tag{3.5}$$

3. Update the GMM parameters (M-step):

$$c_j^{new} = \frac{1}{N}\sum_{i=1}^{N} p(y[i] = j|x[i]; \theta^g) \tag{3.6}$$

$$\mu_j^{new} = \frac{\sum_{i=1}^{N} x_i p(y[i] = j|x[i]; \theta^g)}{\sum_{i=1}^{N} p(y[i] = j|x[i]; \theta^g)} \tag{3.7}$$

$$\sigma_j^{new} = \frac{\sum_{i=1}^{N} (x_i - \mu_j)^2 p(y[i] = j|x[i]; \theta^g)}{\sum_{i=1}^{N} p(y[i] = j|x[i]; \theta^g)} \tag{3.8}$$

4. Repeat steps 2-3 until $\Delta < \epsilon$ (condition of convergence).

3.2 Hidden Markov Model

The essential goal of speech is to convey a message through specific movement of articulators, synchronized with acoustic excitement. A sequence of phones is then produced to encode a sequence of words. Thus, the problem of speech recognition might be defined as maximizing the probability of linguistic sequence W, provided the existing acoustic evidence A. This can be expressed in the following form:

$$P(\widehat{W}|A) = \operatorname{argmax} P(W|A) \tag{3.9}$$

After applying the Bayes theorem, we obtain:

$$P(W|A) = \frac{P(A|W)P(W)}{P(A)} P(\widehat{W}|A) = \operatorname{argmax} P(A|W)P(W) \tag{3.10}$$

$P(W)$ quantity is called prior probability of a given word sequence derived from a *Language Model*. $P(A|W)$, in turn, is a conditional probability of a given phoneme described by an entity called *Acoustic Model*.

During the process of phoneme emission, three distinct phases occur:

- transition from the previous phoneme or silence,
- stable state,
- transition to the next phoneme or silence.

Each phase is significantly different; sometimes, the middle phase is dominating; in other phonemes, it is so short, that can be considered as non-existing. An example of phone segmentation is shown in fig. 3-3.

The most common way to model a sequential pattern, or, so called "time series" is through *Hidden Markov Model* (HMM). A Hidden Markov Model is a statistical model, in which:

- future states depend only on the present state (i.e. the model is memoryless),
- has unobserved states,
- each state is described by a pdf.

A typical HMM topology used in speech recognition is shown in fig. 3-4. Each of the states of HMM has two types of parameters: *the emission pdfs* $b_j()$ and *transition*

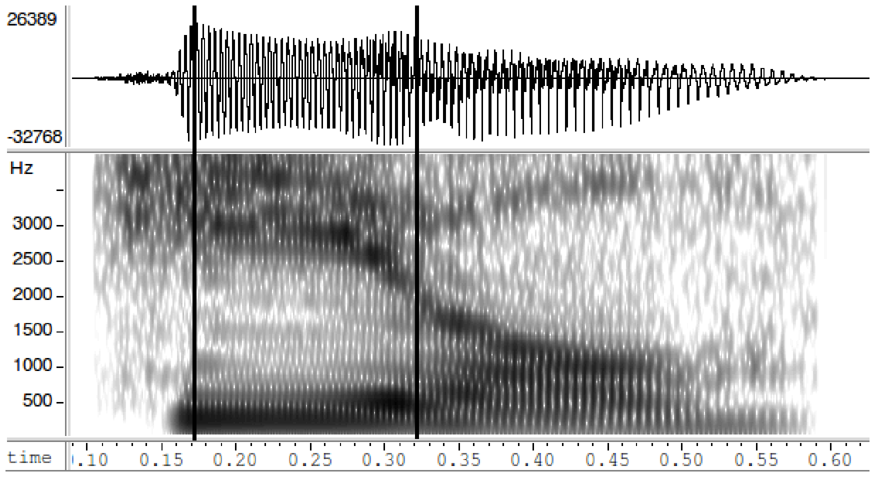

Figure 3-3: An exemplary division of phone /e/ into three parts.

probabilities a_{ij}. Transition probabilities model time durations and are used to transfer from the current state to the next state periodically (usually, every $25ms$). After each transition, a feature vector Y is generated from the emission pdfs. In speech recognition, these pdfs are GMMs, so every $b_j()$ is described by equation (3.3).

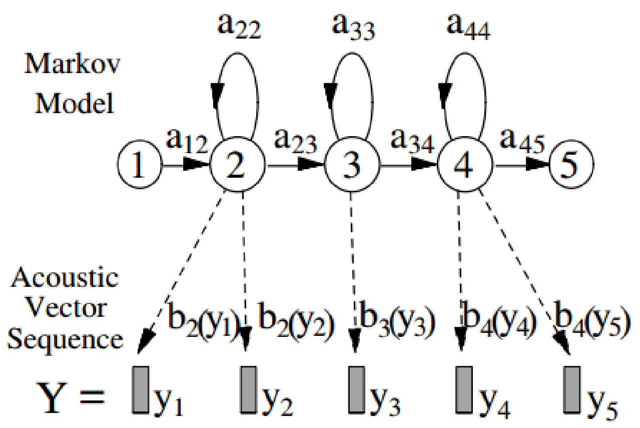

Figure 3-4: 3-state HMM used in phoneme modeling. Note that start- and end-states are not typically included in state count, as they have no self-loop present and do not emit.[5]

If we concatenate all possible phone HMMs into one composite HMM, then we can describe acoustic likelihood as:

$$P(A|W) = \sum_{\theta} P(\theta, A|W) \tag{3.11}$$

where:

$$\theta = \theta_0, \theta 1, ..., \theta_{T+1} \tag{3.12}$$

is a state sequence in the composite HMM and

$$P(\theta, A|W) = a_{\theta_0\theta_1} \prod_{t=1}^{T} b_{\theta_t}(y_t) a_{\theta_t\theta_{t+1}} \tag{3.13}$$

This approach could be summarized as follows: sum over all possible state sequences that could result in observation Y. To compute acoustic likelihoods effectively, apply the Markov assumption and compute it recursively; otherwise, the search space would be too large even in the simplest cases. An algorithm of choice here is the forward-backward algorithm[1], which is a variant of the EM algorithm described in chapter 3.1.

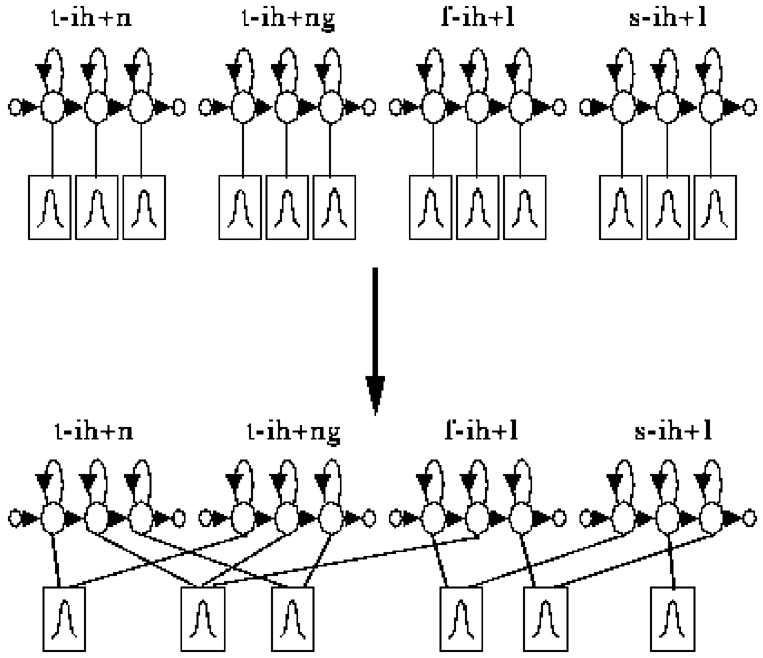

Figure 3-5: (Up) four triphone HMMs, which previously would be a single monophone. (Down) four triphones with state tying.[35]

When modeling phonemes using previously described model (so called *monophone* model), a problem arises with the lost inter-phoneme information. In this approach, no context data is taken into account and it is known, that such systems do not perform well[36]. To improve efficiency, we can use a unique phoneme for each set of neighbouring phonemes, including the silence phoneme. Such a model is called a *triphone model*. It is possible to build a N^3 triphone model for any model with N monophones. To reduce the number of necessary parameters in the triphone model, the following techniques are used:

- similar HMM states in triphone models are statistically clustered and shared across the whole model (this is so called tied model, see fig. 3-5)
- triphones which did not occur in training data are removed

3.3 Recurrent Neural Network

Modeling the $b_j()$ part of eq. (3.13) with a GMM has been an industry standard for a dominant part of modern speech recognition history. Until recently, various experiments with Neural Networks (NNs) have not provided significant improvements in recognition quality. Such state of affairs was mainly caused by the shortage of resources - both in terms of available training data and computational power. This in turn strongly restricted the available NNs architectures, as well as their depth. The advent of NNs was enabled by the elimination of both these obstacles, i.e.:

- creation of large databases of read, broadcast and conversational speech, and
- affordable access to fast and capacious GPUs, along with GPU-accelerated toolkits such as CUDA.

Among many NN architectures explored in context of speech recognition, a specific one named the *Recurrent Neural Network* has attracted a lot of attention. In the basic implementation of NN, we treat sequential inputs and outputs as independent of each other (the same assumption was taken in HMM-GMM tandem). This is clearly not the ideal assumption for speech signal, where the high-level qualities are deeply interdependent. This is highly apparent on the level of whole sentence, where subsequent words are obviously related to the previous ones, but not only; the relation between phones can span over more than the neighbouring triphones.

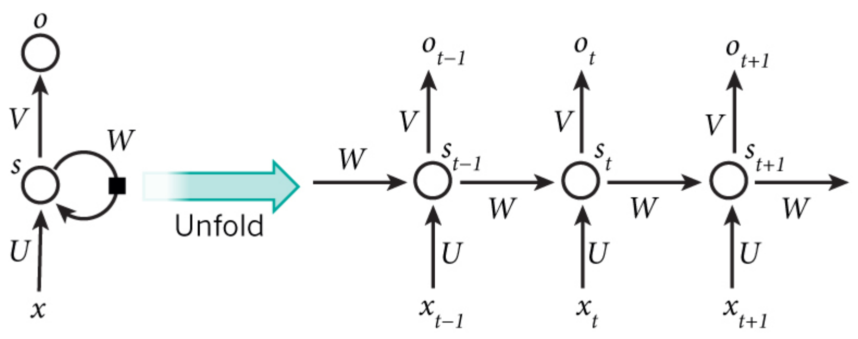

Figure 3-6: A typical RNN, unfolded in time.[16]

To make the NN susceptible to information encoded in long sequences, one can make the output of NN reliant not only on the current input, but also on the previous outputs. In other words, the RNN is a NN model, where some connections between units or layers are forming a direct cycle. Following equations govern the behavior of a single layer RNN:

$$h_t = f(W_{xs}x_t + W_{ss}s_{t-1}) = f(u_t) \tag{3.14}$$

$$y_t = g(W_{sy}s_t) = f(v_t) \tag{3.15}$$

where:

- x_t is the vector of K input values (features),
- s_t is the vector of N hidden state values,
- y_t is the vector of L output values,
- W_{xs} is the $N \times K$ matrix of weights connecting the inputs to the hidden units,
- W_{sy} is the $L \times N$ matrix of weights connecting the hidden units to the output,
- W_{ss} is the $N \times N$ matrix of weights connecting the hidden units from time $t-1$ to time t,
- $f()$ is the hidden layer activation function, usually sigmoid, tanh or rectified linear units (ReLU),
- $g()$ is the output layer activation function, usually linear or softmax,
- u_t is the hidden layer potential,

- v_t, is the output layer potential.

A typical RNN, with additional unfolding in time has been depicted in fig. 3-6.

The hidden state s_t could be perceived as the memory of the NN - it contains the information about the previous input sequences . The length of the history fed into the RNN doesn't have to be limited, but it usually is[10].

The standard technique for learning the weight matrices is called Backpropagation-Through-Time (BPTT) and is an extension of the classic backpropagation algorithm. Newly learned weights can be computed using following equations:

$$W_{sy}^{NEW} = W_{sy} + \gamma \sum_{t=1}^{T} \delta_y^t s_t^T \tag{3.16}$$

$$W_{xs}^{NEW} = W_{xs} + \gamma \sum_{t=1}^{T} \delta_s^t x_t^T \tag{3.17}$$

$$W_{ss}^{NEW} = W_{ss} + \gamma \sum_{t=1}^{T} \delta_s^t s_{t-1}^T \tag{3.18}$$

$$\delta_y^t = (l_t - y_t) \cdot g'(v_t) \tag{3.19}$$

$$\delta_s^t = (W_{ss}^T \delta_s^{t+1} + W_{sy}^T \delta_y^t) \cdot f'(u_t) \tag{3.20}$$

where:

- l_t is the expected RNN output,
- γ is the learning rate constant,
- ^{T} is the transposition operator,
- $\cdot$ is the element-wise multiplication operator.

This algorithm may be summarized in a following way:

- stack the T same hidden layers across the time, $t = 1, 2, \ldots, T$, similarly to multi-layer NN,
- compute the objective function, i.e. cost function defined as the sum-square error between output and held-out target,

- compute the gradient of the cost with the Gradient Descent (GD) algorithm,
- update the weight matrices.

3.4 Long-Short-Term Memory cells

RNN training performed in the way described in chapter 3.3 is susceptible to several factors. Firstly, if the $L_2 - norm$ of the hidden layer weights is greater than some constant (dependent on activation function choice), the gradient computed during GD will explode, i.e. make the training unstable. Conversely, the vanishing gradient problem occurs, when the same $L_2 - norm$ is lesser than the same constant. These conditions makes the training process very unstable and prone to hand-tuning. Secondly, despite having recurrent connections, RNNs are not sufficiently sophisticated to model complex sequence dynamics, especially for long input sequences.

Recently, a more advanced NN architecture with implicit memory structure called *Long-Short-Term Memory* has been used in Speech Recognition. A LSTM-RNN model use several types of gates which alter the flow of data in the network. Due to that property, the network can decide whether the input is relevant and should be saved, if the information previously stored is still needed and when to output the data.

A LSTM cell can be described with the following equations:

$$i_t = \sigma(W_{xi}x_t + W_{hi}h_{t-1} + W_{ci}c_{t-1} + b_i) \tag{3.21}$$

$$f_t = \sigma(W_{xf}x_t + W_{hf}h_{t-1} + W_{cf}c_{t-1} + b_f) \tag{3.22}$$

$$o_t = \sigma(W_{xo}x_t + W_{ho}h_{t-1} + W_{co}c_{t-1} + b_o) \tag{3.23}$$

$$c_t = f_t \cdot c_{t-1} + i_t \cdot \tanh(W_{xc}x_t + W_{hc}h_{t-1} + b_c) \tag{3.24}$$

$$h_t = o_t \cdot \tanh(c_t) \tag{3.25}$$

where:

- i_t is the vector of input gate values,
- f_t is the vector of forget gate values,
- c_t is the vector of cell activation values,
- o_t is the vector of output gate values,
- s_t is the vector of simultaneously hidden layer and output values,
- σ is the logistic sigmoid function,
- W's are the corresponding weight matrices,
- b's are the corresponding bias vectors.

An overview of LSTM-RNN cell has been depicted in fig. 3-7.

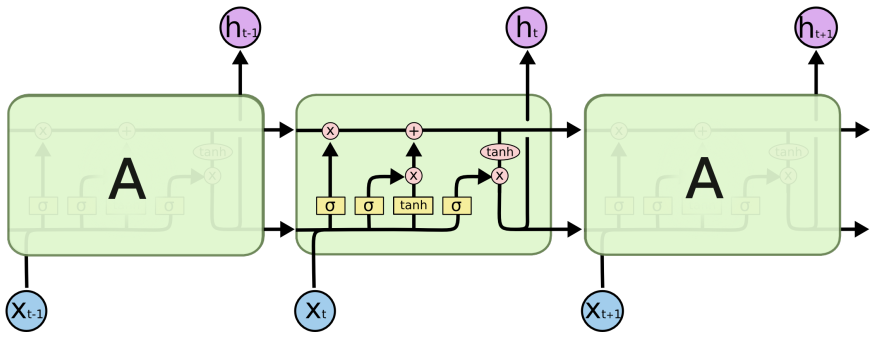

Figure 3-7: An overview of LSTM cell, unrolled in time.[22]

There are several advantages of using LSTM-RNNs over vanilla RNNs. Due to the constraints implied by the ensemble of input/forget/output gates, the vanishing gradient problem does not occur. That fact allows for training the LSTM-RNN in a simpler way, despite having larger number of model parameters. The same BPTT technique is used during training of LSTM-RNN cells.

Another improvement over generic RNNs is that the NN can infer the long-term dependencies between fragments of input sequence. What is more, the relevant chunks of sequence do not have to be neighbouring each other - the recurrent mechanism in LSTM-RNN is much more powerful and can induce context from temporally distant input fragments.

Using LSTM-RNNs in Acoustic modeling have one more, powerful advantage over any other model used before. As the LSTM-RNN is not limited to analysing only consecutive frames of speech, it is much more resistant to both additive noise introduced into speech signal (such as a passing car, a humming engine, etc.) and multiplicative signal perturbation (such as excessive reverberation, channel distortion and many more)[6].

The LSTM-RNNs are also much less sensitive to feature correlation. For that reason, the input feature vector usually consists of pure filterbank features and not MFCCs. That change, due to decreased computational requirements, lead to decreased input latency, especially relevant during live applications of speech recognition.

Chapter 4

Experimental setup

4.1 System overview

The first dilemma, when it comes to developing any software, is the target operating system. In this case, the choice was very simple: Linux. The advantages of using this system are plentiful, mainly:

- When it comes to text processing, Linux has a vast array of built-in tools which are available in any Linux distribution, out of the box. This includes tools such as: *sed*, *awk*, whole *coreutils/gnutils* packages and many more. These facilitate the use of regular expressions and various text manipulation techniques.
- Linux pipes and redirections allow chaining different commands into each other, preventing creation of temporary files. This approach eliminates almost all unnecessary input/output operations and, as a result, vastly speeds up many activities.
- Linux provides tools for easy parallelization of computations. For simple tasks, run on a single machine, tools like *GNU parallel* are sufficient. For tasks, which have to be distributed among a number of different machines, there exist scheduling platforms such as LSF or openlava and they are easily configurable on any Linux distribution.
- A number of speech processing tools already exist in Linux ecosystem. The ones

used during the development of this system are: *Kaldi*, a state-of-the-art Speech Recognition toolkit; *sox*, a sound processing tool; *speex*, a speech codec; *FaNT*, a filtering and noise-adding tool; *SRILM*, a Language modeling toolkit. Most of these tools could be run under Windows, but nevertheless, most of them relies on previously mentioned advantages of Linux to boost their performance.

Setting up a speech recognition system is a complex project, which, daunting at first, as it encompasses many different branches of mathematics, computer science and linguistics, could be divided into various smaller tasks. Modularity of such a system enables easy tweaking and usually allows to separate performance gains introduced by different modules. This approach has been successfully used in the experiments performed for the needs of this thesis. The designed system consists of several well separated units, each performing a distinctive task. These are:

- data preparation stage,
- data enhancement and feature extraction stage,
- lexicon preparation stage,
- HMM-GMM Acoustic Model training,
- LSTM-RNN Acoustic Model training,
- system evaluation stage.

Each of these units will be described in subsequent chapters. All units (except the LSTM-RNN training due to its long training time) have been programmed using declarative approach in common Linux build automation tool, *Make*.

The decision to use Make as a highest-level abstraction layer has been made after a thorough analysis. As a programming language, Make has many flaws: its parser behaves differently from the typical known ones, the error reporting is cryptic and lacking the relevant feedback, the syntax is unintuitive and its features are often obscure. Nevertheless, after overcoming the inherent problems coming with Make the experience is rewarding and in the long way, it simplifies and speeds up the development process. Some of the most notable advantages of such a choice were:

- Make operates on the dependencies described in the Makefile. Therefore, the duty of the programmer lies only in declaring the relations between different modules, their interactions and the expected result. The programmer doesn't have to define the order of the computations - it is automatically derived from the dependency tree, constructed while running Make.
- Thanks to the declarativity of its language, Make automatically parallelizes computations when possible. This can immensely speed up the training process, especially when running on multi-core servers. If programmed correctly, it can utilize the capabilities of HPC (High Performance Computing) clouds.
- Make operates mainly on files. This approach is well suited for building a speech recognition system, where all the relevant data, whether they are intermediate or used in the final product, are stored as a file - e.g. a dictionary, a preprocessed training corpus or an acoustic model.
- Make manages intermediate files in a sane way - if the disk space constraints are enforced, these files are removed when no longer necessary. Conversely, if the storage is not limited, one can leave the intermediate targets for later usage. These might be useful in speeding up the retraining process, especially when only a part of the system is modified, e.g. training parameters.
- "Everything is a file" approach enforces saving various configuration parameters in separate files. This is not a direct advantage, but when used correctly, supports parallel training of multiple models.
- Make checks the timestamps of the dependencies and rebuilds the target only if its timestamp is older from one of its dependencies. That feature prevents unnecessary computations and ensures that all the recent changes have impact on the resulting model.
- The problem with obscure error reporting is partially mitigated when using alternative implementations, such us remake.

The underlying speech recognition system recipe has been designed in Kaldi. Kaldi is an open source toolkit designed strictly for research in the area of speech recognition, written mostly by Dan Povey, Karel Vesely, Arnab Ghoshal and many more[24]. The author of this thesis had committed several bug-fixes to its code during his work in the speech recognition in the last few years. The choice of Kaldi was an apparent one - it is the most advanced and most versatile speech recognition toolkit available. It is well documented, easily extensible, scalable with both CPU and GPU implementations. Its code is written in several languages, mainly:

- C++, used for low level operations. Its executables are mostly *filters*, i.e they expect to be extensively piped (in the data flow sense).
- Perl, used for text manipulation. A great deal of the operations is being performed on the text files, transforming them in various ways (through regular expressions, mappings, conversions, etc.) and Perl is a natural choice for such activities.
- Bash, used for control flow and medium-level abstractions. Bash scripts usually wrap several Perl and C++ commands in one larger script.

Kaldi uses several third-party libraries and tools internally. The most notable are: OpenFST, SRILM and ATLAS. OpenFST is a library used for creating and manipulating weighted finite-state transducers and acceptors. It is extensively used when representing probabilistic models such as n-gram Language Models, lexicons, acoustic contexts and HMMs. Such representation enables using several well-know FST techniques: *determinization* and *minimization* for model optimization, *composition* for joining several models and *shortest path* for choosing the best available result[1].

SRILM (Stanford Research Institute Language Modeling) is a statistical Language Modeling (LM) toolkit. In Speech Recognition context, it is used during preparation of n-gram models, preferably in ARPA format. It provides the tools for training LMs with the most popular smoothing techniques, such as Witten-Bell and Knesser-Nay smoothing; it could also be used for model pruning and normalization, expansion of class models and interpolation of a few models into one[2].

[1]For more information, see [20]

[2]For more information about Language Modeling, see [30] and [31]

Automatically Tuned Linear Algebra Software (ATLAS) is a library for linear algebra. The library can be tuned for specific platforms, utilising the capabilities of different CPUs, such as AVX, certain L1/L2/L3 cache size and processor frequency. The compilation process is difficult and requires some knowledge concerning CPU throttling behavior and Makefile modification.

In subsequent chapters, each stage of Speech Recognition system preparation will be briefly described.

4.1.1 Data preparation

The speech corpora come in many different types, whether regarding technical aspects, such as speech file encoding, transcriptions format and data structure, or quality aspects, such as noisiness, trimming quality and volume normalization. The data preparation step handles all the operations concerning corpora variability and outputs a well-defined input which could be readily used in next stages. An overview of the data preparation step is depicted in fig. 4-1.

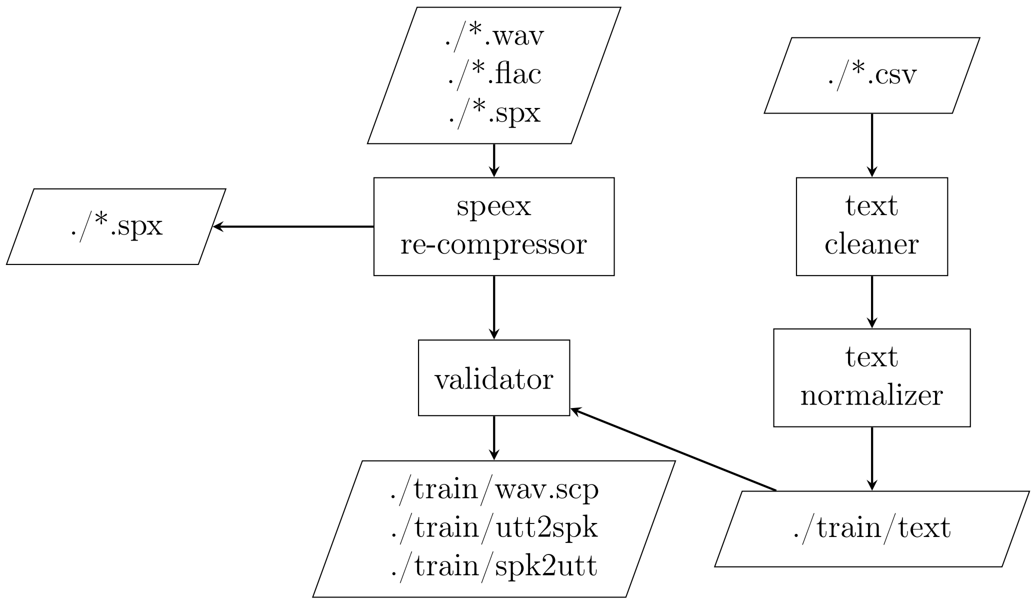

Figure 4-1: Data preparation flowchart.

The left side of the flowchart depicts the preparation of the speech files. Three different audio formats are accepted as the input - flac, wave and speex. To facilitate the usage of several corpora at once, the user can freely mix the formats, as the scripts will use the correct decompressor based on the file header. As the first step, the speech files will be recompressed using speex codec for several reasons:

- to reduce the size of the speech corpus. The input corpora are frequently very large (tens of millions of recordings) and limiting I/O overhead should speed up the training process significantly. This is especially important when using distributed file systems, such as NFS - the throughput of such solution is already significantly lower.
- To maintain sampling rate, bit rate and number of channels corresponding to the anticipated system input. Any discrepancy between training and evaluation corpora in terms of mentioned qualities will make the system unusable without additional adaptations, so it is advised to keep the data homogeneous in terms of signal parameters.
- To introduce distortions distinctive to speex codec. Speex changes the speech file with several techniques, mainly: AGC (Automatic Gain Control) and psychoacoustic enhancements. Both are transforming the speech signal in a non-linear way and it is necessary to keep this transforms the same for both training and evaluation corpora.

The resulting speex files will be saved for later - they will be used in future stages of processing. In parallel, the input *.csv files containing the corresponding transcriptions along with various metadata will be cleaned with an array of Perl/Bash tools. The cleaning includes:

- removing transcriptions with ambiguous fragments,
- removing tags from transcriptions,
- removing unnecessary characters, such as parentheses, quotes, orphaned hyphenation and apostrophes, punctuation marks from transcriptions.

Next, the cleaned text stream will be normalized - the casing will be removed and all numerals will be changed to their literal form to avoid uncertainty when it comes to word pronunciations. The output text will be left for later use.

In the last stage, the outputs from speech and text processing pipes will be fed to the validator, which ensures that they conform to the Kaldi notations. Additionally, all orphaned entries are removed from the training corpus. Finally, the validator outputs several mapping files which are directly used in further training. The speex audio files will be referenced with these mapping files.

4.1.2 Data enhancement and feature extraction

One of the biggest obstacles in the training of the Speech Recognition systems is the data scarcity. Usually, the amount of available data in languages other than English is too small to properly train the HMM-GMM system. This problem is escalated even more when training NN-based models - these require an even bigger training corpora to perform well.

A clever and recently actively researched workaround is to artificially create more training samples. This can be done by various means: synthesizing speech data using Text-to-Speech software, simulating various acoustic conditions with impulse responses, enhancing the signal with additive noise, perturbing the signal in time domain and many else. The last two approaches have been used in this thesis to effectively increase the training corpus size by up to 15 times.

The so called *speed perturbation* technique has been initially proposed in [15]. The experiment relied on changing the speed of speech files by a fixed amount - respectively $\pm10\%$ for 3-fold perturbation and $\pm10\%, \pm5\%$ for 5-fold perturbation. Audio files have been modified with *sox* utility. The reported results indicated 3-6% relative improvement when using 3-fold perturbation and no additional improvement when using 5-fold perturbation, comparing to the baseline. The improvement is attributed to the shift in log Mel spectral envelopes.

The approach to speed perturbation has been slightly modified in this thesis. First of all, the choice of perturbation factor has been randomized within provided range. This should prevent overfitting the Acoustic Models, especially the NN-based ones. Secondly, both 3-fold and 5-fold perturbation have been tested - the reason for that is that our system has been trained on a corpus which was significantly smaller from the ones reported in [15]. Also, the randomization factor could enable further gains from using 5-fold perturbation.

The other data enhancement technique, namely, the addition of various environmental noises to audio samples has been performed using a tool called $FaNT$ (Filtering and Noise Adding Tool). In its vanilla form, FaNT loads both the input wave file and the noise file to the memory, chooses a random noise fragment and applies it to the input with a predefined SNR. The author's Speech Recognition training system used a modified version of FaNT. The main changes are:

- FaNT has been rewritten to be used as a Linux filter. This enables easy integration into Kaldi pipelines and avoids unnecessary disk overhead.
- FaNT no longer loads the whole noise file into memory. Instead, it reads directly the target audio chunk. This is particularly important when the default behavior of applying noise in batch is no longer valid (see the point above) and the new filter behavior caused a massive slowdown.
- FaNT accepts ranges of SNR instead of a single SNR. This allows for the creation of more realistic training corpora.

The noise file used in FaNT consists of several hours of recordings of the various acoustic ambiance. This is a big advantage over the approaches which use artificially generated noises, as it closely resembles the acoustic conditions met in real-life situations.

These two speech enhancement solutions can be combined together with the preprocessing pipes generated during data preparation stage. The final pipeline is then fed to the feature extraction stage. Two kinds of features will be prepared for further use - 13 MFCCs for HMM-GMM training and 40 filterbanks for LSTM-RNN training. To speed up the process of feature extraction, these operations are performed in parallel - the openlava scheduler is used to provide the parallelization. On top of the resulting features, CMVN statistics are additionally computed[3].

[3]This is a very fast operation. It does not have to be parallelized.

An overview of the data enhancement + feature extraction step is depicted in fig. 4-2. Note, that the noisy data are present only in filterbank form and are absent from MFCC processing; this is intentional. The noisy data have been removed from HMM-GMM training, as this model is being used only for LSTM+RNN model bootstrapping and would unnecessarily slow down the training process. Additionally, the noisy features have the same length as the clean features - this means, that the alignments[4] produced during HMM-GMM training could be used for both sets of features. This could not be done with speed perturbed features, as their length is obviously being changed.

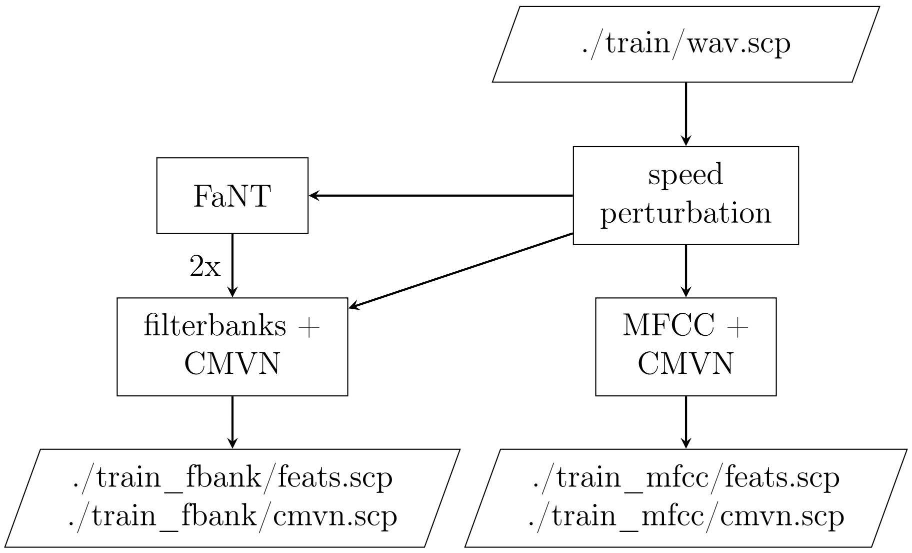

Figure 4-2: Data enhancement and feature extraction flowchart.

4.1.3 Lexicon preparation stage

The current generation of Speech Recognition systems isn't capable of mapping speech features directly into corresponding words. An intermediate representation is necessary and is indeed provided by pronunciation dictionaries. A pronunciation dictionary (sometimes called *lexicon*) is a file containing mappings from each word to the underlying sequence of phones. Some words can have multiple pronunciations - a few possible reasons are:

[4]An alignment is simply the assignment of consecutive HMM states derived from the transcription to the most probable frames in the audio file.

- elision - deletion of a phone when spoken quickly, careless or in some specific context,
- language dialect - dialects usually might modify pronunciation in some way (e.g. "tomato" in American and British English),
- foreign origin - elder people tend to pronounce foreign words in a specific manner, e.g. "tortilla" with the double-L pronounced as /l/ instead of /j/.

A lexicon can be handcrafted, usually by a skilled linguist, or created with the help of grapheme-to-phone (G2P) tools. In this Speech Recognition, the latter approach has been used. First, the input text corpus has been tokenized and all the unique tokens have been selected. Such list has been saved with additional word to integer mapping for later use. Next, the word list is filtered with a Perl module, *Unidecode*, translating the characters which are not specific to the given language to the nearest ASCII representation. The purpose of this step is to increase G2P accuracy when transcribing foreign words. Finally, the resulting list is piped into G2P tool and joined with the original word list.

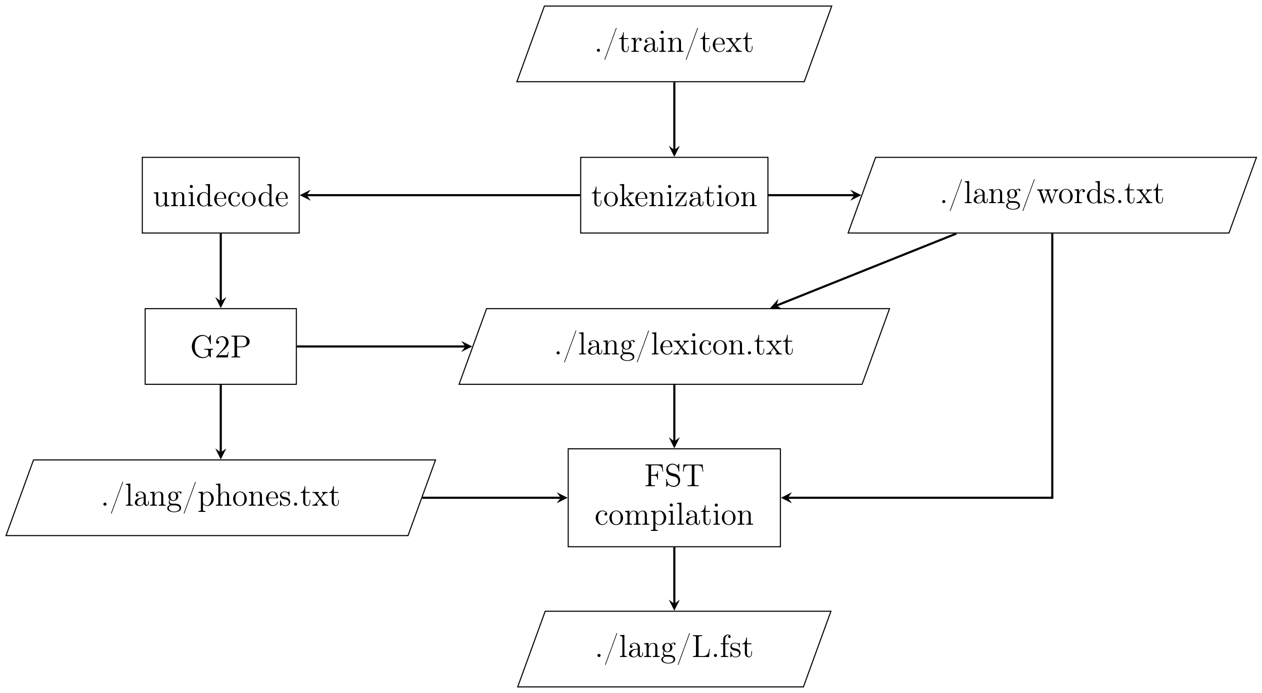

Figure 4-3: Lexicon preparation flowchart.

When possible, Kaldi operates on FST representations of various data structures. This is the case with the lexicon - it is compiled using OpenFST tools to a transducer form. Three mappings are necessary to enable the compilation of the lexicon - phone-to-integer, word-to-integer and word-to phones mappings. Additionally, each phone is decorated with position dependent tag: _B, _I, _E or _S, which are, respectively, word beginning, middle, end or a singleton. An overview of this stage is presented in fig. 4-3

4.1.4 HMM-GMM Acoustic Model training

The HMM-GMM training is run according to the usual Kaldi training recipe. Initially, a monophone model is bootstrapped using the provided training corpus, features and the lexicon. Then, as we don't have any knowledge about the actual positions of phones in the training corpus, the naive alignment is prepared - the phones are distributed evenly across each utterance. Next, the initial set of GMMs is estimated using the EM algorithm described in chapter 3.1. After GMM estimation, the training data is realigned and the procedure is repeated, until the target number of GMMs is trained.

When the monophone system has been trained, the data are once more realigned and the resulting alignments are used in the triphone model training. Additionally, the input features are extended with delta and delta-delta features - first and second derivatives of feature vector. The advantage of using delta features is that the system can learn not only the static patterns, but also the dynamics of the feature vector. The training procedure is fairly similar to the monophone training.

The whole HMM-GMM training is fairly well parallelized with the openlava scheduler.

4.1.5 LSTM-RNN Acoustic Model training

The LSTM-RNN AM used in this thesis consists of several layers. The input layer consists of 123 nodes - 40 filterbank features + energy + delta coefficients + delta-

delta coefficients. Next, three LSTM-RNN layers, each consisting of 500 nodes, are stacked on each other. The final, output layer is a softmax layer, directly mapping into each HMM activation.

The LSTM-RNN AM is trained on top of the triphone alignments which are used as a NN target sequence. In this thesis, a technique called *transfer learning* has been used to speed up the lengthy process of training model from the scratch. This technique allows for the use of the models with incompatible number of outputs caused by the different number of HMMs. It has been also proved, the NN trained using transfer learning technique generalize much better for unseen input[33]. The process can be summed up as follows:

1. Prepare a well trained LSTM-RNN from another Speech Recognition system.
2. Delete the softmax layer of the original NN and insert a new softmax layer with the appropriate number of outputs derived from HMM-GMM model.
3. Train the network with the algorithm of choice, BUT update only the last layer of the NN.

After transfer learning, the whole model is trained for several epochs. The phone classification error rate is closely monitored to prevent overtraining. If the classification error stops decreasing, the learning rate is decreased and the training is resumed from the the previous epoch. The whole process has been parallelized and is being run on several GPUs.

4.1.6 System evaluation

The models from HMM-GMM and LSTM-RNN stages have to be compared in some way. A common good practice is to evaluate the Speech Recognition system on a corpus which wasn't a part of training data. The system is tested with a set of speech recordings and the resulting hypotheses are compared to the original transcriptions. The metric usually used during the evaluations of ASR systems is *Word Error Rate*

(WER) - the distance in words between the ASR hypothesis and the reference transcription, which is defined as:

$$\text{WER} = \frac{S + D + I}{N} \tag{4.1}$$

where:

- S is the number of word substitutions,
- D is the number of word deletions,
- I is the number of word insertions,
- N is the total count of words in the reference.

To evaluate an Acoustic Model, two other components have to be provided. These are - the Language Model[5] (LM) and the evaluation corpus (described in the next chapter). The LM used in this thesis is a 4-gram model with Kneser–Ney smoothing, prepared with SRILM toolkit using the whole training corpus, with the exception of the evaluation corpora (see chapter 4.2). The model order has been set to mimic the similar evaluations performed in the reported articles. After generating an n-gram LM in ARPA format, a $G.fst$ acceptor has to be compiled. This acceptor is used in composition with three other transducers:

- $L.fst$, mentioned in chapter 4.1.3. Its input symbols are phones, and output symbols are words. The resulting $LG.fst$ transducer is determinized and minimized - this speeds up the evaluation process significantly[19].
- $C.fst$, which represents the context dependency. This transducer maps from the triphone to the underlying phone. The output $CLG.fst$ transducer is also determinized and minimized.
- $H.fst$, which is the transducer representation of HMM. The inputs are the transition IDs (which unambiguously identifies a triple of pdf, HMM state and phone) and the outputs are the context-dependent phones. The resulting transducer, $HCLG.fst$ is again determinized and minimized.

[5]For more background on Language Modeling, see [3]

4.2 Data and configuration overview

The system presented in this thesis has been trained using a proprietary database containing Polish speech. The training corpus consists of 200k utterances spoken by 240 different speakers, giving a total 200 hours of speech. The corpus has been artificially extended using speed perturbation technique, providing additional 800 hours of audio. A separate set of two noisy training corpora has been prepared on top of the whole enhanced corpus.

Two subsets have been excluded from the training corpus for evaluation purposes. These were:

- 1500 utterances containing fairly clean utterances,
- 1500 utterances containing utterances recorded in a noisy environment.

The criterion of speech noisiness was the Signal-to-Noise (SNR) level, defined as:

$$\mathrm{SNR}_{\mathrm{dB}} = 10\log(\frac{P_{signal}}{P_{noise}}) \tag{4.2}$$

The clean evaluation set consisted of audio with SNR $\approx 20dB$, whereas, for the noisy set, SNR $\approx 0dB$. These utterances haven't been used during the training phase, as it would make the results unreliable.

The monophone training has been conducted with the target of 1k gaussians. The small gaussian count is preferred, as it speeds up the training process and the monophone model's performance doesn't really matter - it is used only as an intermediate model. The system is trained for 40 epochs, with additional realignment steps:

- every epoch, for epochs 1 to 9;
- every second epoch, for epochs 10 to 18;
- every third epoch, from epoch 20.

The triphone training has a target of 100k gaussians and 8000 HMM states. These numbers have been found optimal (i.e. produced the system with the lowest WER) for speed-perturbed models with 5-fold perturbation. The training has been run for 30 epochs, realignments step is present in every 10^{th} epoch.

The LSTM training is carried out with the initial learning rate of 0.0005. The learning rate is halved when the WER stops decreasing. The model is trained on all the clean+speed perturbed data and, additionally, the randomly chosen hlaf of each noisy corpus - 1600 hours of speech in total.

4.3 Platform overview

The training has been performed on a computing cluster consisting of 96 single CPU nodes and 10 single GPU nodes. Each node had a mounted shared NFS storage, containing all the necessary tools and data. The access to the cluster was provided through the openlava scheduler. Each training job had to be queued through the scheduler commands. In an event of node failure, the job was rescheduled on another free node, preventing the failure of the training process.

The overview of the training times for 5-fold speed-perturbed model training have been compiled in table 4.1. If the stage is not present in the table, it was completed almost instantly.

Operation	**Data preprocessing**	**Feature extraction**	**Monophone training**	**Triphone training**	**LSTM-RNN training**
Epoch duration	1h	30min	15min	24min	12h
Total duration	1h	1h30min	10h	12h	7.5days

Table 4.1: Summary of the processing time of each stage

Chapter 5

System evaluation

5.1 Impact of speed perturbation on HMM-GMM model performance

The first major experiment described in this thesis was the study of the impact of training data speed perturbation on HMM-GMM model performance. Four models have been trained in the course of the study:

- the baseline model with no speed perturbation,
- a model with 3-fold perturbation,
- a model with 5-fold perturbation,
- a model with 5-fold perturbation with additional speed randomization.

The results of each training session, compared to the results obtained in [15] are available in table 5.1. These results are not directly comparable - Switchboard and CallHome English models are NN-based, therefore the gain from speed-perturbation might not be so large, as in this thesis. However, an improvement has been reported in each case. A few conclusions can be made from the results:

- 3-fold speed perturbation significantly increases model performance on clean data and only slightly on noisy data.

- The performance gain on clean data greatly diminishes with increase in folding, although it is still noticeable. On the other hand, the increase in performance on noisy data was doubled.
- Introduction of randomized speed perturbation increased the model performance significantly. It is worth noticing, that this change does not extend the training time; it is essentially a costless performance boost.

	Baseline	3-fold		5-fold		5-fold random	
	WER	WER	Rel. impr.	WER	Rel. impr.	WER	Rel. impr.
Clean	5.95	5.2	12.61	5.06	14.96	4.99	16.13
Noisy	56.44	54.86	2.80	53.43	5.33	51.72	8.36
SWB[1]	13.7	12.9	5.84	-	-	-	-
CHE[2]	27.7	25.7	7.22	-	-	-	-

Table 5.1: Word Error Rate, along with the relative improvement obtained using models with different kind of training data speed-perturbation.

The training time increases linearly with the amount of the training data. This means that, when the baseline model training was ready in around 5 hours, the 5-fold speed-perturbed model training finished after a whole day. While this is clearly not a problem with HMM-GMM training, the NN training tend to converge after a significantly longer period of time. As for that, it is advised to experiment on non-perturbed data and use the speed-perturbation when training the final model only.

5.2 Comparison of HMM-GMM and LSTM-RNN models

In the second experiment, the LSTM-RNN model has been trained on top of the HMM-GMM 5-fold speed-perturbed model. Two additional noisy data sets with

[1]Results on Switchboard corpus, reported in [15]
[2]Results on CallHome English corpus, reported in [15]

SNR = 10dB have been used. The training converged after 15 epochs, with a single learning rate reduction performed after epoch 10. The comparison of the performance of HMM-GMM model with no speed perturbation and the described LSTM-RNN, along with exemplary model performance reported in Kaldi toolkit is presented in table 5.2. While the LSTM-RNN model clearly outperforms the HMM-GMM model on clean data, the real strength of NN model is mostly visible on noisy evaluation set. While the HMM-GMM model struggled to recognize every second word, the neural model performance was only two times worse than on the clean data. This is mostly attributed to the usage of additional noisy corpora during model training.

	Baseline	LSTM-RNN	
	WER	WER	Rel. impr.
Clean	5.95	2.4	59.66
Noisy	56.44	5.1	90.96
SWB[3]	41	18.1	55.85

Table 5.2: Word Error Rate, along with the relative improvement obtained with HMM-GMM and LSTM models.

5.3 Impact of various SNR of training data on LSTM-RNN model performance

The final experiment investigated the impact of SNR of the noisy data on LSTM-RNN model performance. Three different models have been trained in parallel to the baseline model, all on top of the same 5-fold, speed perturbed triphone HMM-GMM model. The training scheme was identical in all cases: 10 epochs of initial LSTM-RNN training, followed by another 5 epochs with reduced learning rate. The reason behind the choice of SNR was following:

- SNR range of [-5..25] was chosen as an initial choice, having the same SNR mean as the baseline model.

[3]Results reported on Switchboard corpus have been included for comparison. They are included in Kaldi distribution[12]. The LSTM model has been trained on 3-fold speed perturbed data.

- SNR range of [0..30] was the next choice, as the previous settings caused a significant drop of model performance on clean data.
- SNR range of [5..35] was the last choice - a possibility of even higher results on clean data, without sacrificing the performance on the noisy data was to be checked.

Table 5.3 compares the performance of each LSTM-RNN model across the various training data SNR ranges. It can be seen that the variants with wider variety of input SNR are clearly outperforming the baseline model with fixed SNR. The further choice of the data augmentation type would be heavily influenced with the target acoustic environment, as every model has its apparent strengths.

	Baseline (SNR=10dB)	SNR=[-5..25]dB		SNR=[0..30]dB		SNR=[5..35]dB	
	WER	WER	Rel. impr.	WER	Rel. impr.	WER	Rel. impr.
Clean	2.4	2.6	-8.33	2.4	0	2.3	4.17
Noisy	5.1	4.7	7.84	4.8	5.88	5.1	0

Table 5.3: Word Error Rate, along with the relative improvement obtained with various LSTM models.

Chapter 6

Conclusions

In this thesis, several different approaches to training a Polish Acoustic Model have been reviewed. The focus of this research was laid on various data augmentation techniques, in particular in relation to the recently popular LSTM-RNN architecture. All of the suggested improvements over the initial HMM-GMM training scheme were used jointly and resulted in = a large relative improvement of WER - 61.3% on clean speech and 91% on noisy speech. The final WER metrics suggest that the training corpus size was sufficiently large.

The performance gains coming from the procedures reported previously in referenced publications have been closely reproduced. Two extensions to the aforementioned enhancement techniques have been tested; both did not increase the computational complexity of the training process and both decreased WER significantly. Thus, it may be concluded that it is advised to always use them in conjunction with speed-perturbation and adding noise to the training corpora.

A few aspects of the performed experiments could be explored further. These are:

- The impact of the speed-perturbation folding on LSTM system hasn't been measured. An experiment similar to the one described in chapter 5.1 could be carried out directly on LSTM models.
- 3-fold random speed perturbation hasn't been tested. This might yield similar gains to the 5-fold random speed perturbation without additional training

overhead.

- A more comprehensive experiment, similar to the one described in chapter 5.3, with different SNR ranges could be carried out. The SNR values tested in this thesis might not be the optimal ones, whether focusing on the clean or noisy data.

It is strongly advised to perform LSTM experiments on a big GPU cluster. The computations can take a very long time and it might be preferred to use a smaller subset of the training data when exploring the different approaches to Acoustic Model training.

Bibliography

[1] Leonard E Baum, Ted Petrie, George Soules, and Norman Weiss. A maximization technique occurring in the statistical analysis of probabilistic functions of markov chains. *The annals of mathematical statistics*, 41(1):164–171, 1970.

[2] Jeff A Bilmes et al. A gentle tutorial of the em algorithm and its application to parameter estimation for gaussian mixture and hidden markov models. *International Computer Science Institute*, 4(510):126, 1998.

[3] Stanley F. Chen and Joshua Goodman. An emprirical study of smoothing techniques for language modeling. Technical Report TR-10-98, Computer Science Group, Harvard University, August 1998.

[4] Li Deng, Geoffrey Hinton, and Brian Kingsbury. New types of deep neural network learning for speech recognition and related applications: An overview. In *2013 IEEE International Conference on Acoustics, Speech and Signal Processing*, pages 8599–8603. IEEE, 2013.

[5] Mark Gales and Steve Young. The application of hidden markov models in speech recognition. *Foundations and trends in signal processing*, 1(3):195–304, 2008.

[6] Jürgen T Geiger, Zixing Zhang, Felix Weninger, Björn Schuller, and Gerhard Rigoll. Robust speech recognition using long short-term memory recurrent neural networks for hybrid acoustic modelling. In *INTERSPEECH*, pages 631–635, 2014.

[7] Stanley A Gelfand. *Hearing: An introduction to psychological and physiological acoustics.* CRC Press, 2016.

[8] Marvin E Goldstein. Aeroacoustics. *New York, McGraw-Hill International Book Co., 1976. 305 p.*, 1, 1976.

[9] Alex Graves, Abdel-rahman Mohamed, and Geoffrey Hinton. Speech recognition with deep recurrent neural networks. In *2013 IEEE international conference on acoustics, speech and signal processing*, pages 6645–6649. IEEE, 2013.

[10] Sepp Hochreiter. The vanishing gradient problem during learning recurrent neural nets and problem solutions. *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, 6(02):107–116, 1998.

[11] Kyle W Hollman, Mark R Holland, James G Miller, Peter B Nagy, and James H Rose. Effective ultrasonic transmission coefficient for randomly rough surfaces. *The Journal of the Acoustical Society of America*, 100(2):832–839, 1996.

[12] `https://github.com/kaldi-asr/kaldi/blame/master/egs/swbd/s5c/RESULTS`. Accessed on 2016-10-13.

[13] Syed Ali Khayam. The discrete cosine transform (dct): theory and application. *Michigan State University*, 114, 2003.

[14] Robin Kinkead. Typing speed, keying rates, and optimal keyboard layouts. In *Proceedings of the Human Factors and Ergonomics Society Annual Meeting*, volume 19, pages 159–161. SAGE Publications, 1975.

[15] Tom Ko, Vijayaditya Peddinti, Daniel Povey, and Sanjeev Khudanpur. Audio augmentation for speech recognition. In *Proceedings of INTERSPEECH*, 2015.

[16] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *Nature*, 521(7553):436–444, 2015.

[17] Aidan Lyon. Why are normal distributions normal? *The British Journal for the Philosophy of Science*, 65(3):621–649, 2014.

[18] Rufin Makarewicz and A Ratajczak. *Dzwieki i fale*. Wydaw. UAM Poznan, 2004.

[19] Mehryar Mohri, Fernando Pereira, and Michael Riley. Weighted finite-state transducers in speech recognition. *Computer Speech & Language*, 16(1):69–88, 2002.

[20] Mehryar Mohri, Fernando Pereira, and Michael Riley. Speech recognition with weighted finite-state transducers. In *Springer Handbook of Speech Processing*, pages 559–584. Springer, 2008.

[21] DCD Oguamanam, HR Martin, and JP Huissoon. On the application of the beta distribution to gear damage analysis. *Applied Acoustics*, 45(3):247–261, 1995.

[22] Christopher Olah. `http://colah.github.io/posts/2015-08-Understanding-LSTMs/img/LSTM3-chain.png`. Accessed on 2016-10-13.

[23] Nicholas G Pace and Finn Jensen. *Impact of littoral environmental variability on acoustic predictions and sonar performance*, volume 1. Springer Science & Business Media, 2002.

[24] Daniel Povey, Arnab Ghoshal, Gilles Boulianne, Lukas Burget, Ondrej Glembek, Nagendra Goel, Mirko Hannemann, Petr Motlicek, Yanmin Qian, Petr Schwarz, et al. The kaldi speech recognition toolkit. In *IEEE 2011 workshop on automatic speech recognition and understanding*, number EPFL-CONF-192584. IEEE Signal Processing Society, 2011.

[25] Halfdan Rump, Shigeki Miyabe, Emiru Tsunoo, Nobutaka Ono, and Shigeki Sagayama. Autoregressive mfcc models for genre classification improved by harmonic-percussion separation. In *ISMIR*, pages 87–92. Citeseer, 2010.

[26] Aleksander Piotr Sek. Akustyka Mowy, Wykład 3. [PDF document].

[27] Cosma Shalizi. `http://www.stat.cmu.edu/~cshalizi/uADA/12/lectures/ch14.pdf`. Accessed on 2016-10-13.

[28] Steven W Smith et al. The scientist and engineer's guide to digital signal processing. page 174, 1997.

[29] Stanley Smith Stevens, John Volkmann, and Edwin B Newman. A scale for the measurement of the psychological magnitude pitch. *The Journal of the Acoustical Society of America*, 8(3):185–190, 1937.

[30] Andreas Stolcke et al. Srilm-an extensible language modeling toolkit. In *Interspeech*, volume 2002, page 2002, 2002.

[31] Andreas Stolcke, Jing Zheng, Wen Wang, and Victor Abrash. Srilm at sixteen: Update and outlook. In *Proceedings of IEEE Automatic Speech Recognition and Understanding Workshop*, volume 5, 2011.

[32] Steve Tauroza and Desmond Allison. Speech rates in british english. *Applied linguistics*, 11(1):90–105, 1990.

[33] Dong Wang and Thomas Fang Zheng. Transfer learning for speech and language processing. In *2015 Asia-Pacific Signal and Information Processing Association Annual Summit and Conference (APSIPA)*, pages 1225–1237. IEEE, 2015.

[34] `https://upload.wikimedia.org/wikipedia/commons/7/74/Normal_Distribution_PDF.svg`. Accessed on 2016-10-13.

[35] Steve Young, Gunnar Evermann, Mark Gales, Thomas Hain, Dan Kershaw, Xunying Liu, Gareth Moore, Julian Odell, Dave Ollason, Dan Povey, et al. The htk book. *Cambridge university engineering department*, 3:175, 2002.

[36] Steve J Young and Philip C Woodland. State clustering in hidden markov model-based continuous speech recognition. *Computer Speech & Language*, 8(4):369–383, 1994.