DEGREE PROJECT IN ELECTRICAL ENGINEERING, SECOND CYCLE, 30 CREDITS

*STOCKHOLM, SWEDEN 2016*

# Investigate more robust features for Speech Recognition using Deep Learning

**TIPHANIE DENIAUX**

**KTH ROYAL INSTITUTE OF TECHNOLOGY**
**SCHOOL OF ELECTRICAL ENGINEERING**

# Abstract

The new electronic devices and their constant progress brought up the challenge of improving the speech recognitions systems. Indeed, people tend to use more and more hands-free devices that are inclined to be used in noisy environments. The evolution of Machine Learning techniques has been very efficient for the last decade and speech recognition system using those techniques appeared. The main challenge of Automatic Speech Recognition systems nowadays is the improvement of the robustness to noise and reverberations. Deep Learning methods were used to either improve the speech representations or defining better distributions probabilities. The problem we face is the drop in the performance of ASR systems when inputs are noisy. The general approach is to define novel speech features that are more robust using Deep Neural Networks. To do so we got through different implementations as the incorporation of autooencoders in the MFCC block diagram or the deep denoising autoencoders with different pre-training methods. The final solution is a system that build more robust features from noisy MFCC. Our input is the demonstration that a denoising system using $q$ quantized DDAEs defined by the clustering of the training data using K-means is more efficient than one denoising system applied to the whole data. The performance gained using such a system is of 2 to 3% in terms of phone error rate and might be improved using more training data and better tuned NN parameters.

# Acknowledgment

First I would like to thank my superviser Saikat Chatterjee for the opportunity he gave me to work on Deep Learning in speech recognition in the Communication Theory department at the School of Electrical Engineering, KTH. His help along this thesis has been a real support, and I really thank him for taking the time to evaluate with me our options and find solutions. I also thank Dr. Md Sahidullah for the help he gave me to start my work, his advices were valued for the rest of my thesis. I thank my examiner Mikael Skoglund and my opponent Fanny for reading my thesis and calling it into question. I thank the Master students that allowed me to attend their presentations because it provided me with experience to prepare my own one. I thank my family members for their support even if they sometimes could not understand what I was specifically working on. I thank Emeric for encouraging me and pushing me into doing my best at any cost and finally I thank all my Stockholm friends but also my french fellows for their friendship and support.

# Contents

# Abbreviations

GM - Gaussian Model
GMM - Gaussian Mixture Model
NN - Neural Network
DNN - Deep Neural Network
DBN - Deep Belief Network
AE - AutoEncoder
SAE - Stacked AutoEncoders
DDAE - Deep Denoising Autoencoder
DBN - Deep Belief Network
RBM - Restricted Boltzmann Machine
MFCC - Mel-Frequency Cepstral Coefficients
SGM - Stochastic Gradient Method
CD - Contrastive Divergence
LM - Language Model
EM - Expectation Maximization
WER - Word Error Rate
PER - Phone Error Rate
FFT - Fast Fourier Transform
DFT - Discrete Fourier Transform
FBE - FilterBanks Energies
SNR - Signal Noise Ratio
VQ - Vector Quantization

# 1 Introduction

## 1.1 Motivation

The growing use of hands-free devices and voice-controlled systems has involved the development of high-performance speech recognition systems. Today's major challenge of Automatic Speech Recognition (ASR) systems is the presence of environmental noise and reverberation that causes a drop in the performance. Machine Learning has become a hot topic since early 2000's and has been used to model the output distribution probabilities of the Hidden Markov models used in Speech Recognition. It is quite recently that the use of Deep learning has reached the same performance as the standard Gaussian Mixture Models. The novel pre-training methods are the new approaches that made this late achievement happened. But other applications of deep learning in speech recognition are also the discovery of bottleneck features [8] or the denoising of features [5]. So the goal of this thesis is to investigate how deep learning can be use in a side-way approach to create more robust speech features.

## 1.2 Structure of the report

This report has been organized as follows. Chapter 2 sums up the background theory of speech recognition and deep learning. Chapter 3 lists and explains the tools used along for this Thesis. In Chapter IV the problem is formulated with its assumptions and the path taken along the thesis is expressed. The implementations done to get to the final solution and the details of the experiments are highlighted and explained in Chapter 5. Finally in Chapter 6 the results are presented and discussed. This last Chapter is followed by the conclusion. The parts of the code used to build the final solution are attached in the appendix.

# 2 Background

In this chapter we provide a brief discussion of essential background in Speech recognition and Deep Learning. Algorithmic details and parameterization are discussed further in the next chapters.

## 2.1 Introduction to Speech Recognition

Basically, an Automatic Speech Recognition System performs a task of recognition from a provided speech signal. In the case of this thesis, the output of the ASR is a text version of the speech and we operate at the phone level. Indeed, a phone is a distinct speech sound commonly associated with a vowel or a consonant. The English language has 42 phonemes that are units of sound and a phone is an acoustic representation associated with a phoneme.

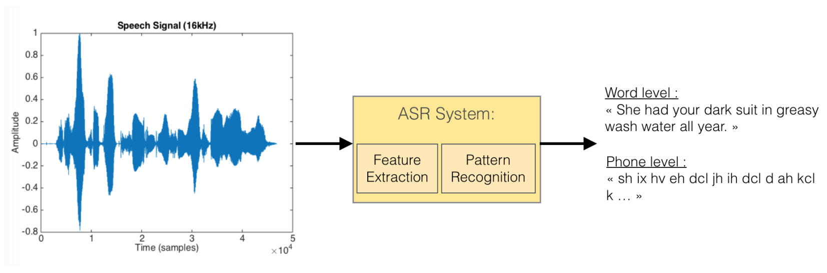


Figure 2.1: ASR System.

### 2.1.1 Feature extraction block

A feature is a mathematical representation of a speech signal. At this stage, the speech waveform is transformed into a parametric representation for an easier analysis and processing in the next pattern recognition's stage. Indeed, raw waveform speech signals present large variations due to speaker variability or environment. Therefore, another domain than the time-domain needs to be used to represent speech and the first to come to mind is the Fourier domain because the frequency domain is relevant for speech. But while human hearing is a compromise between time and frequency resolution, a Fourier transform applied to a whole speech signal discards all timing information in the process. That is why we consider many short segments called frames of length in between 5 and 30 milliseconds. State-of-the-art extraction technique is the Mel-Frequency

Cepstral Coefficients. They have been chosen for their computational simplicity, their low dimensional encoding, and their success at the recognition stage.

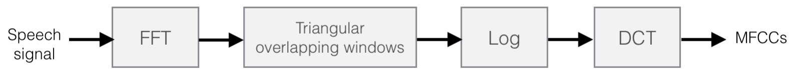


Figure 2.2: Mel-Frequency Cepstral Coefficients block diagram.

Once the MFCC computed their first- and second-order temporal differences are concatenated and the final vector is given as input to the pattern recognition system.

### 2.1.2 Pattern recognition block

The recognition can be realized at several levels: phones, triphones or words. In this thesis we work only on phone recognition as our goal is to demonstrate an improvement in recognition due to the development of more robust features. To deal with the temporal variability of speech the most current ASR systems use Hidden Markov Models combined with Gaussian Mixture Models to model the probability distributions over vectors of features. This model of probability distributions is referred to as the Acoustic Model.

## 2.2 Deep learning in Speech recognition

With the advances made in detection and classification while using machine learning powerful techniques, the speech recognition community started to use Deep neural networks in ASR systems [3]. Basically, deep learning finds its origins within the Neurosciences and has been contributing to many different topics as shown in Figure 2.3.

### 2.2.1 What is deep learning? [3]

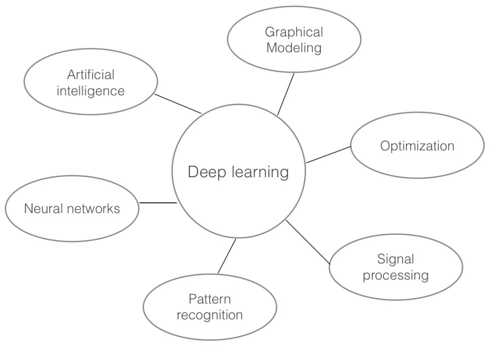


Figure 2.3: Deep learning is at the center of many research areas

Deep learning is a hierarchical learning. That is to say that there are many layers of non-linear information which represent different level of abstraction, and that the whole concept is defined from the lower levels to build a high-level structure. A basic system is exposed in Figure 2.4. Considering the input nodes as $x_i$, an activation function $f$ and the weights $w_j^i$ the hidden nodes $y_j$ are calculated as follows: $y_j = f(\sum_i w_j^i * x_i)$

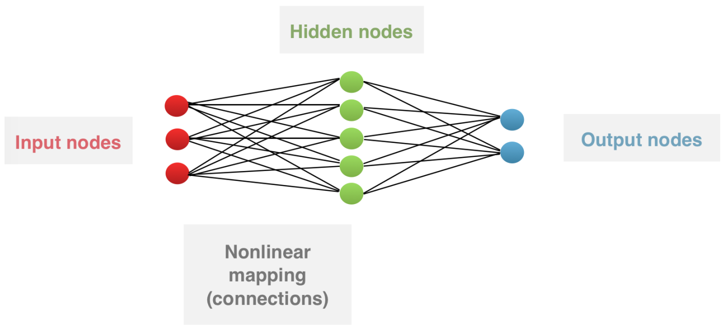


Figure 2.4: Layered network

Two types of learning exist: the supervised learning and the unsupervised learning. Approximately and to simplify we can say that supervised learning is a learning in which either your data is labeled or you have a target so you have a simple cost that depends on the difference between the output and the target to optimize. In the case of unsupervised learning your data is not labeled and the model you learn is generative (in opposition to the discriminative model that corresponds to supervised learning). The recurrent systems used in speech recognition are the hybrid deep networks that bring together supervised and unsupervised learning. Basically, the network is pre-trained in an unsupervised way to boost the effectiveness of the supervised training. This pre-training can be seen as a very efficient way to initialize the weights of the whole network. Then the backpropagation algorithm (detailed in the next chapters) fine-tunes the network and this consists in the supervised learning. The hybrid strategy is a response to an optimization issue that appears as the depth of the networks increases and which is the trapping of local optima.

### 2.2.2 DNN-HMM systems

From early 21$^{st}$ century a new form of acoustic models based on Deep Neural Networks was introduced [3] [9]. Before that, Speech recognition was dominated by GMM-HMM systems. The main improvements brought by DNN are the ability to model data correlation (in this case, feature representation and classification are associated) and also the ability to model nonlinear data. Indeed, when it comes to modeling nonlinear data GMMs become statistically inefficient because it would require a very large amount of gaussians. DNN allows as well to improve the robustness of speech recognition when using DNN-HMM systems as demonstrated in [21]. Nevertheless, the successful performance of GMM-HMM systems has made it difficult for other methods to outperform them. Deep learning algorithms and parameters have had to be tuned a lot before coming close to GMM-HMM performance and overstep it.

### 2.2.3 NN for robust features

The principal motivation for working on better feature extraction systems is that the most relevant the feature is, the better the results are at the recognition stage. Papers [27], [19] and [20] explore respectively new methods to obtain bottleneck features, speaker adaptive features and raw waveform features. In those researches, they focus on DNN-HMM systems in which the novel features are used to learn better acoustic models. Article [14] investigates the performance of nonlinear features of spectrograms that are given as input to a DNN-HMM system for ASR. [18] demonstrates that robust stacked autoencoders are capable of learning robust representations on noisy data. Finally [5] show the efficiency of denoising deep autoencoders on noisy MFCC features.

Thus, DNNs have been used for different purposes in Speech recognition systems. For this thesis, we chose to work on the application of autoencoders and deep autoencoders in feature extraction in order to create robust features.

# 3 Developing tools and architecture

## 3.1 Toolbox of Deep Learning

### 3.1.1 General overview

The Matlab deep learning toolbox used in this thesis [16] presents different libraries for different types of neural networks (NN, CNN, DBN, SAE, CAE) as well as tests and a range of functions. In this section I describe the mathematics useful for my work and the one that are behind this deep learning toolbox.

### 3.1.2 Neural networks

A neural network is fed with input features and their labels (or targets in our case) for supervised learning. Basically, the system learns nonlinear layers of information and optimizes the weights using the backpropagation algorithm to minimize the error between the output and the target. The nonlinearity is introduced in the system through an activation function $f$ used to calculate the nodes of the hidden layers i.e. the deep representations of data. The system $\{$weights $W$, biases$\}$ is initialized randomly using a Gaussian distribution or pre-trained using an unsupervised learning method for $W$ and to 1 for the biases.

- The forward pass: *nnff.m* in [16]
  Assume $x$ is a vector of the input nodes with the biases, $f$ is the activation function and the matrix of weights is $W$. For each $j^{th}$ hidden layer the units' outputs are calculated : $h_j = f(h_{j-1} * W)$.
  For the output layer, it is the output function that is used instead of the activation function $f$. The error and loss are also computed as follows: Assume a target $y$ and an output $z = F_{NN}(x, W)$, the error is $e = y - z$ and the square-loss is $l = \sum \frac{1}{2}.e^2$.

- The backward pass : *nnbp.m* in [16]
  The aim is to find the weights $W$ that minimize the training error loss $L = \sum_{X,Y} l(Y, F_{NN}(X, W))$, where X and Y are respectively matrices of input features and target features.The derivatives are calculated using the delta-rule.

Derivative of the error with respect to the unit:

$$\frac{\delta e}{\delta h_j} = -e_j \tag{3.1}$$

Derivative of the unit with respect to the net input (partial derivatives) :

$$\frac{\delta h_j}{\delta net_j} = h_j(1 - h_j) \tag{3.2}$$

Derivative of the net input with respect to a weight:

$$\frac{\delta net_j}{\delta w_{jk}} = h_k \tag{3.3}$$

And finally for a hidden to output weight

$$\Delta w_{jk} = -e_j \times h_j(1 - h_j) \times h_k \tag{3.4}$$

and for an input to hidden weight

$$\Delta w_{ki} = (\sum_j -e_j \times h_j(1 - h_j) \times w_{jk}) \times h_k(1 - h_k) \times h_i \tag{3.5}$$

The Dropout technique can be used at this stage. It has been introduced by G. E. Hinton [12] and reduces the overfitting and improves the neural networks's training. It basically consists in omitting a part of the features in each training case by setting some units to zero.

The gradients are then applied in *nnapplygrads.m* [16] according to the Stochastic Gradient Method and with tuning options on learning rate ($\mu$) and momentum ($\alpha$).

$$\Delta W = -\mu \Delta W + \alpha \Delta v W \tag{3.6}$$

$$W = W + \Delta W \tag{3.7}$$

where $\Delta vW$ is an accumulated memory of the previous gradients, $\mu$ is the learning rate and $\alpha$ is the momentum.
The constant learning rate allows us to control the rate of learning of the weights i.e. only a ratio of the calculated gradient is taken for update. If the learning rate is too high, it can happen that the training loss explodes (we overstep) and if the learning rate is too low the training loss does not go down or very slowly and it takes longer to train. The momentum can also be introduced. It accelerates the gradient descent by adding in to the $\Delta W$ some of the last weights' adjustments.

All those steps are repeated on a pre-defined number of epochs. For each epoch the forward-pass and backward-pass are performed on all the training data. But, the data is separated into batches (subdivided amounts of data) to feed the backpropagation algorithm. That is to say that to complete one epoch, the number of iterations of the backpropagation algorithm for a database of N speech samples is $\frac{N}{batchsize}$.

Notice that when working on speech recognition, the most common activation function is the logistic sigmoid function (discussed further in the implementation chapter). We also mentioned the pre-training of NNs in the previous chapter. Indeed, the weights need to be well-initialized to avoid the system to get stuck in a local optimum. This is usually done by using unsupervised learning and more precisely Restricted Boltzmann Machines.

### 3.1.3 Pre-training with Restricted Boltzmann Machines [11]

RBMs are energy-based models used as generative models of many different types of data including MFCC. They are used to compose Deep Belief Networks which are a combination of several RBMs and a DNN. Indeed RBMs are an efficient pre-training procedure to NNs. A Restricted Boltzmann Machine is a two-layer network in which stochastic visible units that represent observations are connected to stochastic binary hidden units. As for speech recognition the visible units are real-valued inputs we use Gaussian-Bernouilli RBMs i.e. the hidden units are binary but the input units are linear with gaussian noise. In a RBM there are no visible-visible or hidden-hidden connections and that is why it is called a *restricted* system.[19]

#### Bernouilli-Bernouilli RBM

The joint configuration of the visible (v) and hidden (h) units is given via an energy function for Bernouilli-Bernouilli RBM:

$$E(v,h) = -\sum_{i \in visible} a_i v_i - \sum_{j \in hidden} b_j h_j - \sum_{i,j} v_i h_j w_{ij} \tag{3.8}$$

where $v_i$,$h_j$ are the binary states of visible unit $i$ and hidden unit $j$ and $a_i$,$b_j$ are their bias terms and $w_{ij}$ is the weight between them.

The probability that the network assigns to a visible vector $v$ is:

$$p(v) = \frac{\sum_h e^{-E(v,h)}}{\sum_{v,h} e^{-E(v,h)}} \tag{3.9}$$

And finally, the limited connections within a RBM make the conditional distributions $p(v|h)$ and $p(h|v)$ quite straight forward:

$$p(h_j = 1|v) = \sigma(b_j + \sum_i v_i w_{ij}) \tag{3.10}$$

$$p(v_i = 1|h) = \sigma(a_i + \sum_j h_j w_{ij}) \tag{3.11}$$

where $\sigma$ is the logistic sigmoid function $1/(1 + \exp(-x))$.

In theory, the update rule for the weights is

$$\Delta w_{ij} = \epsilon(< v_i h_j >_{data} - < v_i h_j >_{model}) \tag{3.12}$$

where $< \cdot >_X$ denotes the expectation computed over the indicated distribution.

However in practice, obtaining $< v_i h_j >_{model}$ is difficult and that is why the Contrastive Divergence approximation to the gradient is used instead [10] and the new update rule becomes:

$$\Delta w_{ij} = \epsilon(< v_i h_j >_{data} - < v_i h_j >_{recon}) \tag{3.13}$$

where $< v_i h_j >_{recon}$ is obtained after initialization of the states of the visible units to a training vector, and then an update of the binary states of the hidden units with Eq.(3.10) and finally a setting to 1 of each $v_i$ with the probability from Eq(3.11).

**Gaussian-Bernouilli RBM [25]**

In case of a Gaussian-Bernouilli RBM, the energy function becomes:

$$E(v,h) = -\sum_{i \in visible} \frac{(v_i - a_i)^2}{2\sigma_i^2} - \sum_{j \in hidden} b_j h_j - \sum_{i,j} \frac{v_i}{\sigma_i} h_j w_{ij} \tag{3.14}$$

where $\sigma_i$ is the standard deviation of the Gaussian noise for visible unit $i$.

As it is difficult to learn the variance of the noise for each visible unit, the data is normalized to zero mean and unit variance in practice.

Conditional probabilities for visible and hidden units are

$$p(h_j = 1|v) = \sigma(b_j + \sum_i w_{ij} \frac{v_i}{\sigma_i^2}) \tag{3.15}$$

$$p(v_i = v|h) = N(v|a_i + \sum_j h_j w_{ij}, \sigma_i^2) \tag{3.16}$$

where $N(\cdot|\mu, \sigma)$ denotes the Gaussian probability density function with mean $\mu$ and standard deviation $\sigma$.

As for the Bernouilli-Bernouilli, the CD learning is used to train the RBM's parameters and the number of steps is usually set to 1 (CD1). The update rule becomes:

$$\Delta w_{ij} = \epsilon(< \frac{1}{\sigma_i^2} v_i h_j >_{data} - < \frac{1}{\sigma_i^2} v_i h_j >_{recon}) \tag{3.17}$$

$$\Delta a_i = \epsilon(< \frac{1}{\sigma_i^2} v_i >_{data} - < \frac{1}{\sigma_i^2} v_i >_{model}) \tag{3.18}$$

$$\Delta b_j = \epsilon(< h_j >_{data} - < h_j >_{model}) \tag{3.19}$$

The toolbox [16] offers a package to build DBNs with a training of RBMs with CD1. Unfortunately it allows only Bernouilli-Bernouilli RBMs and that it why I became interested in another deep learning toolbox [22] discussed further in chapter 5.

Fig.3.1 shows how RBMs are used to initialize the weights of a DNN and form a DBN. The example is given with a $d$-layer network $[n_1...n_i...n_d]$ where $n_i$ is the number of nodes of layer $i$. The first Gaussian-Bernouilli RBM $[n_1 n_2]$ is trained with feature vectors as inputs and the second Bernouilli-Bernouilli RBM $[n_2 n_3]$ is trained with the output of the first RBM as input. That is repeated for all the layers of the wanted DBN and then the weights are unfolded to a DNN that is fine-tuned with backpropagation.

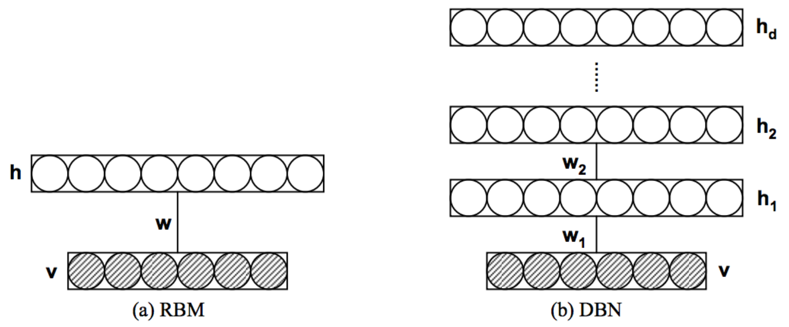


Figure 3.1: Construction of a DBN. (figure extracted from [15])

### 3.1.4 Autoencoders

Autoencoders are a special type of DNN whose input dimension is the same as the output one. They are trained to encode the input into some high-level or compressed representation so that it can be reconstructed from that representation. Hence, the output target is the input. [1]

An AE is composed of an encoder that encodes the input signal into a hidden layer which is a nonlinear representation of the input:

$$h = f_\theta(x) = \sigma(Wx + b) \tag{3.20}$$

with parameters $\theta = \{W, b\}$ (W is the weights matrix and $b$ an offset factor), the input vector $x$ and the parameterized function $\sigma$.

This deterministic mapping $f$ is then mapped back to a reconstructed vector $y$ that has the same dimension as the input vector.

$$y = f_{\theta'}(h) = \sigma(W'h + b') \tag{3.21}$$

with parameters $\theta' = \{W', b'\}$ and parameterized function $\sigma$.

Notice that the parameterized functions of the encoder and the decoder can be different. In [24] they use either an {affine+sigmoid} encoder with either affine decoder with squared error loss or {affine+sigmoid} decoder with cross-entropy loss. Typically, the input of an AE is a feature vector and the output is the reconstruction of this feature. In between, one or more (deep autoencoder) hidden layers represent a transformation of the feature. Usually, AE and

DAE are trained using backpropagation and SGD. To avoid the problems of backpropagation brought up previously, each layer can firstly be trained as an autoencoder in the case of a deep autoencoder (more than one hidden layer).

Using [16] we can build an autoencoder as a three-layer NN in which the input signal is also the target signal.

### 3.1.5 Denoising auto-encoders

[24] A denoising autoencoder is a variant version of the autoencoder described before. It is trained to reconstruct a *clean* version from the corrupted signal given as input. The input signal $x$ is first corrupted into $\hat{x}$. Then $\hat{x}$ is encoded $h = f_\theta(\hat{x}) = \sigma(W\hat{x} + b)$ and decoded $y = f_{\theta'}(h) = \sigma(W'h + b')$ and the reconstruction loss is calculated between the clean version $x$ and the reconstructed signal $y$. Hence, the system learns a clever mapping that denoises signals of the type of the inputs used for training.

Using the same method as the AE we can implement a denoising autoencoder with [16]. The *inputZeroMaskedFraction* parameter allows us to add noise at a certain ratio.

All these tools provide us with the opportunity to build more robust features to be given as input to a pattern recognition system.

## 3.2 Speech recognition tools

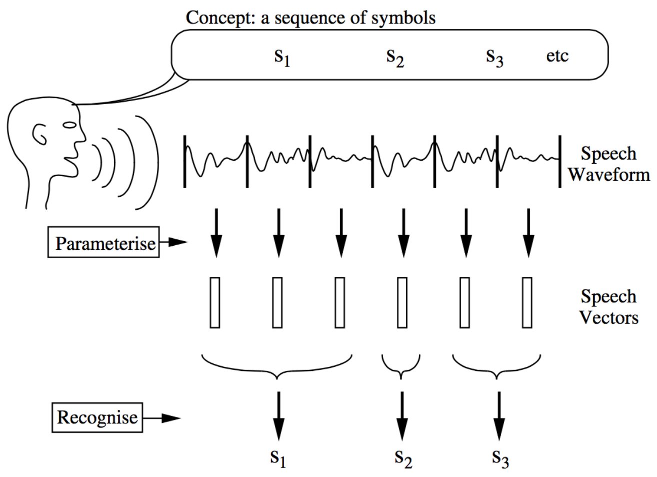


Figure 3.2: Recall of the aim of a speech recognition system. (figure extracted from [26])

The goal of a recognition tool is to process a speech sequence into speech vectors and to deduce from these features the most likely corresponding sequence as shown in Fig 3.2. For the purpose of this thesis, I tested two different free, open-source toolkits that perform speech recognition: HTK [26] and Kaldi [17]. Both allow us to compute the state-of-the-art speech features MFCC and to perform speech recognition using GMM-HMM. In this section I first recall the theory behind a speech recognition system, and then I focus on the Kaldi toolkit.

### 3.2.1 Basic scheme of a speech recognition system

#### HMM-based acoustic flat model

A spoken word $w$ is a sequence of phones $K_w$. It happens that different sequences of phones define the same word $w$ due to a different pronunciation. That is why we consider the likelihood $p(\mathbf{Y}|\mathbf{w})$ over multiple pronunciations $\mathbf{Q}$ [7]:

$$p(\mathbf{Y}|\mathbf{w}) = \sum_{Q} p(\mathbf{Y}|\mathbf{Q})p(\mathbf{Q}|\mathbf{w}) \tag{3.22}$$

where for a particular sequence of pronunciation $\mathbf{Q}$ and for $q^{(wl)}$ a valid pronunciation for word $w_l$,

$$p(\mathbf{Q}|\mathbf{w}) = \prod_{l=1}^{L} P(\mathbf{q}^{(wl)}|w_l) \tag{3.23}$$

Each phone is represented by a density Hidden Markov Model with transition probability parameters $\{a_{ij}\}$ and output distributions $\{b_j()\}$ as pictured in Fig.3.3. In the figure the states $x_i \in 1, 2, 3, 4, 5$.

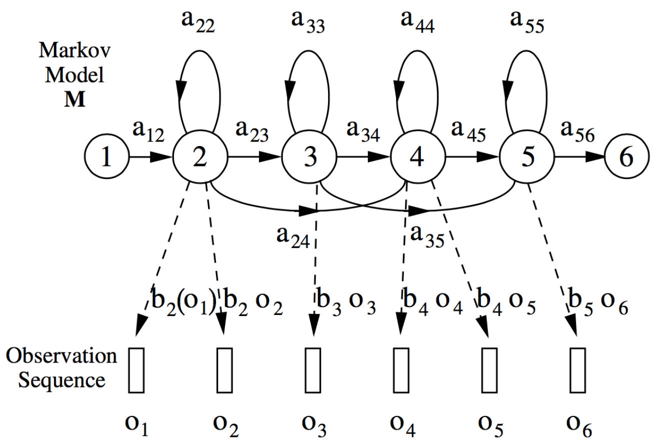


Figure 3.3: HMM-based phone model. (figure extracted from [7])

A Markov model is a finite state machine which changes state once every time step and each time a new speech vector is generated from the probability density $\{b_j()\}$. The HMM change of state or transition is managed from its current state $x_i$ to one of its connected states $x_j$ according to the transition probability $\{a_{ij}\}$ each time step. In practice only the observation $\mathbf{O}$ is known, the state sequence $\mathbf{X} = x_1, ...x_T$ is hidden [26]. The likelihood is the sum over all possible state sequences of the joint probability $P(\mathbf{O}, \mathbf{X}|\mathbf{M})$:

$$p(\mathbf{O}|\mathbf{M}) = \sum_X a_{x(0)x(1)} \prod_{t=1}^{T} b_{x(t)}(o_t) a_{x(t)x(t+1)} \tag{3.24}$$

where $x(0)$ and $x(T+1)$ are constrained to be respectively the entry-state and the exit-state. The most probable likelihood is $\max(p(\mathbf{O}|\mathbf{M}_k))$. Given a set of training examples, the parameters of the models can be determined using the re-estimation procedure detailed below.

### Output distribution: GMM

First, we define the output distributions $b_j(o_t)$ by a Gaussian Mixture Model:

$$b_j(o_t) = \sum_{m=1}^{M} c_{jm} \times \mathcal{N}(o_t; \mu_{jm}, \Sigma_{jm}) \tag{3.25}$$

where M is the number of gaussians, $c_{jm}$ is the weight of gaussian $m$ and $\mathcal{N}(\cdot; \mu, \Sigma)$ is a multivariate Gaussian with mean vector $\mu$ and covariance matrix $\Sigma$:

$$\mathcal{N}(o; \mu, \Sigma) = \frac{1}{\sqrt{(2\pi)^n \|\Sigma\|}} \times e^{-\frac{1}{2}(o-\mu)'\Sigma^{-1}(o-\mu)} \tag{3.26}$$

with $n$ the dimension of the observation $o$.

### Re-estimation with Baum-Welch algorithm

First, the parameters of the HMM are initialized. It is usually done by using the global mean and covariance of training data in the output distributions and setting to equality all transition probabilities.
Then the parameters are re-estimated using Baum-Welch algorithm:

$$\hat{\mu} = \frac{\sum_{t=1}^{T} L_j(t) o_t}{\sum_{t=1}^{T} L_j(t)} \tag{3.27}$$

$$\hat{\Sigma}_j = \frac{\sum_{t=1}^{T} L_j(t)(o_t - \mu_j)(o_t - \mu_j)'}{\sum_{t=1}^{T} L_j(t)} \tag{3.28}$$

where $L(t)$ denotes the probability of being in state $j$ at time $t$.
$L(t)$ is calculated using the Forward-Backward algorithm: for a forward probability $\alpha_j(t)$ for a model M with N states, it can be calculated using a recursive method :

$$\alpha_j(t) = P(o_1, ..., o_t, x(t) = j|M) = (\sum_{i=2}^{N-1} \alpha_i(t-1) a_{ij}) b_j(o_t) \tag{3.29}$$

In the same way, the backward probability can be computed:

$$\beta_i(t) = P(o_{t+1}, ..., o_T, x(t) = j|M) = \sum_{j=2}^{N-1} a_{ij} b_j(o_{t+1}) \beta_j(t-1) \tag{3.30}$$

Once those probabilities computed, we can deduce $L_j(t) = \alpha_j(t)\beta_j(t)/P(O|M)$. We update the Gaussian parameters with the new $L_j(t)$ value and according to the value of $P(O|M)$ we re-iterate or stop the process.

**Decoding**

Now that we have good estimates of the transition probabilities, we need to find the best path through the states that estimates the speech and in theory this is done using the Viterbi algorithm and recursive calculations. In Kaldi the decoding is carried out using graphs and decision trees and the method is explained further in the next paragraphs.

### 3.2.2 HTK versus Kaldi

HTK is a toolbox created to build HMM system dedicated to Speech recognition. After I carried out the tutorials of both toolkits I became aware of their strengths and weaknesses. I chose to continue my work using Kaldi because it was running better on my operating system, appeared more flexible to me and that the codes and the architecture were easier for me to understand. Kaldi is written in C++ and also integrates codes for DNN which makes it more complete and modern as speech recognition toolkit.

### 3.2.3 General overview of Kaldi

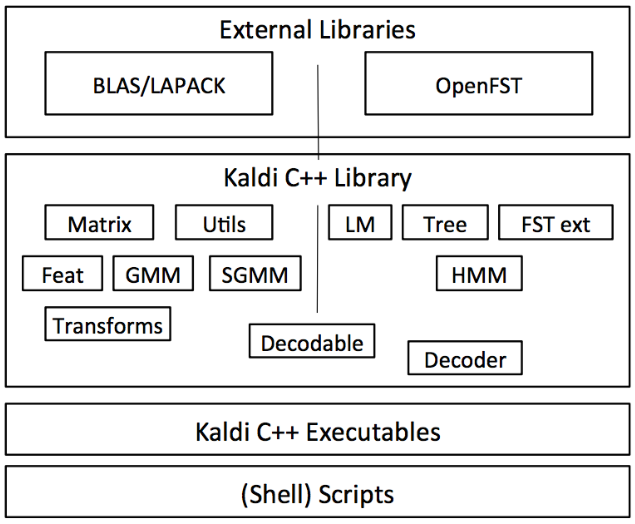


Figure 3.4: Overview of Kaldi tools. (figure extracted from [17])

- External libraries
  BLAS "Basic linear Algebra Subroutines" and LAPACK "Linear Algebra PACKage" are numerical algebra libraries and OpenFST is a library for constructing, combining, optimizing, and searching weighted finite-state transducers (fst) i.e. it permits among other things to the represent a probabilistic model and which is used in Kaldi for the finite-state framework [17].

- Kaldi Library
  The modules of the Kaldi library contain command-line tools to be used for the speech recognition purpose. For instance in the module "*feat*" we can find a command to compute the MFCC or another type of feature.

In this thesis, we used the MFCC library just for the start with Kaldi. Given that the aim of the project is to create more robust features and to assess their performance using a standard pattern recognition system, Kaldi was used only for its GMM-HMM system. Notice also that I used Kaldi tools dedicated to the TIMIT database (further detailed in Chapter 5).

### 3.2.4 From speech to decoding with Kaldi

#### Data , dictionary and language preparation

Data is first prepared with *timit_data_prep.sh*, the path to the TIMIT directory is provided and the program evaluates the list of speakers, finds the list of audio and transcript files and converts them, creates mapping files between speakers and utterances (one utterance is one speech signal) and also a gender mapping and finally writes the STM files necessary when scoring (getting the error rates) with the NIST's sclite tool [6].

*timit_prepare_dict.sh* creates the dictionary which is a sorted list the phones present in the training scripts. And *timit_format_data.sh* does the language preparation which consists in creating a N-gram language model. A N-gram language model provides with the prior probability of a phone sequence $\mathbf{k} = k_{n-1}, ..., k_1$:

$$P(\mathbf{k}) = \prod_{k=1}^{K} P(k_n|k_{n-1}, ..., k_1) \tag{3.31}$$

In practice to form a N-gram LM the product in Eq3.31 is truncated and the formula becomes:

$$P(\mathbf{k}) = \prod_{k=1}^{K} P(k_n|k_{n-1}, ..., k_{n-N+1}) \tag{3.32}$$

In our case, a phone bigram LM is computed using [4] and the data is converted into a "canonical" form and save in binary-format *.fst* files.
In *prepare_lang.sh* a directory is set up. The phones are organized into silence and non-silence categories and the script allows us to remove the optional silence phone *sil* by requiring its probability to be 0. This is done to avoid the scoring of silence phones.

#### Feature extraction

In my work, feature extraction is done in matlab and the vectors are converted into Kaldi format. This Kaldi format is a 2-file format that describes the data:

- *scp* format
  A text file in which each line has a key (utterance id) and extended filename that tells kaldi where to find the data. See Appendix A.2 for examples.

- *ark* format
  A binary file in which each utterance defined by its key has its object data. See Appendix A.1 for examples.

The conversion from Matlab to Kaldi format is detailed in Chapter 5.

#### Training with *train_mono.sh*

The script computes a flat and monophone training with deltas-deltas features (that is to say for instance MFCC+deltas+deltas-deltas). In *gmm-init-mono* a flat-start monophone set is created in which each base phone is a monophone single-Gaussian HMM with means and covariances equal to the mean and covariance of the training data. *shared-phones* option allows common probability density functions for specified sets of phones, otherwise all phones are separated. *compile-train-graphs* compiles the training graphs. Then the statistics of the GMMs are accumulated and a first estimation is done. This process {accumulation of statistics + re-estimation of gaussian parameters using iterations of EM (cf Baum Welch))} is iterated for a fixed number of times. A decision tree is created for each state in each phone and they are exported to graphs that serve for the decoding. The final model is saved under *final.mdl*.

#### Decoding *decode.sh* and scoring *score_basic.sh*

Decoding is performed using the graphs saved at the end of training that contains the language model, the dictionary and the HMM definition. The system check that the feature vectors' dimensions of testing are the same as the ones of training. And *gmm-latgen-faster* is used to decode the testing data and the results are saved in a archive. For scoring, *lattice-best-path* uses the previous results saved in an archive to find the best path of the phone sequence. The resulted phone map is compared to the reference map and the Phone Error Rate or Word Error Rate is computed with *compute-wer*.

# 4 Problem statement

Once I got my hands on those tools, I was able to understand how to play with DNNs and to think of how to improve the current MFFC. The main goal of this thesis is to consider a side-way that was not thought of. Indeed, most authors of the research papers cited until now have been using DNNs, and speech recognition systems for years and most importantly they have at their disposal more computational power than I have. In the following paragraphs I explain the thought process I followed along this thesis.

The first and basic assumption that we made is that the nonlinear mapping learned by a DNN might be more robust to noise than a standard linear mapping. The idea behind this supposition is that the network learns high-level representations of data and so capture the main characteristics of speech. That is why we thought of introducing autoencoders in the standard construction of the MFCC. We began by trying Fig.4.1.

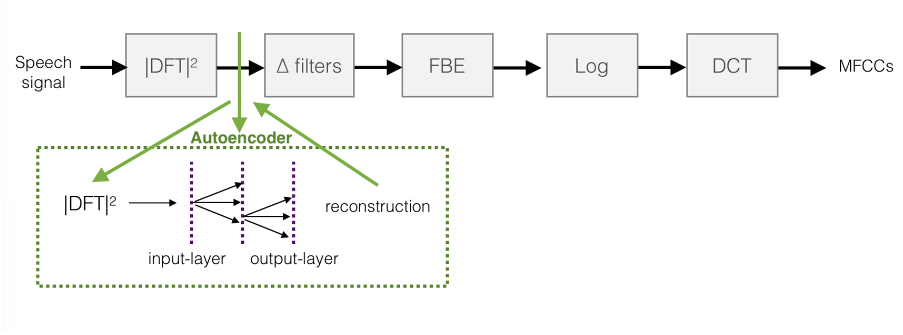


Figure 4.1: Integration of an AE in the MFCC block diagram.

The global idea was that we could finally train NNs to replace blocks of the MFCC computations, see Fig.4.2.

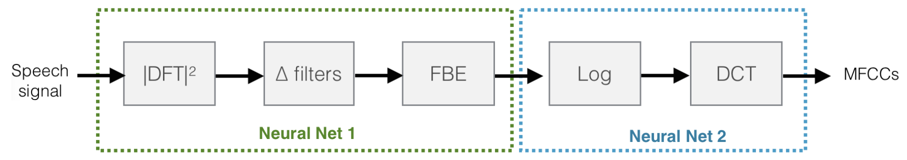


Figure 4.2: Replacement of MFCC blocks by NNs

After a standard training (weights randomly initialized over a gaussian distribution) of the AE shown in 4.1, the SNR between the input feature $x$ and the reconstruction $\hat{x}$ was very low (no more than 2dB). Recall,

$$SNR_{dB} = 10 \times \log_{10}\left(\frac{\|x\|^2}{\|x - \hat{x}\|^2}\right) \tag{4.1}$$

However, the deep learning toolbox [16] only consider binary-binary RBMs so we could not use it efficiently on the real-valued power spectrum. That is why I came to use [22] that permits the training of a Gaussian-Bernouilli RBM. Inspiration for the setting of parameters used for unsupervised pre-training was taken from [11] and [13]. Even with such a pre-training we were unable to increase the SNR efficiently.

The main challenge is to produce features that are robust to noise, so I became interested in denoising autoencoders. Denoising autoencoder are trained to produce a clean reconstruction of a noisy input. The system I built was inspired by [5]. The MFCC are computed and a deep denoising autoencoder is used to "clean" the data. But similarly to my first implementation, the problems with unsupervised learning were still here so I decided to pre-train the first mapping of the denoising AE with an other AE using [16]:

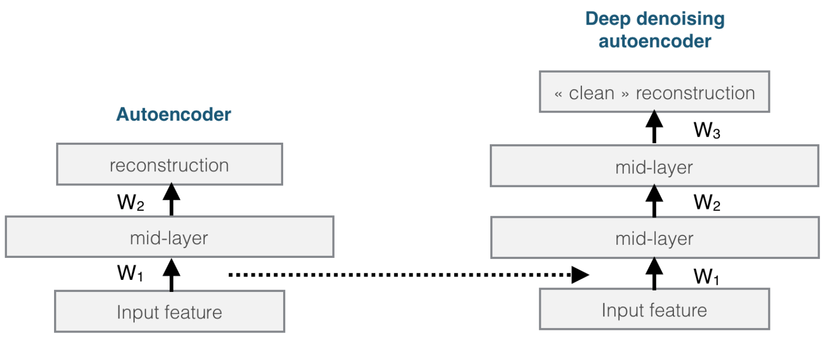


Figure 4.3: Deep denoising autoencoder with pre-training

On this working base, we thought of improving the performance of the deep denoising autoencoder by using a clustering method. The data would be divided using a standard clustering algorithm and each group would train a specific deep autoencoder. The assumption is that the less the data varies the more efficient is the mapping of the DDAE.

The explanation on the implementation and the experiments' setting of those different attempts are detailed in the next chapter.

# 5 Experiments and implementation

All the implementation was done using Matlab and Xcode. A global view of the implementation is available in Fig.5.1.

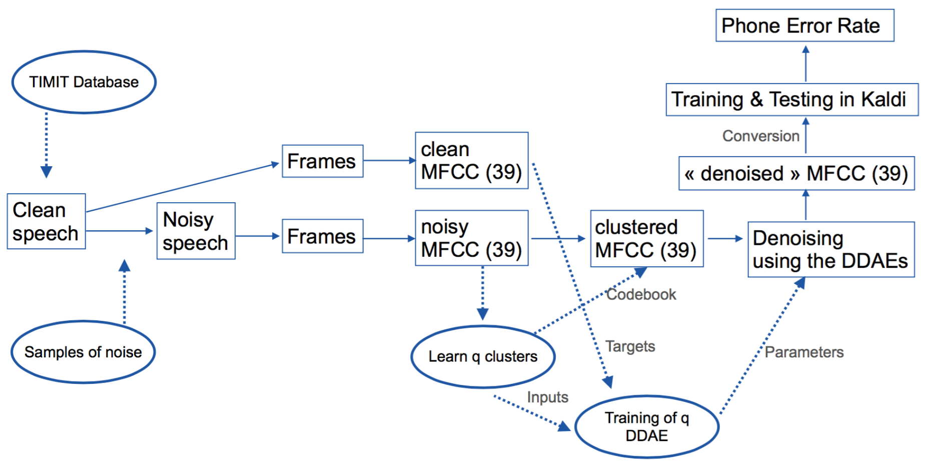


Figure 5.1: Global scheme of the system

First I describe the speech database used in the experiments and then I detail the implementation.

## 5.1 TIMIT database

The DARPA TIMIT Acoustic-Phonetic Continuous Speech Corpus provides the user with an speech database in the English language. It contains 10 sentences spoken by 630 different speakers from 8 major dialect regions of the United States. The database is organized as this : 8 directories $\{dr1, ..., dr8\}$ that correspond to the different dialects. Into a directory drN, the data is divided into folders corresponding to the speakers and referenced with a {3-letter + digit} code e.g. "DAB0".

For one speech signal i.e. one sentence spoken by one particular speaker, the TIMIT corpus includes 4 different files:

- *.wav*: the 16kHz speech signal named after the sentence code e.g. "SA1".
- *.txt*: transcription file of the words the person said (the sentence).
- *.wrd*: transcription file of the time-aligned words the person said.
- *.phn*: transcription file of the time-aligned phones the person said.

Samples of the transcription files are shown in B.1,B.2 and B.3 respectively. And the code to load the data into Matlab is detailed in C.1. Not all testing files from TIMIT were use for the experiments, only 192 utterances were selected according to the basic *run.sh* kaldi example script for TIMIT: for each dialect region 3 speakers are selected and for each one of them we consider 8 utterances.

## 5.2 AE integrated in MFCC computational line

### 5.2.1 Framing setting

Frame length: 25ms
Frame shift: 10ms
These values are standard ones for speech recognition and there are the ones used by default in Kaldi's feature extraction system.

### 5.2.2 Basic MFCCs computation

The MFCC implementation is based on [2] and the global codes are detailed in C.2.1 and C.2.2.
We start by implementing the basic MFCC. As in Kaldi, the number of triangular filters is set to $M = 23$ and the feature dimension is set to $Q = 12$. The log-energy is added as the $13^{th}$ coefficient of the MFCC and finally the deltas and deltas-deltas are computed and concatenated to the MFCC to form a 39-dimension feature vector. Recall, the deltas and deltas-deltas are computed between the frames, they are the first and second order frame-to-frame differences.

The filterbanks are created as follows: first the start-, center- and end-frequencies of each filter are calculated in the mel-domain; then they are transformed back to the normal frequency domain to be turned into the sample scale; finally each filter is calculated in the normal frequency domain using standard lines' equations. Code for computing the filterbanks is available in C.2.3. Notice that the Hamming window of 5.2 is applied before calculating the power-spectrum C.2.4.

### 5.2.3 Integration of the AE in the basic scheme

At first, when we integrate the AE in the basic MFCC scheme we took the mid-layer nodes as the new feature as shown in Fig.5.3

#### Improvement attempts

The script used with the second toolbox is detailed here C.3.1. To understand the issues of the pre-training with RBMs, we tried different sizes of input features. Particularly, I created a system in which each FBE of a frame was the

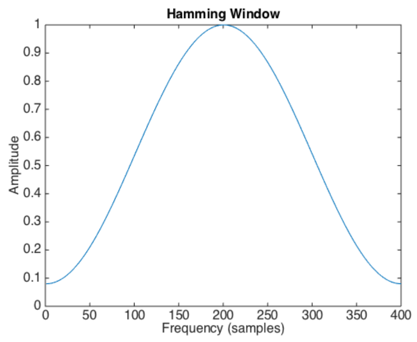


Figure 5.2: Plot of the Hamming window

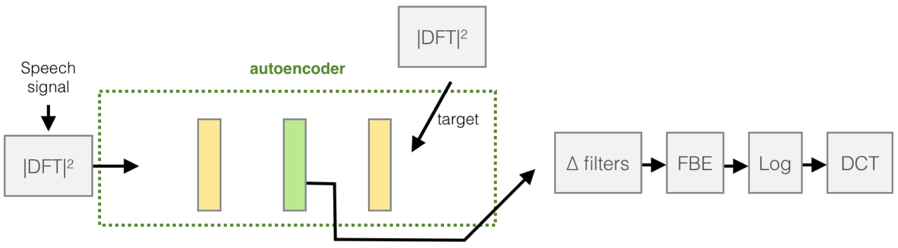


Figure 5.3: Integration of AE in the MFCC computations

input feature of a set of autoencoders. In that way, the input-size of the NN was reduced and it included between 3 and around 10 samples. Even then the SNR stayed very low. Indeed it seemed that the parameters could not learn a mapping correctly because the variance of data was very large: for some vectors the reconstruction was quite good but for others it was very bad.

After this observation, we decided to concatenate the obtained features with the standard MFCC but still the performance of recognition was poorer than when only the MFCC were used. Even when using a coefficient-to-coefficient concatenation to reduce the correlation of the resulted vector. Then we switched to use the output of the AE as the input to the rest of the MFCC block diagram, but there was no difference at all. We also moved the AE along the MFCC block diagram but there was no difference in performance. I concluded there was something in pre-training and training that was going wrong but I could not put my hand on it. Finally I decided to bypass the issues created by the RBM pre-training and I focused on denoising autoencoders.

## 5.3 Denoising DNN with supervised pre-training

According to [5] we created a deep denoising autoencoder. Our method differs in the pre-training for which we use an AE to initialize the first weight matrix $W_1$ of the network as shown in Fig.4.3.

### 5.3.1 Noisy signals

The TIMIT database only provides us with clean speech. To realize the training and testing of our system, we added noise to the data. Three different types of noises were used: gaussian white noise, car noise and babble noise. For the sake of reflecting reality conditions, the noisy signals were built at 3 different levels of SNR, 5dB, 10dB and 15db for each type of noise. The code used to create noisy signals is detailed in C.4.1

### 5.3.2 Final solution: denoising deep autoencoder with supervised pre-training and VQ

Using the DDAE described above, we introduce vector quantization in the process. The training data is clustered using the K-means algorithm and the codebook is saved and used for denoising the testing data. Experiments were run for the number of centroids $q = \{2, 4, 8, 16\}$.

### 5.3.3 K-means theory

The K-means algorithm is quite simple. For a start, random centroids are picked up. Then the euclidian-distance between all observations and the centroids are computed and the affiliation to a cluster is defined by the minimum distance to a centroid. The new centroids of the clusters are re-calculated. These two steps are repeated until convergence.

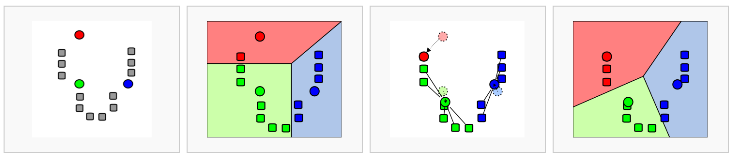


Figure 5.4: Demonstration of the K-means algorithm extracted from https://en.wikipedia.org/wiki/K-means_clustering

### 5.3.4 NN setting

For what concerns the parameters of the DDAE:

#### The activation function

The sigmoid function is the most common activation function used for NNs in speech applications. It is defined over $[-\infty, +\infty] \rightarrow ]0, 1[$ by

$$S(t) = \frac{1}{1 + e^{-t}} \tag{5.1}$$

#### The pre-training autoencoder

The layers are of sizes [39 100] since the input is the noisy 39-dimension MFFC.
The activation function equals the output function and is the Sigmoid.
A noisy factor of 0.5 is added.

The learning rate is set to 0.5 and the momentum is also set to 0.5.
The optimization is run on 25 epochs with a batchsize of 250.

#### The DDAE

The layers are of sizes [39 100 100 39] since the input is the noisy 39-dimension MFFC and the output is its "clean" reconstruction.
The activation function is the sigmoid (encoder) and the output function is affine (decoder).
The learning rate is set to 0.004 and the momentums to 0.5.
The optimization is run on 50 epochs with a batchsize of 250.

The code for the training is detailed in C.4.2

### 5.3.5 Kaldi : conversion

After the computation of the denoised features in Matlab, the training and testing data are converted to Kaldi format. And the tool "kaldi-to-matlab" [23] is used in C.4.3 to create the *.ark* and *.scp* files mentioned earlier.

### 5.3.6 Kaldi setting

For what concerns the parameters of Kaldi: max_iter_inc=30 (last iteration to increase gaussian on)
totgauss=1000 (number of target gaussians)
num_iters=40 (number of iterations of training(
realign_iters="1 2 3 4 5 6 7 8 9 10 12 14 16 18 20 23 26 29 32 35 38" (iterations for re-alignment)
boost_silence=1.0 (factor by which to boost silence likelihoods in alignment)
beam=6
careful=false (alignment option)

I do not divide the data for HMM optimization so {train_nj=1, test_nj=1}.
The reason why we chose to run a very simple recognition using a monophone system is because the aim here is to compare the performance of our features to the state-of-the-art MFFC. As long as we have the performance of the basic { MFFC + GMM-HMM } association in this system we can compare it to the association {our feature + GMM-HMM}.
You can notice in looking at the code in C.4.4 that I modified a bit the code usually provided by Kaldi for TIMIT recognition. Indeed I wanted to feed the system with my own features and to do so I was forced to remove the automatic calculations of deltas and deltas-deltas in Kaldi.

# 6 Results

By using the command "bash RESULTS test" kaldi displays the word error rate (expressed in percentage) of all tested systems in the terminal. In our case, we do consider the phone error rate instead that is defined by

$$PER = \frac{S + D + I}{N} \tag{6.1}$$

where $S$ is the number of substitutions, $D$ the number of deletions, $I$ the number of insertions and $N$ the total number of phones in data.

## 6.1 Effect of the VQ of the feature vectors on denoising

We can start with looking at the statistics of the data VQ with Table.6.1.

Table 6.1: Data clustering statistics

| clusters | weights of each cluster in the total data | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| $q = 2$ | 0.52 | 0.48 | | | | | | |
| $q = 4$ | 0.28 | 0.16 | 0.27 | 0.29 | | | | |
| $q = 8$ | 0.15 | 0.016 | 0.14 | 0.15 | 0.18 | 0.09 | 0.11 | 0.16 |
| $q = 16$ | 0.07 | 0.01 | 0.06 | 0.06 | 0.1 | 0.04 | 0.08 | 0.03 |
| | 0.06 | 0.01 | 0.08 | 0.1 | 0.07 | 0.08 | 0.06 | 0.08 |

We can see that for $q = \{2, 4\}$ the data is represented quite equitably. However when q increases some clusters become really small. This can involve issues in the training of the neural network. Indeed, at least one frame is needed to learn a parameter. In our case, the NN must learn around $39 \times 100 + 100 \times 100 + 100 \times 39 \approx 18000$ parameters. For the sake of training, the amount of training data given to the NN was calculated to allow the learning of a parameter over 5 frames. When the number of clusters $q$ was increased, the amount of training data was increased accordingly to match the 5 frames per parameter.

For each clustering level, we plot the SNR of the output of the DDAE as a function of $q$ to evaluate the impact of clustering on the efficiency of the DDAE. The results are shown in Fig.6.1.

We notice that for $q = 4$ the system must have encounter an issue. Apart from that, the general assumption that clustering data improves the efficiency of a global DDAE system seems to be verified.

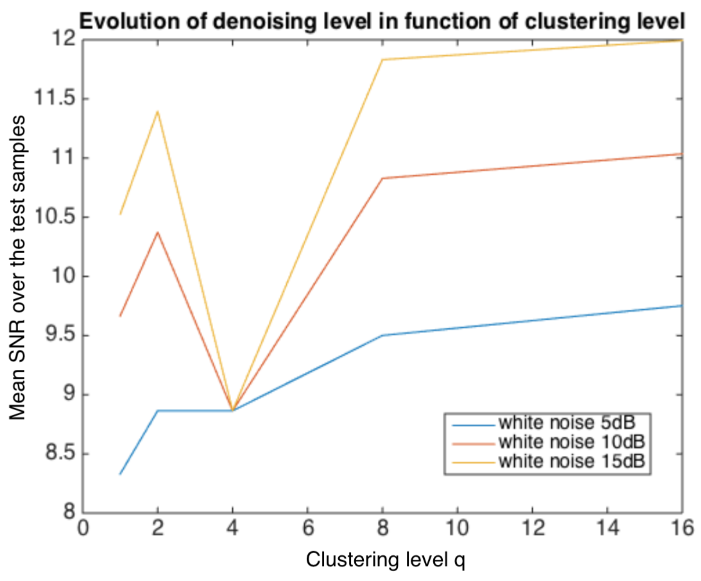
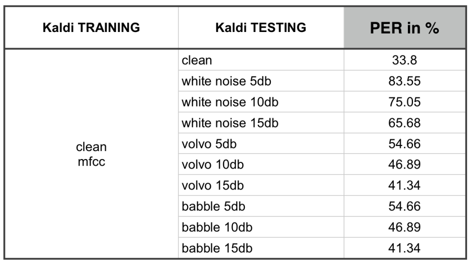


Figure 6.1: Evolution of denoising level in function of clustering level $q$

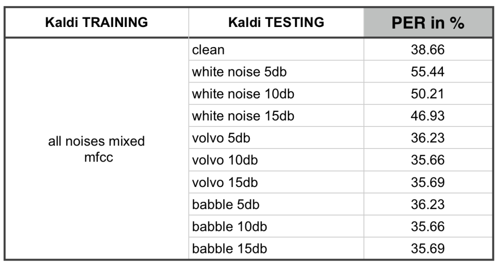
## 6.2 Effect of the VQ of the feature vectors on speech recognition

Recall, the tests were computed over a range of 3 different noises added to the clean speech at different levels: 5dB, 10dB and 15db. Fig.6.2 shows the reference PER to which we can compare the PER achieved by our new denoised feature vectors. Fig.6.3 shows the results in terms of PER of our speech recognition system for $q = \{1, 2, 4, 8, 16\}$



| Kaldi TRAINING | Kaldi TESTING | PER in % |
|---|---|---|
| clean mfcc | clean | 33.8 |
| | white noise 5db | 83.55 |
| | white noise 10db | 75.05 |
| | white noise 15db | 65.68 |
| | volvo 5db | 54.66 |
| | volvo 10db | 46.89 |
| | volvo 15db | 41.34 |
| | babble 5db | 54.66 |
| | babble 10db | 46.89 |
| | babble 15db | 41.34 |

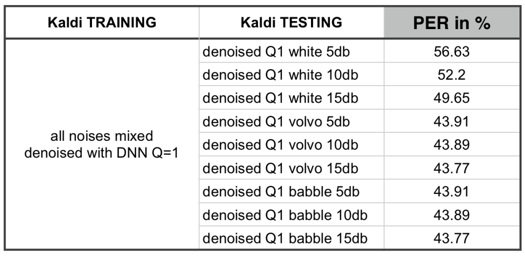
(a) Clean training



| Kaldi TRAINING | Kaldi TESTING | PER in % |
|---|---|---|
| all noises mixed mfcc | clean | 38.66 |
| | white noise 5db | 55.44 |
| | white noise 10db | 50.21 |
| | white noise 15db | 46.93 |
| | volvo 5db | 36.23 |
| | volvo 10db | 35.66 |
| | volvo 15db | 35.69 |
| | babble 5db | 36.23 |
| | babble 10db | 35.66 |
| | babble 15db | 35.69 |

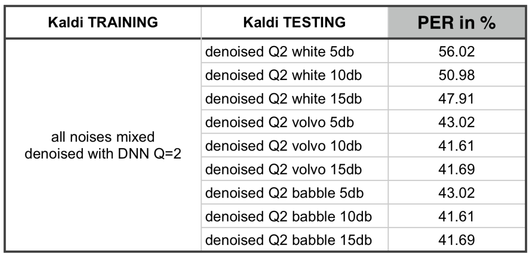
(b) Noisy training

Figure 6.2: Results in terms of PER (%) for speech recognition



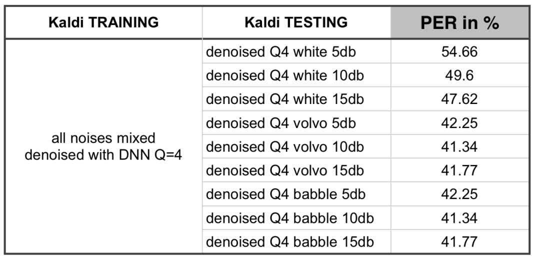
| Kaldi TRAINING | Kaldi TESTING | PER in % |
|---|---|---|
| all noises mixed denoised with DNN Q=1 | denoised Q1 white 5db | 56.63 |
| | denoised Q1 white 10db | 52.2 |
| | denoised Q1 white 15db | 49.65 |
| | denoised Q1 volvo 5db | 43.91 |
| | denoised Q1 volvo 10db | 43.89 |
| | denoised Q1 volvo 15db | 43.77 |
| | denoised Q1 babble 5db | 43.91 |
| | denoised Q1 babble 10db | 43.89 |
| | denoised Q1 babble 15db | 43.77 |

(a) Denoised training $q = 1$



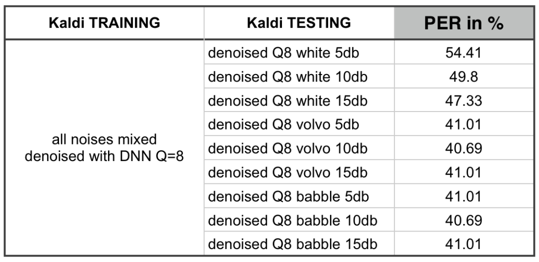
| Kaldi TRAINING | Kaldi TESTING | PER in % |
|---|---|---|
| all noises mixed denoised with DNN Q=2 | denoised Q2 white 5db | 56.02 |
| | denoised Q2 white 10db | 50.98 |
| | denoised Q2 white 15db | 47.91 |
| | denoised Q2 volvo 5db | 43.02 |
| | denoised Q2 volvo 10db | 41.61 |
| | denoised Q2 volvo 15db | 41.69 |
| | denoised Q2 babble 5db | 43.02 |
| | denoised Q2 babble 10db | 41.61 |
| | denoised Q2 babble 15db | 41.69 |

(b) Denoised training $q = 2$



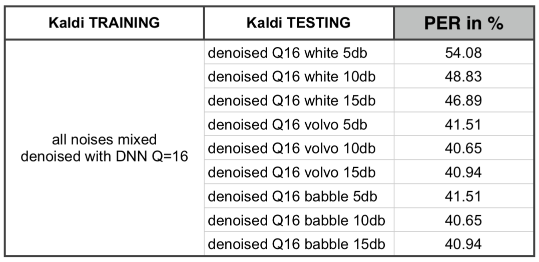
| Kaldi TRAINING | Kaldi TESTING | PER in % |
|---|---|---|
| all noises mixed denoised with DNN Q=4 | denoised Q4 white 5db | 54.66 |
| | denoised Q4 white 10db | 49.6 |
| | denoised Q4 white 15db | 47.62 |
| | denoised Q4 volvo 5db | 42.25 |
| | denoised Q4 volvo 10db | 41.34 |
| | denoised Q4 volvo 15db | 41.77 |
| | denoised Q4 babble 5db | 42.25 |
| | denoised Q4 babble 10db | 41.34 |
| | denoised Q4 babble 15db | 41.77 |

(c) Denoised training $q = 4$



| Kaldi TRAINING | Kaldi TESTING | PER in % |
|---|---|---|
| all noises mixed denoised with DNN Q=8 | denoised Q8 white 5db | 54.41 |
| | denoised Q8 white 10db | 49.8 |
| | denoised Q8 white 15db | 47.33 |
| | denoised Q8 volvo 5db | 41.01 |
| | denoised Q8 volvo 10db | 40.69 |
| | denoised Q8 volvo 15db | 41.01 |
| | denoised Q8 babble 5db | 41.01 |
| | denoised Q8 babble 10db | 40.69 |
| | denoised Q8 babble 15db | 41.01 |

(d) Denoised training $q = 8$



| Kaldi TRAINING | Kaldi TESTING | PER in % |
|---|---|---|
| all noises mixed denoised with DNN Q=16 | denoised Q16 white 5db | 54.08 |
| | denoised Q16 white 10db | 48.83 |
| | denoised Q16 white 15db | 46.89 |
| | denoised Q16 volvo 5db | 41.51 |
| | denoised Q16 volvo 10db | 40.65 |
| | denoised Q16 volvo 15db | 40.94 |
| | denoised Q16 babble 5db | 41.51 |
| | denoised Q16 babble 10db | 40.65 |
| | denoised Q16 babble 15db | 40.94 |

(e) Denoised training $q = 16$

Figure 6.3: Results in terms of PER (%) for speech recognition

## 6.3 Discussion of the results

### Speech recognition performance

In real conditions, speech recognition is always performed on noisy speech because when hands-free devices are used there is always a certain level of environmental noise. The performance achieved by the system using clean data for training is then considered as the top efficiency that it is possible to reach with this particular system setting. When training the system on data reflecting real outdoor conditions, the PER rises of about 5% when testing clean speech but it drops by around 20% when testing noisy speech. Indeed the performance of the clean-system on noisy speech testing is very very poor.

Our DDAE solution based on VQ seems to reduce the PER of about 2-3% for all kind of noises. If we consider the babble noise at a level of 5dB which challenges quite well the recognition system because it must recognize speech among speech, the performance increases by 2.9%. See Fig.6.4

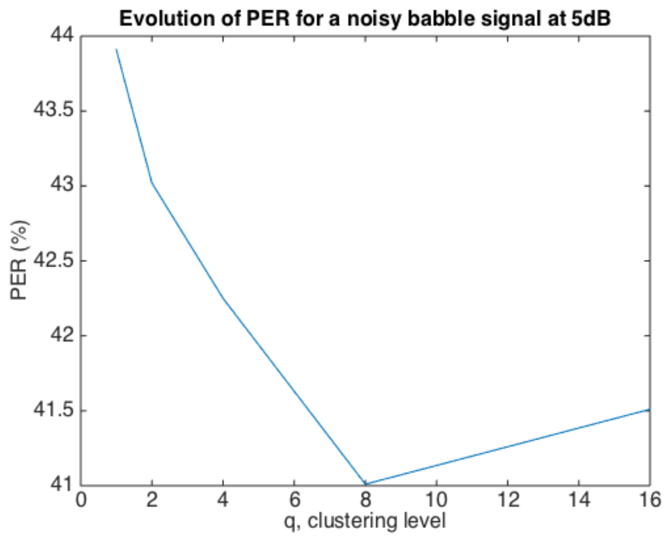


Figure 6.4: Evolution of PER for a noisy babble signal at 5dB

### Efficiency of the DDAE

The DDAE efficiency as shown in the previous section is mitigated. For sure the NN mapping introduces a noise of its own but it also should be able to learn a very efficient denoising mapping. Fig.6.1 shows that the increase of $q$ involves an increase of the SNR. But it must be noticed that the DDAE allows particularly the denoising of the very noisy signals (with a 5dB SNR) but also damages the less noisy signals and particularly the ones with 15dB SNR.

# 7 Conclusion and future work

Issues in the implementation forced us to use an alternative way to initialize the weight matrix of the NN. It follows from that implementation that the denoising system proposed in this thesis is less reliable than the one presented in [5].

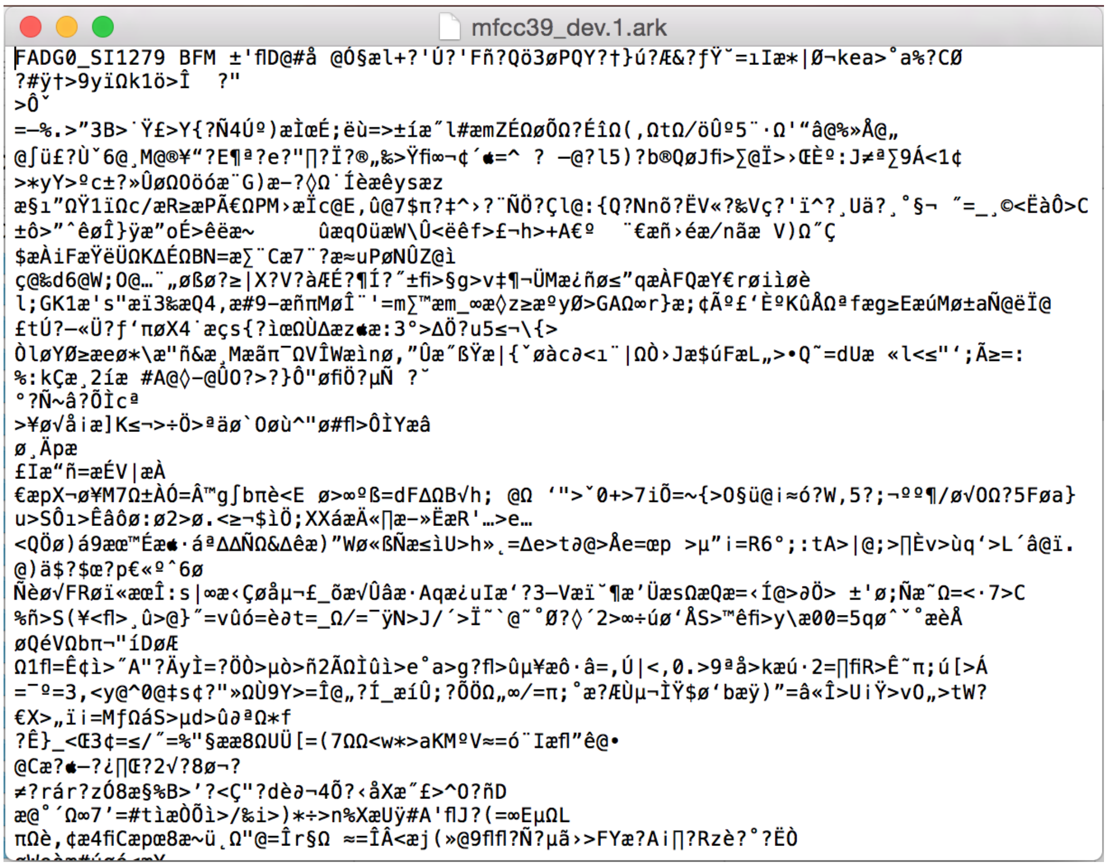
However the input of this work is the use of a clustering algorithm so that the denoising system becomes more fitted to the data. Indeed, the division of data allows the DDAE to learn more particular denoising mappings which are more efficient for producing robust features. We also must point out the time-consuming aspect of our system. Naturally, learning q different DDAEs does, in addition to require a large amount of data, take a lot of time to run.

The results revealed in the previous chapter demonstrate that the novel $\{$MFCC + DDAE$_q$ + GMM-HMM$\}$ allows an 2-3% improvement in the recognition. Nevertheless it might be improved by exploring those leads:

- Explore the overfitting of the system by comparing the performance of the DDAE on its training data and on the testing data.
- Increase the amount of training data for the neural nets. Indeed in this work the amount was limited by the TIMIT corpus but also by the computational cost of learning from such large databases.
- Concerning pre-training, the first step would be to extend the AE pre-training to each layer of the DDAE as it is done with RBM pre-training. A straight forward improvement is also to use the RBM unsupervised learning to pre-train the DDAE.
- Concatenate the denoised MFCC with the standard MFFC to try compensating for the noise introduced by the DDAE. And maybe use the coefficient-to-coefficient trick to reduce the correlation of the output vector.

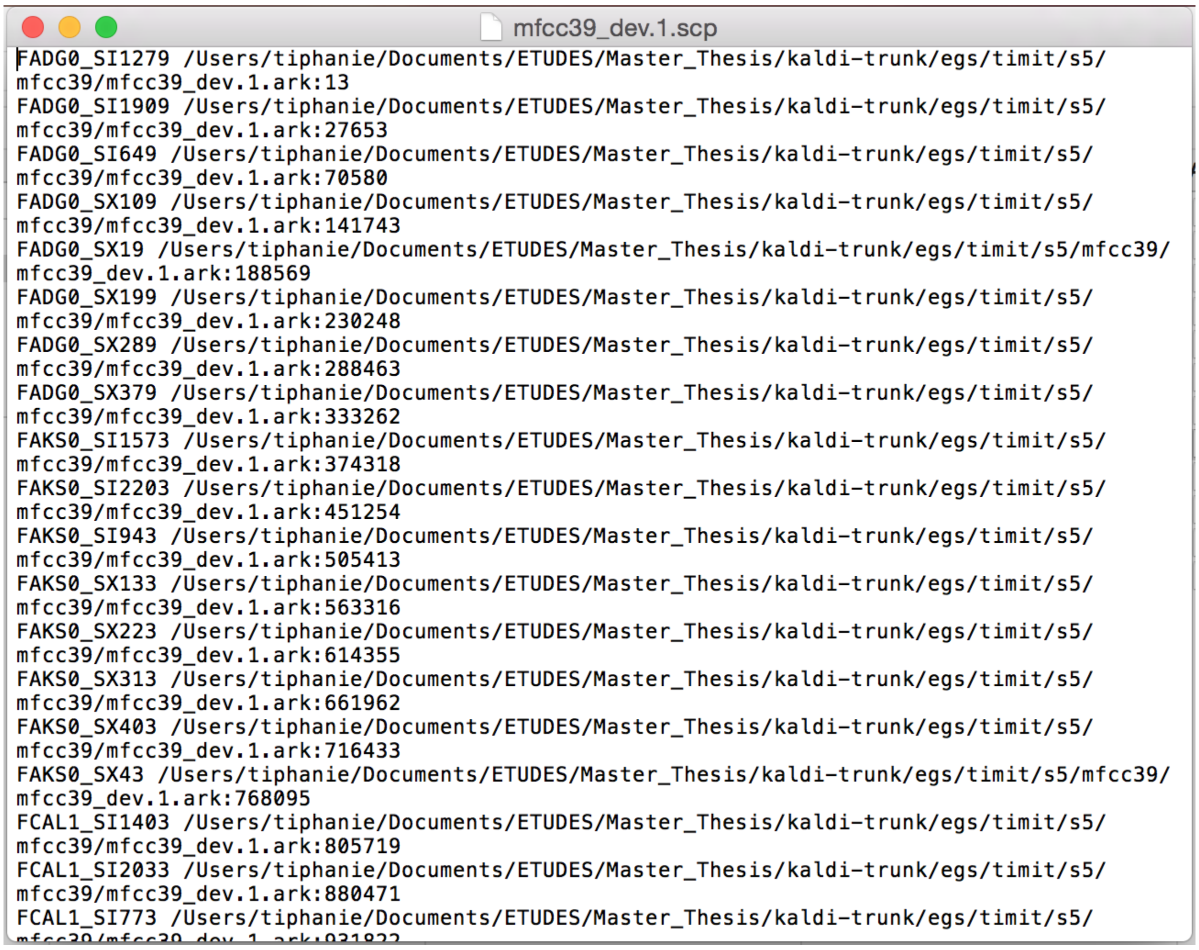
# Appendices

# A Kaldi formats



```
FADG0_SI1279 BFM ±'ﬂD@#å @Ó§æl+?'Ú?'Fñ?Qö3øPQY?†}ú?Æ&?ƒŸ˘=ıIæ∗|Ø¬kea>˚a%?CØ
?#ÿ†>9yïΩk1ö>Î  ?"
>Ô˘
=–%.>”3B>˙Ÿ£>Y{?Ñ4Úº)æÌœÉ;ëù=>±íæ˝l#æmZÉΩøÕΩ?ÉîΩ(,ΩtΩ/öÛº5¨·Ω'“â@%»Å@„
@∫ü£?Ù˘6@¸M@®¥“?E¶ª?e?"∏?Ï?®„‰>Ÿﬁ∞¬¢´=^ ? –@?l5)?b®QøJﬁ>∑@Ï>›ŒÈº:J≠ª∑9Á<1¢
>∗yY>ºc±?»ÛøΩOöóæ¨G)æ–?◊Ω˙Íèæêysæz
æ§ı”ΩŸ1ïΩc/æR≥æPÃ€ΩPM›æÏc@E,û@7$π?‡^›?¨ÑÖ?Çl@:{Q?Nnõ?ËV«?‰Vç?'ï^?¸Uä?¸˚§¬ ˝=_¸©<ËàÔ>C
±ô>”ˆêøÎ}ÿæ”oÉ>êëæ~      ûæqOüæW\Û<ëêf>£¬h>+A€º  ¨€æñ›éæ/nãæ V)Ω˝Ç
$æÀiFæŸëÜΩKΔÉΩBN=æ∑¨Cæ7¨?æ≈uPøNÛZ@ì
ç@‰d6@W;O@…¨„øßø?≥|X?V?àÆÉ?¶Í?˝±ﬁ>§g>v‡¶¬ÜMæ¿ñø≤”qæÀFQæY€røiìøè
l;GK1æ's"æï3‰æQ4,æ#9–æñπMøÎ¨'=m∑™æm_∞æ◊z≥æºyØ>GAΩ∞r}æ;¢Ãº£‘ÈºKûÅΩªfæg≥EæúMø±aÑ@ëÏ@
£tÚ?–«Ü?ƒ‘πøX4˙æçs{?ìœΩÙΔæzæ:3°>ΔÖ?u5≤¬\{>
ÒløYØ≥æeø∗\æ"ñ&æ¸Mæãπ¯ΩVÎWæìnø,”Ûæ˝ßŸæ|{˘øàc∂<ı¨|ΩÒ›Jæ$úFæL„>•Q˜=dUæ «l<≤"‘;Ã≥=:
%:kÇæ¸2íæ #A@◊–@Û0?>?}Ô"øﬁÖ?µÑ ?˘
°?Ñ~â?ÕÌcª
>¥ø√åiæ]K≤¬>÷Ö>ªäø`Oøù^"ø#ﬂ>ÔÌYæâ
ø¸Äpæ
£Iæ“ñ=æÉV|æÀ
€æpX¬ø¥M7Ω±ÀÓ=Â™g∫bπè<E ø>∞ºß=dFΔΩB√h; @Ω ‘">˘0+>7iÕ=~{>O§ü@i≈ó?W,5?;¬ºº¶/ø√OΩ?5Føa}
u>SÔı>Êâôø:ø2>ø.<≥¬$ìÖ;XXáæÄ«∏æ–»ËæR'…>e…
<QÖø)á9æœ™Éæ·áªΔΔÑΩ&Δêæ)”Wø«ßÑæ≤ìU>h»˛=Δe>t∂@>Åe=œp >µ”i=R6°;:tA>|@;>∏Èv>ùq‘>L´â@ï.
@)ä$?$œ?p€«º˘6ø
Ñèø√FRøï«æœÎ:s|∞æ‹Çøåµ¬£_õæ√Ûâæ·Aqæ¿uIæ‘?3–Væï˘¶æ’ÜæsΩæQæ=‹Í@>∂Ö> ±'ø;Ñæ˜Ω=<·7>C
%ñ>S(¥<ﬂ>¸û>@}˝=vûó=è∂t=_Ω/=¯ÿN>J/´>Ï˜`@˜˚Ø?◊´2>∞÷úø‘ÅS>™êﬁ>y\æ00=5qøˆ˘˚æèÅ
øQéVΩbπ¬"íDøÆ
Ω1ﬂ=Ê¢ì>˝A"?ÄyÌ=?ÖÒ>µò>ñ2ÃΩÌûì>e˚a>g?ﬂ>ûµ¥æô·â=,Ú|<,0.>9ªå>kæú·2=∏ﬁR>Ê˜π;ú[>Á
=¯º=3,<y@^0@‡s¢?"»ΩÙ9Y>=Î@„?Í_æíÛ;?ÕÖΩ„∞/=π;˚æ?ÆÙµ¬ÌŸ$ø‘bæÿ)”=â«Î>UiŸ>vO„>tW?
€X>„ïi=MƒΩáS>µd>û∂ªΩ∗f
?Ê}_<Œ3¢=≤/˝=%"§ææ8ΩUÜ[=(7ΩΩ<w∗>aKMºV≈=ó¨Iæﬂ”ê@•
@Cæ?–?¿∏Œ?2√?8ø¬?
≠?rár?zÓ8æ§%B>’?<Ç"?dè∂¬4Õ?‹åXæ˝£>^O?ñD
æ@˚´Ω∞7’=#tìæÒÕì>/‰i>)∗÷>n%XæUÿ#A'ﬂJ?(=∞EµΩL
πΩè,¢æ4ﬁCæpœ8æ~ü¸Ω"@=Îr§Ω ≈=ÎÂ<æj(»@9ﬂﬂ?Ñ?µã››>FYæ?Ai∏?Rzè?˚?ËÒ
```

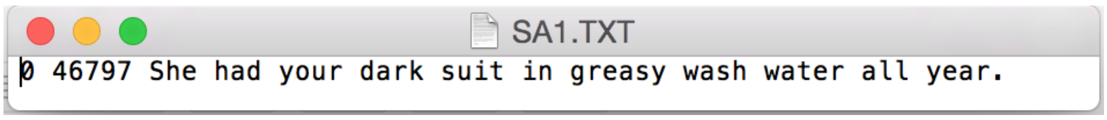
Figure A.1: ark-file produced from Matlab


mfcc39_dev.1.scp

```
FADG0_SI1279 /Users/tiphanie/Documents/ETUDES/Master_Thesis/kaldi-trunk/egs/timit/s5/
mfcc39/mfcc39_dev.1.ark:13
FADG0_SI1909 /Users/tiphanie/Documents/ETUDES/Master_Thesis/kaldi-trunk/egs/timit/s5/
mfcc39/mfcc39_dev.1.ark:27653
FADG0_SI649 /Users/tiphanie/Documents/ETUDES/Master_Thesis/kaldi-trunk/egs/timit/s5/
mfcc39/mfcc39_dev.1.ark:70580
FADG0_SX109 /Users/tiphanie/Documents/ETUDES/Master_Thesis/kaldi-trunk/egs/timit/s5/
mfcc39/mfcc39_dev.1.ark:141743
FADG0_SX19 /Users/tiphanie/Documents/ETUDES/Master_Thesis/kaldi-trunk/egs/timit/s5/mfcc39/
mfcc39_dev.1.ark:188569
FADG0_SX199 /Users/tiphanie/Documents/ETUDES/Master_Thesis/kaldi-trunk/egs/timit/s5/
mfcc39/mfcc39_dev.1.ark:230248
FADG0_SX289 /Users/tiphanie/Documents/ETUDES/Master_Thesis/kaldi-trunk/egs/timit/s5/
mfcc39/mfcc39_dev.1.ark:288463
FADG0_SX379 /Users/tiphanie/Documents/ETUDES/Master_Thesis/kaldi-trunk/egs/timit/s5/
mfcc39/mfcc39_dev.1.ark:333262
FAKS0_SI1573 /Users/tiphanie/Documents/ETUDES/Master_Thesis/kaldi-trunk/egs/timit/s5/
mfcc39/mfcc39_dev.1.ark:374318
FAKS0_SI2203 /Users/tiphanie/Documents/ETUDES/Master_Thesis/kaldi-trunk/egs/timit/s5/
mfcc39/mfcc39_dev.1.ark:451254
FAKS0_SI943 /Users/tiphanie/Documents/ETUDES/Master_Thesis/kaldi-trunk/egs/timit/s5/
mfcc39/mfcc39_dev.1.ark:505413
FAKS0_SX133 /Users/tiphanie/Documents/ETUDES/Master_Thesis/kaldi-trunk/egs/timit/s5/
mfcc39/mfcc39_dev.1.ark:563316
FAKS0_SX223 /Users/tiphanie/Documents/ETUDES/Master_Thesis/kaldi-trunk/egs/timit/s5/
mfcc39/mfcc39_dev.1.ark:614355
FAKS0_SX313 /Users/tiphanie/Documents/ETUDES/Master_Thesis/kaldi-trunk/egs/timit/s5/
mfcc39/mfcc39_dev.1.ark:661962
FAKS0_SX403 /Users/tiphanie/Documents/ETUDES/Master_Thesis/kaldi-trunk/egs/timit/s5/
mfcc39/mfcc39_dev.1.ark:716433
FAKS0_SX43 /Users/tiphanie/Documents/ETUDES/Master_Thesis/kaldi-trunk/egs/timit/s5/mfcc39/
mfcc39_dev.1.ark:768095
FCAL1_SI1403 /Users/tiphanie/Documents/ETUDES/Master_Thesis/kaldi-trunk/egs/timit/s5/
mfcc39/mfcc39_dev.1.ark:805719
FCAL1_SI2033 /Users/tiphanie/Documents/ETUDES/Master_Thesis/kaldi-trunk/egs/timit/s5/
mfcc39/mfcc39_dev.1.ark:880471
FCAL1_SI773 /Users/tiphanie/Documents/ETUDES/Master_Thesis/kaldi-trunk/egs/timit/s5/
```

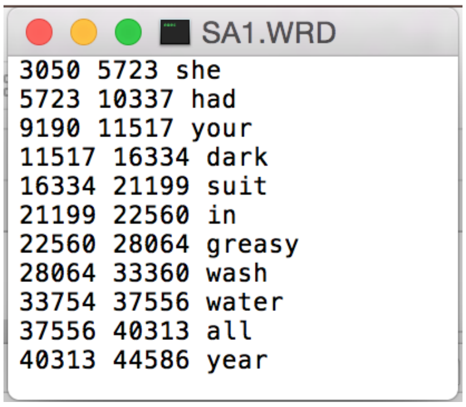
Figure A.2: scp-file produced from Matlab

# B TIMIT files



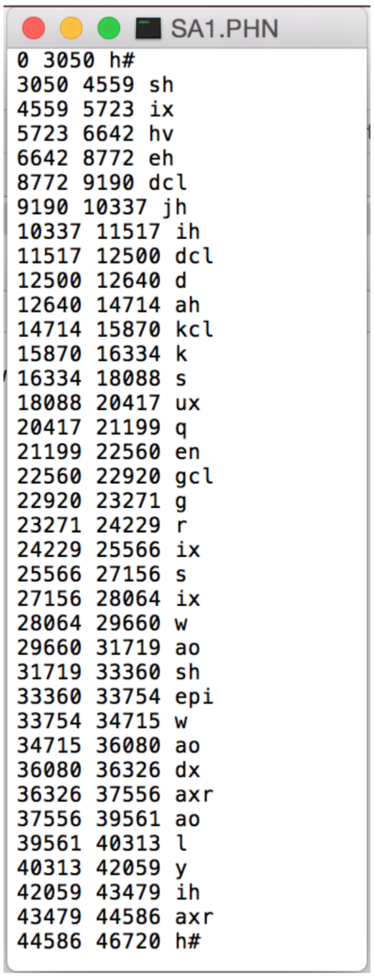
```
SA1.TXT
0 46797 She had your dark suit in greasy wash water all year.
```

Figure B.1: orthographic transcription file *.txt*



```
SA1.WRD
3050 5723 she
5723 10337 had
9190 11517 your
11517 16334 dark
16334 21199 suit
21199 22560 in
22560 28064 greasy
28064 33360 wash
33754 37556 water
37556 40313 all
40313 44586 year
```

Figure B.2: time-aligned word transcription file *.wrd*



```
0 3050 h#
3050 4559 sh
4559 5723 ix
5723 6642 hv
6642 8772 eh
8772 9190 dcl
9190 10337 jh
10337 11517 ih
11517 12500 dcl
12500 12640 d
12640 14714 ah
14714 15870 kcl
15870 16334 k
16334 18088 s
18088 20417 ux
20417 21199 q
21199 22560 en
22560 22920 gcl
22920 23271 g
23271 24229 r
24229 25566 ix
25566 27156 s
27156 28064 ix
28064 29660 w
29660 31719 ao
31719 33360 sh
33360 33754 epi
33754 34715 w
34715 36080 ao
36080 36326 dx
36326 37556 axr
37556 39561 ao
39561 40313 l
40313 42059 y
42059 43479 ih
43479 44586 axr
44586 46720 h#
```

Figure B.3: time-aligned phone transcription file *.phn*

# C Matlab files

## C.1 Database loading

```
function [dataTrain,dataTest, dataDev] = funct_SpeechTimit(pathes,opt)
% Go build a structure of raw speech signals to be used in the remainder of the system
% Inputs :
%           - path_train : path to a txt file that lists all training files
%           - path_test : path to a txt file that lists all testing files
%           - opt.nbtrain : number of speech signals to convert / default :
%           all
%           - opt.nbtest : number of speech signals to convert / default :
%           all
%
% Outputs :
%           - Save the structures in a .mat in the current folder

if nargin<2
    opt.nbtrain = 5000;                              % all speech signals to be converted
    opt.nbtest = 5000;
end

%   Initialization
dataTrain = struct('utt',[],'rawSpeech',[],'frames',[],'mfcc',[],'part1',[],'ourFeature',[]);
dataTest = struct('utt',[],'rawSpeech',[],'frames',[],'mfcc',[],'part1',[],'ourFeature',[]);
dataDev = struct('utt',[],'rawSpeech',[],'frames',[],'mfcc',[],'part1',[],'ourFeature',[]);

fidTrain = fopen(pathes{1});
fidTest = fopen(pathes{2});
fidDev = fopen(pathes{3});

tlineTest = fgetl(fidTest);
tlineTrain = fgetl(fidTrain);
tlineDev = fgetl(fidDev);

%%   TRAINING : Framing
i = 0;                                               % counter
while ischar(tlineTrain) && i < opt.nbtrain
    i = i + 1;

    % Name of utterance (for Kaldi tool)
    C = strsplit(tlineTrain,{'/','.'}) ;
%     dataTrain.utt{i} = [C{9} '_' C{10} '_' C{11} '_' C{12}] ;

    % Name with train_drX is blocking in Kaldi
    % Trial without it:
    dataTrain.utt{i} = [C{11} '_' C{12}] ;
    % Read Timit
    y =readsph(tlineTrain,'s',-1);

    % Add to structure
    dataTrain.rawSpeech{i} = y' ;                     % raw speech in row vector

    % Update line
    tlineTrain = fgetl(fidTrain);
end


%%   TESTING : Framing
i = 0;                                               % counter
while ischar(tlineTest) && i < opt.nbtest
    i = i + 1;
    % Name of utterance (for Kaldi tool)
    C = strsplit(tlineTest,{'/','.'}) ;
%     dataTest.utt{i} = [C{9} '_' C{10} '_' C{11} '_' C{12}] ;

    % Name with train_drX is blocking in Kaldi
    % Trial without it:
    dataTest.utt{i} = [C{11} '_' C{12}] ;

    % Read Timit
    y =readsph(tlineTest,'s',-1);

    % Add to structure
    dataTest.rawSpeech{i} = y' ;                     % Raw speech in row vector

    % Update line
    tlineTest = fgetl(fidTest);
end

%%   DEV : Framing
i = 0;                                               % counter
while ischar(tlineDev)
    i = i + 1;

    % Name of utterance (for Kaldi tool)
    C = strsplit(tlineDev,{'/','.'}) ;
%     dataDev.utt{i} = [C{9} '_' C{10} '_' C{11} '_' C{12}] ;

    % Name with train_drX is blocking in Kaldi
    % Trial without it:
    dataDev.utt{i} = [C{11} '_' C{12}] ;

    % Read Timit
    y =readsph(tlineDev,'s',-1);

    % Add to structure
    dataDev.rawSpeech{i} = y' ;                     % raw speech in row vector

    % Update line
    tlineDev = fgetl(fidDev);
end


%% SAVING DATA
save('dataTrain.mat','dataTrain','-v7.3') ;
save('dataTest.mat','dataTest','-v7.3') ;
save('dataDev.mat','dataDev','-v7.3') ;
```

```
fclose(fidTrain) ;
fclose(fidTest) ;
fclose(fidDev) ;

end                                                   % End function
```

## C.2 MFCC implementation

### C.2.1 MFCC

```
function [mfccout] = funct_GetMfcc(frames, FB, HamWin, dct_matrix)
% Calculates the MFCCs of speech frames with M triangular filters stored in
% FB
% Inputs :
%           - frames : speech frames (from normalized signal)
%           - M : number of triangular filters
%           - FB : triangular filterbanks
% Outputs :
%           - mfcc : stored MFCC of all the speech frames


% Energy of the speech frames
E = 10*log(sum((frames .* frames)'));


% Hamming windowed Power spectrum calculation
PS = power_spectrum(frames,HamWin);


% Initial filter banks (FB) using intial parameters
% Given as inputs : FB


% Initial filter bank energies (FBE) using initial filter bank

FBE_ini = PS * FB' ;


% Intial compressed filter bank energies (Com_FBE) using polynomial
% function

Com_FBE_ini = log10(FBE_ini);


% Initial feature vector (MFCC) evaluation

FCC_ini = dct_matrix * Com_FBE_ini';

% Add log-energy as 13th coefficient
mfccout = FCC_ini';

end
```

### C.2.2 MFCC + deltas + deltas-deltas

```
function [mfccout] = funct_GetMfccComplete(frames, FB, HamWin, dct_matrix)
% Calculates the MFCCs of speech frames with M triangular filters stored in
% FB
% Inputs :
%           - frames : speech frames (from normalized signal)
%           - M : number of triangular filters
%           - FB : triangular filterbanks
% Outputs :
%           - mfcc : stored MFCC of all the speech frames


% Energy of the speech frames
E = 10*log(sum((frames .* frames),2));


% Hamming windowed Power spectrum calculation
PS = power_spectrum(frames,HamWin);


% Initial filter banks (FB) using intial parameters
% Given as inputs : FB


% Initial filter bank energies (FBE) using initial filter bank

FBE_ini= PS * FB' ;


% Intial compressed filter bank energies (Com_FBE) using polynomial
% function

Com_FBE_ini=log10(FBE_ini);


% Initial feature vector (MFCC) evaluation

FCC_ini= dct_matrix * Com_FBE_ini';

% Add log-energy as 13th coefficient
mfcc=[FCC_ini; E'];

% Add deltas
d = deltas(mfcc,5);
dd = deltas(d,5);
mfccout = [mfcc;d;dd];
mfccout = mfccout';
end
```

### C.2.3 FilterBanks

```
function [FB, startFreq, centreFreq, endingFreq] = funct_Filterbanks(M,Fs,frameLengthSamples)
% Calculates the frequencies of the filterbanks according to the frame's
% lengths
% Inputs :
%           - M : Number of triangular filters
%           - Fs : Sampling frequency of speech signals
%           - frameLengthSamples : Frame length in samples
% Outputs :
%           - startFreq : start frequency index
%           - centreFreq : centre frequency index
%           - endFreq : terminating frequency index
%           - FB : triangular filterbanks
% Global parameters
NyquistFreq = Fs/2;                                 % Half of sampling
%rate of a discrete signal /!\ different from Nyquist rate!!
DFTLength = frameLengthSamples;                     % We use DFT length
% = length of frame in samples

% ----------------------------------------------------------------------------
%%%%%%%%%%%%%%%%%%%%%%% Calculates Filterbanks indexes %%%%%%%%%%%%%%%%%%%%%%%
% ----------------------------------------------------------------------------

% Mel scale parameters
% Dft signal's symmetry involves that nyquist frequency is the max one we have
maxMelFreq = 2595 * log10(1+NyquistFreq/700);
% Delta is half the width of the filters in mel scale
del = maxMelFreq/(M+1);

% Initialization
%   MEL domain
omegaCentreFreq_Mel = zeros(1,M);
omegaStartFreq_Mel = zeros(1,M);
omegaEndingFreq_Mel = zeros(1,M);
%   Normal Freq domain
OmegaCentreFreq = zeros(1,M);
OmegaStartFreq = zeros(1,M);
OmegaEndingFreq = zeros(1,M);
centreFreq = zeros(1,M);
startFreq = zeros(1,M);
endingFreq = zeros(1,M);

% Calculation of Center-, Start- and End- frequencies in Mel Domain
for i=1:M
    if i==1 % Case first filter
        omegaCentreFreq_Mel(i) = del;
        omegaStartFreq_Mel(i) = 0;
        omegaEndingFreq_Mel(i) = omegaCentreFreq_Mel(i)+del;
    else % Case all other ones
        omegaCentreFreq_Mel(i) = omegaCentreFreq_Mel(i-1)+del;
        omegaStartFreq_Mel(i) = omegaCentreFreq_Mel(i-1);
        omegaEndingFreq_Mel(i) = omegaCentreFreq_Mel(i)+del;
    end

    % In case the last ending freq exceeds maxEnding one
    if (omegaEndingFreq_Mel(M) > maxMelFreq)
        OmegaEndingFreq(M) = maxMelFreq;
    end

end

% Coming back to Normal Freq Domain
for i=1:M
    OmegaCentreFreq(i) = 700 * (10^(omegaCentreFreq_Mel(i)/2595) - 1);
    OmegaStartFreq(i) = 700 * (10^(omegaStartFreq_Mel(i)/2595) - 1);
    OmegaEndingFreq(i) = 700 * (10^(omegaEndingFreq_Mel(i)/2595) - 1);
    % Case of ending freq exceeds nyquist freq
    if (OmegaEndingFreq(M) > NyquistFreq)
        OmegaEndingFreq(M) = NyquistFreq;
    end
end

% Frequencies on the DFT scale = Samples/ Indexes
centreFreq = round(OmegaCentreFreq * (DFTLength/Fs));
startFreq = round(OmegaStartFreq * (DFTLength/Fs));
```

```
endingFreq = round(OmegaEndingFreq * (DFTLength/Fs));


% Ensure that first filter starts at 0
startFreq(1)=0;
% Ensure that last filter ends at dft/2 = nyquist freq
endingFreq(M)=DFTLength/2;


% ----------------------------------------------------------------
%%%%%%%%%%%%%%%%%%%%%%% Calculates Triangular Filterbanks %%%%%%%%
% ----------------------------------------------------------------

% Initialization
triangleWindow=zeros(M,DFTLength/2+1);

% Calculation of the individual filters in normal freq domain
for i=1:M
    % temporary window vector to do calculations
    dummyWindow=zeros(1,DFTLength/2+1);
    for k=0:DFTLength/2
        % Calculation of filter values
        if ( (startFreq(i) <= k) && (k <= centreFreq(i)))
            dummyWindow(k+1)= 2 * (endingFreq(i)*centreFreq(i)...
                - startFreq(i)*centreFreq(i) - endingFreq(i)*startFreq(i)...
                 + startFreq(i)^2 )^(-1) * (k-startFreq(i));
        elseif ( (centreFreq(i) < k) && (k <= endingFreq(i)))
            dummyWindow(k+1)= 2 * (startFreq(i)*centreFreq(i)...
                - startFreq(i)*endingFreq(i) - centreFreq(i)*endingFreq(i)...
                 + endingFreq(i)^2 )^(-1) * (centreFreq(i)-k) ...
                + 2 * (endingFreq(i)-startFreq(i))^(-1);
        end
    end
    triangleWindow(i,:)=dummyWindow;                % Each row stores a filter
end

FB=triangleWindow;

end
```

### C.2.4 Power spectrum

```
% This file returns the power spectrum of the speech signal matrix

function PS_M=power_spectrum(s_M,HamWin)

% PS_M <- Power spectrum matrix  (row index is the speech frame index)

% s_M ->  speech signal matrix
% HamWin -> Hamming window

[row col]=size(s_M);

PS_M=zeros(row,(length(HamWin)/2)+1);


for j=1:row

    sig_w=s_M(j,:) .* HamWin;                       % For original signal
    sig_fft=fft(sig_w);                             % Fourier tranform
    sxx_sig= (abs(sig_fft)).^2;                     % Power spectrum
    sxx_sig_part=sxx_sig(1:length(sxx_sig)/2 + 1);       % Half is sufficient (symmetry)

    PS_M(j,:)=sxx_sig_part;                              % Return

end
```

## C.3 RBM learning

### C.3.1 Training using [22]

```
% Using another toolbox for RBMs : DeepNeuralNetwork
% Get random RBM
dims = [inputSize bottleneckSize inputSize] ;
dbn = randDBN(dims, 'GBDBN' ) ; % dbn = (number of nodes, type of RBM)
nrbm = numel(dbn.rbm);

opts.MaxIter = 100;
opts.BatchSize = 20;
opts.Verbose = true;
opts.StepRatio = 0.1;
opts.Layer = nrbm-1;
opts.Object = 'CrossEntropy';

dbn = pretrainDBN(dbn, train_input, opts);
dbn= SetLinearMapping(dbn, train_input, train_input);
opts.Layer = 0;
dbn = trainDBN(dbn, train_input, train_input, opts);

out = v2h( dbn, train_input );
snr = SNR_calc(train_input,out);
```

## C.4 DDAE implementation

### C.4.1 Noisy signals

```
function [frames] = funct_SpeechFramingNoisy(s, frameLengthSamples, frameShiftSamples,noise,SNR)
% Normalizes and divides the speech signal into frames
% Inputs :
%            - s : speech signal
%            - frameLengthSamples : Frame length in samples
%            - frameShiftSamples : Frame shift in samples
%            - noise : noise signal
%            - SNR : SNR of the wanted noisy signal
% Outputs :
%            - frames

% Initialization
n = 0;                                              % Frame counter
start = 1;                                          % Initialization boundary of 1st frame
finish = frameLengthSamples;                        % Initialization boundary of 1st frame
frames = zeros(floor(length(s)/frameLengthSamples),frameLengthSamples);

% Normalization
s = s-mean(s);
s = s / max(abs(s));
```

```
noise=noise(1:length(s));
noise=noise/std(noise);
noise_var=(var(s) / (10^(SNR/10)));
noise_sample= (noise_var^(0.5)) * noise;

s_noisy=s+noise_sample;


% Framing
while (finish < (length(s)- frameLengthSamples))

    n = n + 1;                                      % Frame indexation

    frames(n,:) = s_noisy(start:finish);                  % Frame from signal

    start = start + frameShiftSamples;          % update boundaries for next frame
    finish = finish + frameShiftSamples;
end

end
```

### C.4.2 Training of the NN

```
function [dbn] = funct_TrainDBNcj(inputCell, HamWin, FB, opts, myOpt)
% Train the Auto-encoder: Given speech frames, it gets the power spectrum
% an used this as input and output for AE.
% Inputs :
%           - inputCell : cell containing all input representations for the
%           DBN
%
% Outputs :
%           - dbn: all DBN parameters saved in a structure


 % Number of speech signals for training AE
if myOpt.NSig==0
    NSig = 50;
else
    NSig = myOpt.NSig;
end

%% DATA

[train_S_norm, target_S] = funct_ProcessDataForDbn(inputCell,NSig, HamWin, FB, myOpt);

%% Train RBM

% Approx 320/optAE.inputsize RBM to train ("frequency separation" for AE to
% work)


% Options defined in opts (input to the function)


switch myOpt.in


    case 'PS'
        % In this case each DBN has a different structure/dimensions
        % according to the FB
        NbDBN = size(myOpt.mat,1);
        for h=1:1:NbDBN
            train_input = train_S_norm{h};
            train_target = target_S{h};
            dims = [size(train_input,2) size(train_input,2) size(train_target,2)];
            dbn_ini = randDBN(dims, myOpt.typeRBM );
            nrbm = numel(dbn_ini.rbm);
            opts.Layer = nrbm-1;
            dbn{h} = dbn_ini;
            dbn{h} = pretrainDBN(dbn{h}, train_input, opts);
            dbn{h}= SetLinearMapping(dbn{h}, train_input, train_target);
            opts.Layer = 0;
            dbn{h} = trainDBN(dbn{h}, train_input, train_target, opts);

            disp([num2str(h) '/' num2str(NbDBN)])
        end

    case {'PSFB','PSFBlog'}
        % In this case each DBN has a different structure/dimensions
        NbDBN = size(myOpt.mat,1);
        for h=1:1:NbDBN
            train_input = train_S_norm{h};
            train_target = target_S(:,h);
            dims = [size(train_input,2) myOpt.bottleneckSize size(train_target,2)];
            dbn_ini = randDBN(dims, myOpt.typeRBM );
            nrbm = numel(dbn_ini.rbm);
            opts.Layer = nrbm-1;
            dbn{h} = dbn_ini;
            dbn{h} = pretrainDBN(dbn{h}, train_input, opts);
            dbn{h}= SetLinearMapping(dbn{h}, train_input, train_target);
            opts.Layer = 0;
            dbn{h} = trainDBN(dbn{h}, train_input, train_target, opts);

            disp([num2str(h) '/' num2str(NbDBN)])
        end

    case 'DctLog'
        % Only one DBN for this case : from M filters to Q coefs

        train_input = train_S_norm;
        train_target = target_S;
        dims = [size(train_input,2) myOpt.bottleneckSize size(train_target,2)];
        dbn_ini = randDBN(dims, myOpt.typeRBM );
        nrbm = numel(dbn_ini.rbm);
        opts.Layer = nrbm-1;
        dbn = pretrainDBN(dbn_ini, train_input, opts);
        % Parameters change for training
        opts.MaxIter = 50;
        opts.StepRatio = 0.00004;
        % Training
        dbn = SetLinearMapping(dbn, train_input, train_target,opts);
        opts.Layer = 0;
        dbn = trainDBN(dbn, train_input, train_target, opts);

    case 'Denoise'

        rand('state',0)
        sae = saesetup([size(train_S_norm,2) myOpt.bottleneckSize(1)]);
        nae = size(sae.ae,2);
        sae.ae{1}.activation_function       = 'sigm';
        sae.ae{1}.learningRate              = 0.5;
        sae.ae{1}.inputZeroMaskedFraction   = 0.5;
        sae.ae{1}.output                      = 'sigm';
        opts.numepochs =   25;
        opts.batchsize = 250;
        opts.momentum = 0.5;
        sae = saetrain(sae, train_S_norm, opts);

        % Use the SDAE to initialize a FFNN
        nn = nnsetup([size(train_S_norm,2) myOpt.bottleneckSize size(target_S,2)]);
        nn.activation_function                = 'sigm';
        nn.output                             = 'linear';
        nn.learningRate                       = 0.004;
        nn.momentum                           = 0.5;
        nn.W{1} = sae.ae{1}.W{1};

        % Train the FFNN
        opts.numepochs =  50;
        opts.batchsize = 250;
        nn = nntrain(nn, train_S_norm, target_S, opts);
        [er, bad] = nntest(nn, train_S_norm, target_S);
        disp(['er training:' num2str(er)])
        dbn = nn ;

    case 'DenoiseQuant'
        % DBN for each cluster of Noisy Mfcc
        for i=1:myOpt.quant
            train_input = train_S_norm{i};
            train_target = target_S{i};
            dims = [size(train_input,2) myOpt.bottleneckSize size(train_target,2)];

            rand('state',0)
            sae = saesetup([size(train_input,2) myOpt.bottleneckSize(1)]);
            nae = size(sae.ae,2);
            sae.ae{1}.activation_function       = 'sigm';
            sae.ae{1}.learningRate              = 0.5;
            sae.ae{1}.inputZeroMaskedFraction   = 0.5;
            sae.ae{1}.output                    = 'sigm';
            opts.numepochs =   25;
            opts.batchsize = 250;
            opts.momentum = 0.5;
            sae = saetrain(sae, train_input, opts);

            % Use the SDAE to initialize a FFNN
            nn = nnsetup([size(train_input,2) myOpt.bottleneckSize size(train_target,2)]);
            nn.activation_function              = 'sigm';
            nn.output                           = 'linear';
            nn.learningRate                     = 0.004;
            nn.momentum                         = 0.5;
            nn.W{1} = sae.ae{1}.W{1};

            % Train the FFNN
            opts.numepochs =  50;
            opts.batchsize = 250;
            nn = nntrain(nn, train_input, train_target, opts);
            [er, bad] = nntest(nn, train_input, train_target);
            disp(['er training:' num2str(er)])

            dbn{i} = nn ;

        end

end
```

```
disp('==>> funct_TrainDBNcj succeeded');
```

### C.4.3 Format conversion

```
function [] = funct_WriteKaldiFormat(utt,feats,outputFolder,filename,N)
% Writes the features form Matlab structure to Kaldi format.
% Generates .ark and .scp files in indicated outputfolder;
%
% Inputs :
%           - utt : names of the speech signals from data.utt
%           - feats : features structure from data.feature
%           - outputFolder : Folder in which to put generated files
%           - filenames : string of the name for .ark file
%           - N : nb of utterance
% Outputs : NONE
%

% Create Headers for ark writing
HEADER_MAT_TRAIN = cell(N,5);
HEADER_MAT_TRAIN(1,:) = {utt(1), size(feats{1},1),size(feats{1},2),1,size(feats{1},1)};
FEATURE_MAT_TRAIN = [];
FEATURE_MAT_TRAIN = [FEATURE_MAT_TRAIN; feats{1}];
for i=2:N
    HEADER_MAT_TRAIN(i,:) = {utt(i), size(feats{i},1) , size(feats{1},2),...
        HEADER_MAT_TRAIN{i-1,5}+1, HEADER_MAT_TRAIN{i-1,5} + size(feats{i},1)};
    FEATURE_MAT_TRAIN = [FEATURE_MAT_TRAIN; feats{i}];
end
disp('HEADER_MAT_TRAIN and FEATURE_MAT_TRAIN computed');

% Create Mfcc directory
command = ['mkdir ' outputFolder];
[~,~]=system(command);

% Write for Kaldi
statArk = arkwrite(filename, HEADER_MAT_TRAIN, FEATURE_MAT_TRAIN);
% BE CAREFUL NAME OF FILES : .1. = 1 split

disp(['Status ark-writing:' num2str(statArk)]);

statScp = ark2scp(filename);

disp(['Status scp-writing:' num2str(statScp)]);
end
```

### C.4.4 Kaldi script

#### Training an acoustic model

```
## DATA PREPARATION - TIMIT
timit=/Users/tiphanie/Documents/ETUDES/Master_Thesis/Databases/TIMIT ;

local/timit_data_prep.sh $timit || exit 1

local/timit_prepare_dict.sh

# Caution below: we remove optional silence by setting "--sil-prob 0.0",
# in TIMIT the silence appears also as a word in the dictionary and is scored.
utils/prepare_lang.sh --sil-prob 0.0 --position-dependent-phones false --num-sil-states 3 \
data/local/dict "sil" data/local/lang_tmp data/lang

local/timit_format_data.sh
#########################

## MONOPHONE TRAINING
name='2mfcc'
cat "$name"/"$name"_train.1.scp > data/feats.scp
cp "$name"/"$name"_train.1.scp data/train/feats.scp

mkdir -p exp/make_$name
mkdir -p exp/mono-$name

steps/compute_cmvn_stats.sh data/train exp/make_$name/train ./$name

data=data/train
lang=data/lang
dir=exp/mono-$name
stage=-4
echo "$0: Initializing monophone system."
utils/split_data.sh data/train $train_nj
../../../src/featbin/apply-cmvn --utt2spk=ark:data/train/split1/1/utt2spk\
 scp:data/train/split1/1/cmvn.scp scp:data/train/split1/1/feats.scp ark:-\n
../../../src/featbin/feat-to-dim scp:data/train/split1/1/feats.scp -\n
../../../src/gmmbin/gmm-init-mono --shared-phones=data/lang/phones/sets.int\
 data/lang/topo 39 exp/mono-$name/0.mdl exp/mono=$name/tree

echo "$0: Compiling training graphs"
max_iter_inc=30 # Last iter to increase #Gauss on.
totgauss=1000 # Target #Gaussians.
dir=exp/mono-$name
numgauss=`../../../src/gmmbin/gmm-info --print-args=false $dir/0.mdl\
 | grep gaussians | awk '{print $NF}'`
incgauss=$[($totgauss-$numgauss)/$max_iter_inc] # per-iter increment for #Gauss

cmd=utils/run.pl
sdata=$data/split$train_nj;
oov_sym=`cat $lang/oov.int`

$cmd JOB=1:$train_nj $dir/log/compile_graphs.JOB.log ../../../src/bin/compile-train-graphs\
 $dir/tree $dir/0.mdl  $lang/L.fst \
"ark:utils/sym2int.pl --map-oov $oov_sym -f 2- $lang/words.txt < $sdata/JOB/text|" \
"ark:|gzip -c >$dir/fsts.JOB.gz"

echo "$0: Aligning data equally (pass 0)"
$cmd JOB=1:$train_nj $dir/log/align.0.JOB.log \
../../../src/bin/align-equal-compiled "ark:gunzip -c $dir/fsts.JOB.gz|"\
 scp:data/train/split1/1/feats.scp ark,t:-  \| ../../../src/gmmbin/gmm-acc-stats-ali \
 --binary=true $dir/0.mdl scp:data/train/split1/1/feats.scp ark:- \
$dir/0.JOB.acc

power=0.25 # exponent to determine number of gaussians from occurrence counts

../../../src/gmmbin/gmm-est --min-gaussian-occupancy=3  --mix-up=$numgauss\
 --power=$power $dir/0.mdl "../../../src/gmmbin/gmm-sum-accs - $dir/0.*.acc|"\
  $dir/1.mdl 2> $dir/log/update.0.log
rm $dir/0.*.acc

echo "-------------FIRST STAGES DONE-----------------"

# Training
num_iters=40    # Number of iterations of training
```

```
beam=6 # will change to 10 below after 1st pass
realign_iters="1 2 3 4 5 6 7 8 9 10 12 14 16 18 20 23 26 29 32 35 38";
boost_silence=1.0 # Factor by which to boost silence likelihoods in alignment
careful=false
feats=scp:data/train/split1/1/feats.scp
# note: using slightly wider beams for WSJ vs. RM.
x=1
while [ $x -lt $num_iters ]; do
echo "$0: Pass $x"
if [ $stage -le $x ]; then
if echo $realign_iters | grep -w $x >/dev/null; then
echo "$0: Aligning data"
mdl="../../../src/gmmbin/gmm-boost-silence --boost=$boost_silence \
`cat $lang/phones/optional_silence.csl`\ $dir/$x.mdl - |"
$cmd JOB=1:$train_nj $dir/log/align.$x.JOB.log \
../../../src/gmmbin/gmm-align-compiled $scale_opts --beam=$beam\
 --retry-beam=$[$beam*4] --careful=$careful "$mdl" "ark:gunzip -c\
  $dir/fsts.JOB.gz|" "$feats" "ark,t:|gzip -c >$dir/ali.JOB.gz"
fi
$cmd JOB=1:$train_nj $dir/log/acc.$x.JOB.log \
../../../src/gmmbin/gmm-acc-stats-ali  $dir/$x.mdl "$feats" "ark:gunzip\
 -c $dir/ali.JOB.gz|" $dir/$x.JOB.acc

$cmd $dir/log/update.$x.log \
../../../src/gmmbin/gmm-est --write-occs=$dir/$[$x+1].occs\
 --mix-up=$numgauss --power=$power $dir/$x.mdl \
"../../../src/gmmbin/gmm-sum-accs - $dir/$x.*.acc|" $dir/$[$x+1].mdl
rm $dir/$x.mdl $dir/$x.*.acc $dir/$x.occs 2>/dev/null
fi
if [ $x -le $max_iter_inc ]; then
numgauss=$[$numgauss+$incgauss];
fi
beam=10
x=$[$x+1]
done

( cd $dir; rm final.{mdl,occs} 2>/dev/null; ln -s $x.mdl final.mdl; ln -s $x.occs final.occs )

utils/summarize_warnings.pl $dir/log

# Graphs
utils/mkgraph.sh --mono data/lang_test_bg exp/mono-$name exp/mono-$name/graph

echo Done
```

### Decoding for a different sort of features from the training

E.g. you have a model trained on clean features and you want to test it on noisy features.

```
graphdir='mono-mfcc39'

name='mfcc39babble5db'
dir="mono-clean-$name"

cat "$name"/"$name"_test.1.scp > data/feats.scp

cp "$name"/"$name"_test.1.scp data/test/feats.scp

mkdir -p exp/make_$name

steps/compute_cmvn_stats.sh data/test exp/make_$name/test ./$name
```

```
# Create mono directory
mkdir ./exp/$dir

# Copy final.mdl from training to testing
cp exp/$graphdir/final.mdl exp/$dir/final.mdl

steps/decode_tiphanie.sh --nj "$decode_nj" --cmd "run.pl" --skip-scoring true\
 exp/$graphdir/graph data/test exp/$dir/decode_test

local/score_basic.sh data/test/ exp/$graphdir/graph/ exp/$dir/decode_test/
```

# Bibliography

[1] Yoshua Bengio. Learning deep architectures for ai. *Foundations and Trends in Machine Learning*, 2, 2009.

[2] Saikat Chatterjee and W. Bastiaan Kleijn. Auditory model-based design and optimization of feature vectors for automatic speech recognition. *IEEE Transactions on Audio, Speech and Language Processing*, 2011.

[3] Li Deng and Dong Yu. Deep learning: Methods and applications. *Foundations and Trends® in Signal Processing*, 7(3–4):197–387, 2013.

[4] Marcello Federico and Nicola Bertoldi. Irstlm toolkit. *http://hlt-mt.fbk.eu/technologies/irstlm*, 2013.

[5] Xue Feng, Yaodong Zhang, and J. Glass. Speech feature denoising and dereverberation via deep autoencoders for noisy reverberant speech recognition. In *Acoustics, Speech and Signal Processing (ICASSP), 2014 IEEE International Conference on*, pages 1759–1763, May 2014.

[6] Jonathan Fiscus. Sctk version: 2.4.10. *http://www1.icsi.berkeley.edu/Speech/docs/sctk-1.2/sclite.htm*, 2015.

[7] Mark Gales and Steve Young. The application of hidden markov models in speech recognition. *Foundations and Trends® in Signal Processing*, 1(3):195–304, 2007.

[8] J. Gehring, Y. Miao, F. Metze, and A. Waibel. Acoustics, speech and signal processing (icassp), 2013 ieee international conference on. In *Extracting deep bottleneck features using stacked auto-encoders*, pages 3377–3381, May 2013.

[9] G. Hinton, L. Deng, D. Yu, G. E. Dahl, A. r. Mohamed, N. Jaitly, A. Senior, V. Vanhoucke, P. Nguyen, T. N. Sainath, and B. Kingsbury. Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups. *IEEE Signal Processing Magazine*, 29(6):82–97, Nov 2012.

[10] G. E. Hinton. Training products of experts by minimizing contrastive divergence. *Neural Computation*, 2002.

[11] Geoffrey E. Hinton. A practical guide to training restricted boltzmann machines. In Grégoire Montavon, Genevieve B. Orr, and Klaus-Robert Müller, editors, *Neural Networks: Tricks of the Trade (2nd ed.)*. Springer, 2012.

[12] Geoffrey E. Hinton, Nitish Srivastava, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Improving neural networks by preventing co-adaptation of feature detectors. *CoRR*, abs/1207.0580, 2012.

[13] Navdeep Jaitly and Geoffrey E. Hinton. Learning a better representation of speech soundwaves using restricted boltzmann machines. In *ICASSP*. IEEE, 2011.

[14] Navdeep Jaitly and Geoffrey E. Hinton. Using an autoencoder with deformable templates to discover features for automated speech recognition. In *INTERSPEECH-2013*, 2013.

[15] A. Mohamed, G. Dahl, and G. Hinton. Deep belief networks for phone recognition. *NIPS 22 workshop on deep learning for speech recognition*, 2009.

[16] R. B. Palm. Prediction as a candidate for learning deep hierarchical models of data. Master's thesis, Technical University of Denmark, DTU Informatics, E-mail: reception@imm.dtu.dk, Asmussens Alle, Building 305, DK-2800 Kgs. Lyngby, Denmark, 2012.

[17] Daniel Povey, Arnab Ghoshal, Gilles Boulianne, Lukas Burget, Ondrej Glembek, Nagendra Goel, Mirko Hannemann, Petr Motlicek, Yanmin Qian, Petr Schwarz, Jan Silovsky, Georg Stemmer, and Karel Vesely. The kaldi speech recognition toolkit. In *IEEE 2011 Workshop on Automatic Speech Recognition and Understanding*, Hilton Waikoloa Village, Big Island, Hawaii, US, dec 2011. IEEE Signal Processing Society.

[18] Yu Qi, Yueming Wang, Xiaoxiang Zheng, and Zhaohui Wu. Robust feature learning by stacked autoencoder with maximum correntropy criterion. In *Acoustics, Speech and Signal Processing (ICASSP), 2014 IEEE International Conference on*, May 2014.

[19] A. r. Mohamed, T. N. Sainath, G. Dahl, B. Ramabhadran, G. E. Hinton, and M. A. Picheny. Deep belief networks using discriminative features for phone recognition. In *Acoustics, Speech and Signal Processing (ICASSP), 2011 IEEE International Conference on*, pages 5060–5063, May 2011.

[20] Tara N. Sainath, Ron J. Weiss, Andrew Senior, Kevin W. Wilson, and Oriol Vinyals. Learning the speech front-end with raw waveform cldnns. In *INTERSPEECH-2015*, 2015.

[21] M. L. Seltzer, D. Yu, and Y. Wang. An investigation of deep neural networks for noise robust speech recognition. In *Acoustics, Speech and Signal Processing (ICASSP), 2013 IEEE International Conference on*, pages 7398–7402, May 2013.

[22] M. Tanaka and M. Okutomi. A novel inference of a restricted boltzmann machine. In *Pattern Recognition (ICPR), 2014 22nd International Conference on*, 2014.

[23] Emmanuel Vincent and Shinji Watanabe. Kaldi to matlab conversion tools. *http://kaldi-to-matlab.gforge.inria.fr/*, 2014.

[24] Pascal Vincent, Hugo Larochelle, Isabelle Lajoie, Yoshua Bengio, and Pierre-Antoine Manzagol. Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion. *Journal of Machine Learning Research*, 11:3371–3408, 2010.

[25] T. Yamashita, M. Tanaka, E. Yoshida, Y. Yamauchi, and H. Fujiyoshii. To be bernoulli or to be gaussian, for a restricted boltzmann machine. In *Pattern Recognition (ICPR), 2014 22nd International Conference on*, pages 1520–1525, 2014.

[26] Steve Young, Gunnar Evermann, Mark Gales, Thomas Hain, Dan Kershaw, Xunying (Andrew) Liu, Gareth Moore, Julian Odell, Dave Ollason, Dan Povey, Valtcho Valtchev, and Phil Woodland. *The HTK Book*. Cambridge University Engineering Department, http://htk.eng.cam.ac.uk/, htk version 3.4 edition, 2006.

[27] Dong Yu and Michael L. Seltzer. Improved bottleneck features using pretrained deep neural networks. In *INTERSPEECH-2011*, 2011.

TRITA TRITA-EE 2016:043
ISSN 1653-5146