# Spectral Speech Enhancement using Deep Neural Networks

- Design, Analysis & Evaluation -

A Thesis Submitted to the Department of Electronic Systems
Aalborg University, by

Anders Post Jacobsen & Morten Kolbæk

In Partial Fulfillment of the Requirements for the Degree of
Master of Science (MSc) in Engineering (Signal Processing and Computing).

June 2015

Aalborg University
Department of Electronic Systems
Fredrik Bajers Vej 7B
DK-9220 Aalborg

**AALBORG UNIVERSITY**
STUDENT REPORT

**Department of Electronic Systems**
Fredrik Bajers Vej 7
DK-9220 Aalborg Ø
http://es.aau.dk

**Title:**
Spectral Speech Enhancement with Deep Neural Networks

**Theme:**
Signal Processing and Computing

**Project Period:**
Sep. 2014 - Jun. 2015

**Project Group:**
15gr1073

**Participants:**
Anders Post Jacobsen
Morten Kolbæk

**Supervisors:**
Zheng-Hua Tan
Jesper Jensen

**Copies:** 3

**Page Numbers:** 160

**Date of Completion:**
June 2, 2015

**Abstract:**

Speech enhancement is an important issue within a wide range of applications such as mobile phones, speech recognition and hearing aids. In various acoustic environments, especially at low Signal-to-Noise Ratios (SNRs), the goal of speech enhancement methods is to solve the cocktail party problem. Regarding intelligibility, different machine learning methods that aim to estimate an ideal binary mask have revealed promising results. This master's thesis covers the work of speech enhancement by use of the machine learning method Deep Neural Network (DNN). In particular, a MATLAB implementation of a system based on DNNs for estimating an ideal binary mask was carried out.
Simulations have revealed that the proposed DNN based speech enhancement algorithm can enhance noisy speech in terms of an intelligibility predictor (STOI) and a quality predictor (PESQ). Likewise, it has been found that by using a soft mask, instead of a binary mask, additional improvement in STOI and PESQ can be achieved.
The project is suggested and motivated by both Aalborg University and Oticon A/S.

# Contents

# List of Figures

# List of Tables

# List of Acronyms

| | |
|---|---|
| **AI** | Artificial Intelligence |
| **AMS** | Amplitude Modulation Spectrogram |
| **ANN** | Artificial Neural Network |
| **ASA** | Auditory Scene Analysis |
| **ASR** | Automatic Speech Recognition |
| **AUC** | Area Under ROC curve |
| **BBRBM** | Bernoulli-Bernoulli RBM |
| **BM** | Boltzmann Machine |
| **BNN** | Biological Neural Network |
| **CASA** | Computational Auditory Scene Analysis |
| **CD** | Contrastive Divergence |
| **CDF** | Cumulative Distribution Function |
| **CNN** | Convolutional Neural Network |
| **CPU** | Central Processing Unit |
| **DBN** | Deep Belief Network |
| **DBRNN** | Deep Bidirectional Recurrent Neural Network |
| **DCT** | Discrete Cosine Transform |
| **DFT** | Discrete Fourier Transform |
| **DNN** | Deep Neural Network |
| **ERB** | Equivalent Rectangular Bandwidth |
| **EM** | Expectation-Maximization |
| **FA** | False Alarm |
| **FCT** | Fast Cosine Transform |

**FFT** Fast Fourier Transform

**FNN** Feed-forward Neural Network

**FIR** Finite Impulse Response

**FLOPS** Floating-point Operations Per Second

**GBRBM** Gaussian-Bernoulli RBM

**GMM** Gaussian Mixture Model

**GPU** Graphics Processor Unit

**HINT** Hearing In Noise Test

**HIT-FA** HIT minus False Alarm

**HPC** High Performance Computing

**IBM** Ideal Binary Mask

**IID** Independent and Identically Distributed

**IIR** Infinite Impulse Response

**ITS** Inverse Transform Sampling

**IUT** Implementation Under Test

**KLD** Kullback-Leibler Divergence

**L-BFGS** Limited-memory Broyden–Fletcher–Goldfarb–Shanno

**LP** Linear Prediction

**LC** Local Criterion

**MAP** Maximum a Posteriori

**MCMC** Markov Chain Monte Carlo

**ME** Miss Error

**MFCC** Mel-Frequency Cepstrum Coefficient

**MIMD** Multiple Instruction Multiple Data

**ML** Maximum Likelihood

**MMSE** Minimum Mean Square Error

**MRF** Markov Random Field

**NN** Neural Network

**OIM** Objective Intelligibility Measure

**PT** Parallel Tempering

**PCD** Persistent Contrastive Divergence

**PESQ** Perceptual Evaluation of Speech Quality

**PLP** Perceptual Linear Prediction

**PSD** Power Spectral Density

**PDF** Probability Density Function

**PMF** Probability Mass Function

**RASTA** Relative Spectral

**RASTA-PLP** Relative Spectral Transform - Perceptual Linear Prediction

**RBM** Restricted Boltzmann Machine

**ROC** Receiver Operating Characteristics

**ReLu** Rectified Linear unit

**RNN** Recurrent Neural Network

**SIMD** Single Instruction Multiple Data

**SISD** Single Instruction Single Data

**SNR** Signal-to-Noise Ratio

**STFT** Short-Time Fourier Transform

**STOI** Short-Time Objective Intelligibility

**SVM** Support Vector Machine

**TIMIT** Texas Instruments Massachusetts Institute of Technology

**T-F** Time Frequency

**VAD** Voice Activity Detection

**SSN** Speech Shaped Noise

**WSS** Wide Sense Stationary

# Preface

This master's thesis consists of the work done by the students Anders Post Jacobsen and Morten Kolbæk as part of the fulfillment of their Master of Science (MSc) degree in Engineering (Signal Processing and Computing) at Aalborg University.

The thesis is based on a project proposal prepared by Professor Jesper Jensen from Oticon A/S and Aalborg University and Associate Professor Zheng-Hua Tan from Aalborg University. Both of them have functioned as supervisors during the project period.

The scope of the thesis was to investigate how Deep Neural Networks (DNNs) can be used in a speech enhancement context and more specifically it was of interest to investigate if the intelligibility improvement reported in [Healy et al., 2013] is realistic, in a real life scenario.

In the Neural Network (NN) communities an abuse of terminology is abound specially related to the word *backpropagation*. To minimize the degree of confusion about the use of *backpropagation* it is now described how the word is used in this thesis. *Backpropagation* is a method used to compute partial derivatives of Feed-forward Neural Networks (FNNs) for which an optimization algorithm, such as gradient descent, can be used to train [Rumelhart et al., 1986]. However, in the literature the term *backpropagation* or *backpropagation training* is often used to describe that a NN, of any kind, is trained using the partial derivatives computed using the *backpropagation* method combined with some optimization method [Healy et al., 2013] [Chen and Lin, 2014] [Hinton et al., 2012a]. In this thesis it has been decided that the word *backpropagation* will be used in a similar way and hopefully the meaning is clear from the context.

Another abuse of terminology is related to the terms DNN, and Deep Belief Network (DBN). In general a DNN is just a NN with more than one hidden layer [Hinton et al., 2012a]. NNs and DNNs have been known for decades but in 2006 the term DBN was introduced, which refers to a probabilistic generative model that can be used to initialize a DNN [Hinton et al., 2006]. The introduction of this new method of initializing or pre-training a DNN led to the two words being used interchangeably [Chen and Lin, 2014] [Mohamed et al., 2012]. In this thesis the intention was to use the two terms correct such that when the term DBN is mentioned a probabilistic generative model is meant and when the word DNN is used a non-probabilistic ordinary FNN is meant. In Chapters 4 and 5 a detailed description of DNNs and DBNs are given.

A copy of this master's thesis as well as audio samples and test results are available on the appendix DVDs. The master's thesis is structured in the following way:

**Chapter 1**

This chapter defines the objectives of the thesis and motivates its relevance.

**Chapter 2**

This chapter presents some basic theory related to signal processing of human speech.

**Chapter 3**

This chapter presents some classical speech enhancement algorithms.

**Chapter 4**

This chapter presents the basic theory about Artificial Neural Networks (ANNs).

**Chapter 5**

This chapter presents the basic theory about DNN pre-training using DBNs.

**Chapter 6**

This chapter presents the design of a speech enhancement system using DNNs.

**Chapter 7**

This chapter presents test results and discusses the performance of the system designed in Chapter 6.

**Chapter 8**

This chapter concludes on the results based on the objectives defined in Chapter 1.

**Appendix A, B, C, D & E**

These appendices contain details about derivations of different algorithms as well as the MATLAB implementation of the speech enhancement system designed in the thesis.

Aalborg University, June 2, 2015

Anders Post Jacobsen
<ajacob10@student.aau.dk>

Morten Kolbæk
<mkolba13@student.aau.dk>

# Chapter 1

# Introduction

Improving speech quality and speech intelligibility of noisy audio, is of great interest. Enhancing speech is vital in a vast amount of applications, such as mobile communications, speech recognition systems, Voice over IP (VoIP) and in hearing aids, to mention a few. What all these applications have in common is that they are often used in a noisy environment such as a metro station, a shopping mall or in a forest on a rainy day. Situations in which the sound of interest, being a person speaking in a cell phone for example, is being disturbed by the naturally occurring sounds from the background.

Whether or not the sound from the background is considered as a disturbance, depends on the situation. A musical background at a restaurant might be pleasant to normal hearing people but perceived as a disturbance for hearing impaired people for which the voices of their companions are the sounds of interest.

In this case a standard hearing aid, simply amplifying the audio, will not alleviate such a problem and more intelligent speech enhancement algorithms are required if hearing aids should be able to assist hearing impaired persons in such and similar settings [Wang, 2008].

The primary task for an effective speech enhancement algorithm is hereby twofold. First, the disturbance must be identified and secondly it must be attenuated effectively without affecting the target signal.

Recent advances within the field of machine learning and speech processing have introduced new methods which address primarily the second problem about segregating the target signal from the disturbance and the scope of this master's thesis is to investigate how these new methods perform.

## 1.1 The Cocktail Party Problem

The problem all the applications described above are facing is similar and can be described as the "cocktail party problem", which describes the task of hearing or detecting a sound of interest in a complex auditory setting, just as the ones mentioned above [McDermott, 2009].

In 1953 a paper reported that the human brain is capable of segregating a mixture of speakers and hereby focus their auditory attention on only one speaker [Cherry, 1953]. These findings led to the research field known as Auditory Scene Analysis (ASA) and the results reported in [Cherry, 1953] has, as late as in 2013, been supported by clinical controlled experiments. Even though this effect has been known for more than 60 years it is still unknown exactly how the brain does speech segregation [Golumbic et al., 2013].

The fact that the normal human brain is so successful in auditory segregation has lead to the research field known as Computational Auditory Scene Analysis (CASA) [Wang and Brown, 2006]. Where ASA attempts to identify how the human brain does speech segregation from a physiologically point of view, CASA is more practical oriented and is attempting to develop algorithms that perform speech segregation in practice, using techniques from ASA and traditional digital signal processing [Wang and Brown, 2006].

The cocktail party problem is an active field of research [Simpson et al., 2015] [Golumbic et al., 2013] and recently, advances have been made in the CASA field with the development of the Ideal Binary Mask (IBM) which is further described in Section 3.2. The IBM approach is a speech segregation method inspired from ASA which has been proposed as the goal of CASA [Wang, 2005]. Since it is necessary to have the clean signal and noise signal in isolation to obtain the IBM, it should be emphasized that the approach is not practical applicable. Algorithms that estimates the IBM have though been capable of significantly increasing speech intelligibility of noisy speech signals [Kim et al., 2009] [Hang and Wang, 2012]. Combined with recent advances in the neural network communities, known as Deep Learning, has led to the current state-of-the-art speech segregation algorithm [Healy et al., 2013].

## 1.2 Deep Learning

Deep Learning is an emerging area within the field of machine learning which constitute hierarchical learning algorithms based on DNNs [New York Times, 2012] [Deng and Yu, 2014] [Bengio et al., 2015]. The concepts are inspired from nature and from how it is believed that humans solve complex problems such as speech, speaker and object recognition [Bengio, 2009]. What is essential in Deep Learning is that algorithms learn their own representation of the data, which is different from the previous machine learning paradigm where features were primarily hand-crafted and the actual machine learning was limited to be purely discriminative, in the case of a classification task [Le et al., 2011b] [Deng and Yu, 2014] [Bengio et al., 2015]. In practice the feature learning is achieved by DNN pre-training which are described in detail in Chapter 5

Deep learning has in recent years achieved remarkable results in a lot of different applications [Le et al., 2012] [He et al., 2015] [Taigman et al., 2014] [Hannun et al., 2014] [Töscher et al., 2009] [Kaggle, 2012] [IDSIA, 2014] [Mnih et al., 2015].

In [Le et al., 2012], for examples, a team at Google lead by Andrew Y. Ng, trained a DNN, having 1 billion trainable parameters, on 10 million randomly chosen images from YouYube. The training was unsupervised and a computer cluster consisting of 1000 computers trained the DNN for three days. When the training was stopped it was found that the DNN had, by its own, constructed high level feature detectors capable of detecting among other things cats and human faces. This is without ever have been told what to look for.

In [Kaggle, 2012] a team from the University of Toronto lead by George Dahl, Ph.D student of Geoffrey Hinton who is considered one of the leading scientists of Deep Learning, won a $22,000 competition held by the pharmaceutical company Merck. The competition was about identifying the best statistical techniques for predicting biological activities of different molecules, which are involved in the design of new medicine. George Dahl and his team won the competition using a DNN.

In [Töscher et al., 2009] a team from Commendo and AT&T won a $1,000,000 competition held by Netflix where the objective was to design a collaborative filtering algorithm used to predict user ratings for films. The winning solution was similar to the former examples based on DNNs and Deep Learning.

One of the more spectacular achievements using Deep Learning is done by the company DeepMind [Mnih et al., 2015]. They managed to train a DNN to play old arcade video games and the DNN even learned to play the games better than a human professional [Mnih et al., 2015] [BBC Newsnight, 2014]. This achievement is similar to the IBM chess computer Deep Blue which in 1997 won over the, at that time, chess world champion Garry Kasparov. The major difference between the two achievements is that Deep Blue was designed for playing chess and was using a brute force strategy [IBM Research, 2015], whereas DeepMind used a DNN that did not know anything about the task it was supposed to solve. The DNN learned by itself how to play the different games it was presented for. DeepMind was last year bought by Google for 650 million dollars, which is an indication on how big a future Google believes Deep Learning has [The Guardian, 2014].

Other remarkably results provided by Deep Learning, which has a higher commercial value, are significant improvements in object recognition, face recognition and speech recognition. Specifically, current state-of-the-art object recognition algorithms has achieved above-human performance [He et al., 2015] [Taigman et al., 2014] [Hannun et al., 2014].

Due to all the remarkably results Deep Learning has become a buzz word and a Deep Learning hype has emerged [New Yorker, 2013] [The Huffington Post, 2015]. A hype so big that very polarized debates within the computer science community is debating whether or not Deep Learning will lead to the technological singularity, which is the point where Artificial Intelligence (AI) exceeds human intelligence [The Register, 2015] [IEEE Spectrum, 2015] [Wired, 2015] [Kurzweil, 2006]. Even the prominent engineer, founder of PayPal, Tesla Motors and SpaceX, Elon Musk has been participating in the debate and donated $10,000,000 to a research project aiming at ensuring AI keeps beneficial for the human race [Future of Life Institute , 2015] [CNN Money, 2014]. Something other prominent researchers within the field believe is nonsense [The Register, 2015] [IEEE Spectrum, 2015] [Dr. Andrew Ng, 2015].

## 1.3 Thesis Scope

Communicating in noisy environments can be a demanding task but it is even more demanding for hearing impaired people since they often lack the ability to focus their auditory attention on one out of many speakers [McDermott, 2009]. In Denmark approximately 800.000 people, out of a population of approximately 5.6 million, suffer from degraded hearing and approximately 300.000 use a hearing aid to alleviate their hearing disability [Christensen, 2006] [Bengtsson and Røgeskov, 2010] [Danmarks Statistik, 2014].

It is hereby of utmost relevance to solve the cocktail party problem and the ideas presented in [Healy et al., 2013] might be a step in the right direction. In [Healy et al., 2013] techniques from CASA and Deep Learning have been combined in the design of a speech enhancement algorithm reportedly capable of increasing speech intelligibility for both normal hearing and hearing impaired persons which according to Healy et al. is the first time such results have been reported.

Despite the fact that the work presented in [Healy et al., 2013] is performed by acknowledged researchers, recent studies have questioned the validity of their results [May and Dau, 2014]. In [Healy et al., 2013] the same noise recordings where used in both the training and testing phase which is not considered as good practice in the machine learning communities and led to the suspicion that the results reported in [Healy et al., 2013] could not be realized in practice. It is therefore of interest to investigate if the speech enhancement method reported in [Healy et al., 2013] performs as well as claimed, in real life scenarios. The scope of this thesis is thereby to investigate the algorithmic performance of this method and investigate if the results reported are realistic. If the results are as good as claimed the method show great potential for a large range of applications where speech enhancement is essential, including hearing aids. The scope of the thesis has been delimited into the five objectives described below.

### Objective 1

The first objective is concerned with the effect of unsupervised pre-training of DNNs. Based on the significant interest about Deep Learning it is expected that pre-trained DNNs perform better than DNNs trained in a purely supervised fashion. However, recent papers using DNNs [Huang et al., 2014a] [Simpson, 2015] have not reported the use of pre-training, presumably because they did not find any performance improvement by using pre-training, which is stated directly in [Maas et al., 2012].

In [Deng et al., 2013] it is argued that DNNs are difficult to train, due to the fact that it requires a lot of experience to properly tune all parameters correctly. It is also stated that the effect of pre-training is dependent on the amount of labeled training data. From the arguments above it is found interesting and relevant to investigate what effect pre-training has on the performance of the speech enhancement system to be designed.

### Objective 2

The second objective is related to the construction of the training and test data. In the machine learning community common practice is to divide a data set into a training set, a test set and perhaps a validation set [Bishop, 2009]. The machine learning algorithm is then trained using the training and validation sets and tested using the test set, which for the algorithm, is unknown, hence a realistic measure of performance is achieved.

As already mentioned, in [May and Dau, 2014] it is pointed out that the methodology used by [Healy et al., 2013], but also in [Kim et al., 2009], was not scientifically in line with good practice, since the same noise data (looped-noise) was used for generating training and test data. This means that the DNNs may be able to learn the noise sequence and simply subtracting it out, hence achieving a performance which is non-realizable in practice. It is hereby of interest to investigate how big an impact it has on the performance of a DNN when the same noise is used for training and testing, opposed to the scenario when unique (non-looped) noise is used for training and testing.

### Objective 3

The third objective is primarily based on inspiration found in the Automatic Speech Recognition (ASR) community. In the ASR community it has been found that ASR systems become more robust against noise if a pre-processing speech enhancement stage was used, utilizing a soft mask opposed to a hard mask i.e., the IBM [Barker et al., 2000] [Raj and Stern, 2005] [Narayanan and Wang, 2013]. It has been, and still is, a topic of research in the speech enhancement communities whether or not a soft mask performs better than a hard mask in a speech enhancement context [Singh et al., 2014] [Wang, 2008] [Narayanan and Wang, 2013] [Loizou, 2013]. Even though it has been found that the soft mask improves quality it has not yet been clarified if a soft mask improves speech intelligibility and there are opposing arguments in the literature whether or not a soft mask has potential to improve speech intelligibility [Loizou, 2013, p. 614] [Wang, 2008] [Jensen and Hendriks, 2012]. Hence, investigating the effects regarding intelligibility of using a soft mask opposed to a binary mask, in a DNN based speech enhancement system, is of great interest.

### Objective 4

The fourth objective is related to the robustness of the DNNs, which is evaluated on how good the DNNs generalize to data they have not seen during training. This is done to evaluate the performance of the algorithm in an environment as close as possible to a real life scenario. Such an evaluation could be based on presenting a DNN for a new noise type or the same noise type but at different SNRs. It could also be related to the speech material, which could be altered or the speaker which could be changed. It could also be more technical related such as presenting the algorithm for audio recorded with different microphones. These test environments can be setup in an unlimited number of ways and in this thesis it has been decided to investigate how sensitive the DNNs are to new noise types and alterations in SNRs.

### Objective 5

The fifth objective is included to evaluate the performance of the DNNs developed during this thesis against two more traditional methods, namely the Short-Time Fourier Transform (STFT) based Wiener filter and the spectral amplitude Minimum Mean Square Error (MMSE) estimator [Loizou, 2007]. The argument for the comparison is to identify possible strengths and weaknesses the DNN based speech enhancement method might have, compared to two traditional methods.

The five objectives can be summarized into the corresponding five questions which are to be answered during the thesis. That is, the answer to these five questions will be presented in the conclusion in Chapter 8.

## Objectives

1. Does unsupervised pre-training of DNNs, when used for IBM estimation, improve performance?

2. What is the effect of using looped noise, as opposed to non-looped noise, regarding DNN classification performance?

3. Does using a soft decision rather than a hard decision, in a DNN based speech enhancement system, improve speech intelligibility and/or quality?

4. How well does DNNs used for IBM estimation generalize to unseen noise types and SNRs?

5. How do DNN based speech enhancement algorithms perform relative to the classical STFT based Wiener filter and the spectral magnitude MMSE estimator?

# Chapter 2

# Fundamentals of Speech

When designing speech enhancement algorithms, having some of the fundamental theory and terminology within speech processing in place, is imperative. This chapter will give a short introduction to some important concept used in speech processing. A brief overview of the speech production process and characteristics of speech signals is given since the latter is a consequence of the former. It turns out that speech production, seen across short time durations, can be expressed as a relatively simple linear model, which represents an approximation of the speech signal. These subjects are discussed in Section 2.1. When analyzing and performing enhancement on speech signals it is mostly done is some Time Frequency (T-F) domain. In this context, a special type of the Fourier transform is very useful and widely used, namely the STFT. Another T-F representation, that takes into account certain characteristics of the human auditory system, is the filter bank based cochleagram. These T-F representations are explained in Section 2.2. Since it is often desired to reconstruct the time signal from its T-F representation, synthesis methods is also explained in Section 2.4. When one is to represent a speech signal it can be done in a both compact and perceptual meaningful way. This can be carried out in terms of different acoustic features. These features can for example be used as input to a machine learning classifier. Different such features will be discussed in Section 2.5.

## 2.1 The Speech Signal and Production

The very basic aim of speech is to communicate between humans, meaning that a speech signal is generated by a speaker and received by a listener. Speech is an acoustic signal, which is highly non-stationary and this property makes it difficult to analyze. It should be noted that non-stationary is meant in a statistical sense, meaning that the statistical properties vary with time. In this thesis, the term stationary signal is therefore meant as a realization from a Wide Sense Stationary (WSS) stochastic process. Short frames (10-30 ms) of speech is though often assumed to be stationary and this implies Fourier analysis to be applicable [Loizou, 2007, p. 31].

Since speech is signals communicated among humans, the human anatomy that generates such signals will be subject to a brief description since any prior knowledge of the target signal can, and should, be used when doing speech enhancement. There are three main parts involved in the speech production process in terms of human anatomy. The lungs, the larynx (also known as the voice box) and the vocal tract. In the context of speech production, the functionality of the lungs is to deliver air pressure used to generate speech. When inhaling, the air pressure in the lungs decreases. When exhaling, the air pressure in the lungs increases and this causes air to flow into the larynx. The larynx is build up of different muscles and ligaments which control the vocal folds. When the vocal folds are vibrating in a periodic manner, the speech signal is said to be in the voiced state. The time of one cycle in this periodicity is called a pitch period. Furthermore, the reciprocal of a pitch period is called the fundamental frequency. The speech signal is said to be in an unvoiced state when the vocal folds are non-vibrating. Therefore, the speech signal have a periodic time representation in the voiced state, and a non-periodic time representation in the unvoiced state. In the voiced state vowels are generated and in the unvoiced state many consonants are generated [Loizou, 2007, p. 50]. The last stage of the speech production is the vocal tract where the oral cavity and the nasal cavity is located. The vocal tract acts as a time-varying linear filter shaping the speech output signal. This filtering function is depended on the placement of the tongue, teeth, lips and jaw.

The anatomic description of the human speech production system can be modeled in a somewhat more engineering appealing way. Here the vocal tract can be seen as a linear filter applied to either a pulse generator or a random noise generator depending on the voiced state. This linear filter is characterized by a set of resonance frequencies which varies with the shape of the vocal tract. Because these resonance frequencies form the speech spectrum they are referred to as formants[Deller et al., 1993, p. 107]. The linear speech model is illustrated in Fig. 2.1.This model can also be described mathematically as

$$s(t) = e(t) * v(t) \tag{2.1}$$

where $s(t)$ is the speech signal, $e(t)$ is the excitation model that is either a pulse train generator or a random noise generator and the $*$ is the convolution operator. The last stage $v(t)$ is the impulse response of the vocal filter. The speech model can likewise be expressed in the frequency domain as

$$S(f) = E(f) \cdot V(f). \tag{2.2}$$

This simplified linear model of the speech production system only contains four sets of parameters, namely the fundamental frequency, the parameters of the noise generator, the voice state and the vocal tract parameters. From these four sets of parameters the speech signal $s(t)$ can be approximated. Of course this representation of the speech signal is only an approximation but the model have been widely used in applications such as low-bit rate speech coding, where it is an advantage to represent the speech signal in a compact form with few parameters [Loizou, 2007, p. 55]. Since in many applications, only the speech signal is observable the parameters of the excitation model and the vocal filter cannot be obtained directly. Cepstral analysis, which is described in Section 2.5, can be used to deconvolve the excitation model from the impulse response of the vocal filter. Hereby the parameters can be obtained when only measuring the speech signal.

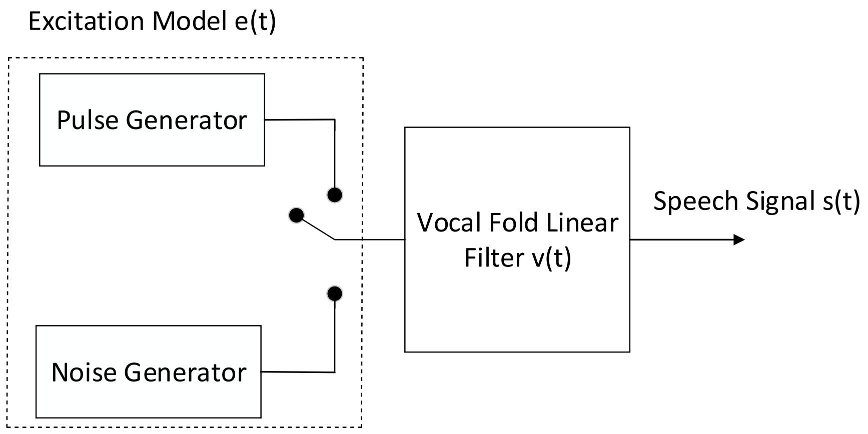


**Figure 2.1:** The speech model includes both an excitation model and a vocal filter. The switch in the model controls if the state is voiced or unvoiced meaning that the signal from the excitation model comes from either a pulse generator or a random noise generator.

## 2.2 Short-Time Fourier Transform & the Spectrogram

The Discrete Fourier Transform (DFT) is widely used in the analysis of various types of signals. For deterministic signals one can find the frequency spectrum and for stationary stochastic signals the Power Spectral Density (PSD) can be obtained via its auto-correlation function. One of the drawbacks, in the context of speech signals, is that such signals are highly non-stationary. Despite this, the DFT can be applied to small frames of speech (10-30 ms) in which it is assumed that the signal is approximately stationary. This is the motivation behind the Short-Time Fourier Transform (STFT). The idea is to cut the entire signal into successive and overlapping frames of L samples. Next, an analysis window is applied and finally the DFT of each windowed frame is computed. Hereby, the discrete STFT will both be a function of time and frequency. Formally, the discrete STFT is defined as in Eq. (2.3) [Loizou, 2013, p. 31]

$$X(n,\omega) = \sum_{m=-\infty}^{\infty} x(m)w(n-m)e^{-j\frac{2\pi}{N}\omega m} \tag{2.3}$$

where $x(m)$ is the discrete time signal and $w(m)$ is some window function. It can be seen that the STFT is a function of two variables, namely a time index $n$ and a frequency index $\omega$ where $\omega = 0, 1, ..., N-1$ hence a T-F representation is obtained. N can be interpreted as the frequency resolution of the STFT. The two dimensional function $|X(n,\omega)|^2$ is called a spectrogram [Loizou, 2007, p. 32]. In terms of speech signals it is useful to see how the power spectrum changes over time and this makes the spectrogram a powerful tool. Figure 2.2 shows the spectrogram of a speech signal and it is seen that it is both a function of time and frequency. The spectrogram is generated by the MATLAB Voicebox function *spgrambw* [Mike Brookes, 1997]. The colors in the spectrogram illustrates different power intensities.

Finally, it should be noted how the window length $L$ affects the appearance of the spectrogram. A long duration window will give god frequency resolution but a poor time resolution. In this case a narrow-band spectrogram is generated. On the other hand a short duration window will give poor frequency resolution but a god time resolution. When this is the case the spectrogram is refereed to as wideband [Loizou, 2007, p. 43].

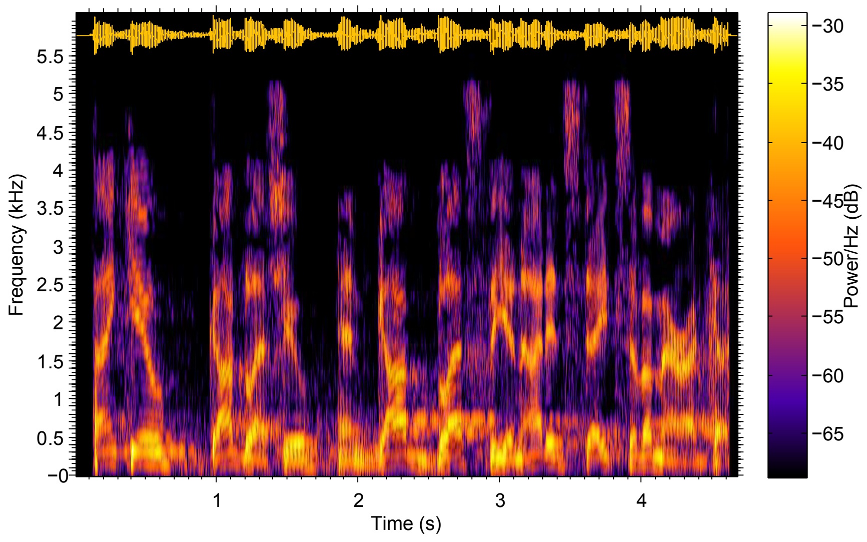


**Figure 2.2:** The spectrogram reveals how power at different frequencies changes over time. Above the spectrogram, the waveform of the signal is plotted in yellow. A Matlab script of the plot can be found on the appendix DVDs.

## 2.3 The Gammatone Filter Bank & the Cochleagram

The aforementioned spectrogram makes no specific assumption on how the human auditory system works. Therefore, another approach is to make a T-F representation related to how the human auditory system works. This implies a filter bank that mimic the way the human hearing is functioning. One approach to model such filters are by use of a gammatone filter bank. The impulse response of a gammatone filter is given by [Wang and Brown, 2006, p.15]

$$g(t) = t^{N-1} \exp[-2\pi t b(f_c)] \cos(2\pi f_c t + \phi) u(t) \tag{2.4}$$

where $N$ is the filter order, $f_c$ is the center frequency, $\phi$ is the phase, $u(t)$ is the unit step function and $b(f_c)$ is the bandwidth of the filter given a specific center frequency.

A reasonable approximation for the filter has been proposed with a filter order of $N = 4$ [Wang and Brown, 2006, p.16]. The bandwidth used for the gammatone filter bank is the Equivalent Rectangular Bandwidth (ERB). The ERB is defined as the bandwidth of a rectangular window having the same maximum amplitude and power as a corresponding filter as illustrated in Fig. 2.3.

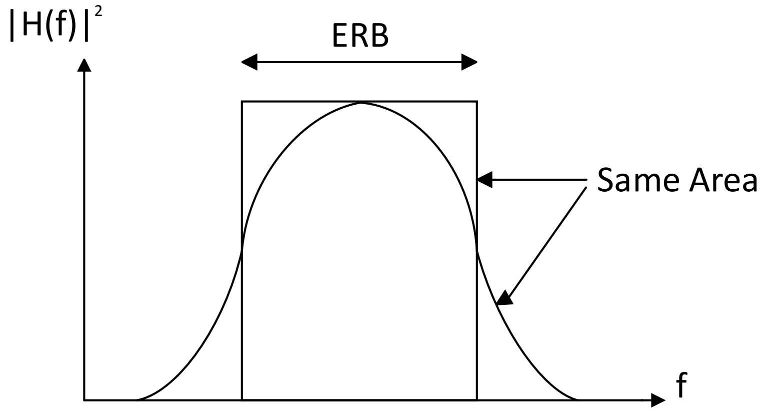


**Figure 2.3:** A rectangular window having the same maximum amplitude value and power (area under the curve) defines the ERB of the corresponding filter.

A decent ERB conversion of an auditory filter used by humans have been shown to be [Wang and Brown, 2006, p.16]

$$ERB(f_c) = 24.7 + 0.108 f_c. \tag{2.5}$$

From Eq. (2.5) it can be seen that a center frequency is converted to a bandwidth and when applied on a filter bank with multiple center frequencies the bandwidth is increased when going from a lower to a higher center frequency. Since the output of the gammatone filter bank is framed, a T-F representation of the signal is now obtained. This representation is referred to as a cochleagram [Wang and Brown, 2006, p.18]. Fig. 2.4 shows the amplitude response of a 64-band gammetone filter bank with center frequencies from 50-8000 Hz. As seen the bandwidth of the filters increases with the center frequencies of the filters. The MATLAB implementation of the gammetone filter bank is made by ZZ Jin, DeLiang Wang and JF Woodruff and is available from [Jin and Wang, 2007a]. It should be noted that this implementation also contains a loudness-based gain adjustment so filters at different center frequencies have different gains.

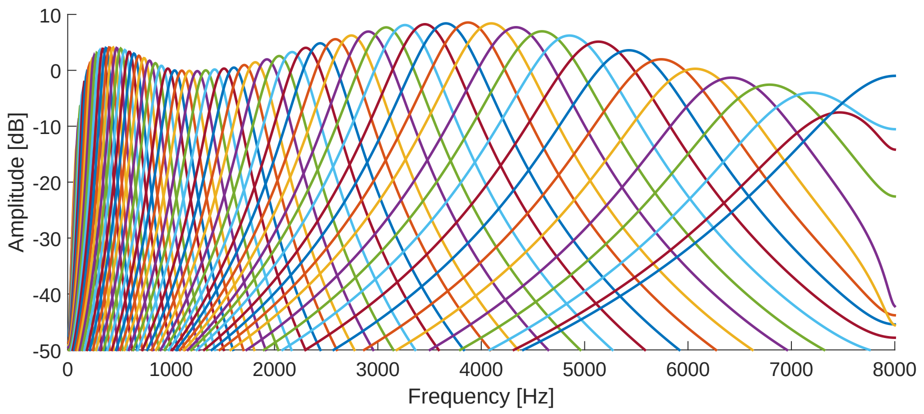


**Figure 2.4:** Amplitude response of loudness-based gain adjusted Gammatone filter bank

## 2.4 Speech Synthesis

When a T-F representation of a speech signal is available it is often necessary to reconstruct its corresponding time signal. This is performed by speech synthesis which can be carried out in different ways. For example, when a signal denoted $x(n)$ is to be reconstructed from its STFT $X(n, \omega)$, a commonly used method in the context of speech enhancement, is the overlap-and-add method [Loizou, 2007, p. 38]. Here it should be assumed that $X(n, \omega)$ is sampled in time for every $R$ samples and this will be denoted $X(rR, \omega)$. The overlap-and-add method is then given by the expression [Loizou, 2007, p. 38]

$$\hat{x}(n) = \sum_{r=-\infty}^{\infty} \left[ \frac{1}{N} \sum_{k=0}^{N-1} X(rR, \omega_k) e^{j\omega_k n} \right] \tag{2.6}$$

where $\hat{x}(n)$ is the reconstructed version of $x(n)$. The expression in the brackets of Eq. (2.6) is the inverse DFT and this will provide the sequence [Loizou, 2007, p. 38]

$$y_r(n) = x(n)w(rR - n). \tag{2.7}$$

By inserting Eq. (2.7) into (2.6) we obtain [Loizou, 2007, p. 38]

$$\hat{x}(n) = \sum_{r=-\infty}^{\infty} y_r(n) = x(n) \sum_{r=-\infty}^{\infty} w(rR - n). \tag{2.8}$$

The way to interpret this expression is that all sequences, that overlap at a specific time index $n$, is summed. Furthermore, if the summation is constant for all $n$ then the time signal $x(n)$ can be reconstructed exactly within a constant, i.e. $\hat{x}(n) = Cx(n)$ where $C$ is some constant [Loizou, 2007, p. 38]. If a L-point window is used in Eq. (2.8) a commonly used value of $R$ is $L/2$, meaning 50 % frame overlap.

One method to synthesize an output signal from a filter bank is described in [Wang and Brown, 2006, p.23]. Here the method is applied on a gammatone filter bank which is described in Section 2.3. The gammatone filtered signal is time reversed, passed through the filter and finally time reversed again. This procedure is carried out to remove across-channel phase shifts and produce an output with zero phase [Wang and Brown, 2006, p.23]. However, this time reversing is non-real time applicable and for real time applications a phase compensated gammatone filter bank must be used [Wang and Brown, 2006, p. 17]. The phase-compensated signal in each channel can now be divided into frames and multiplied by a Hanning window to obtain a T-F representation. A broad class of speech enhancement methods, further described in Chapter 3, operates in this domain by applying weights to each T-F unit. The weighted outputs, from all channels, are then summed up to produce the synthesized time signal.

## 2.5 Acoustic Feature Representation

When talking about speech signals there are, like any other types of signals, various ways of representing it. This could include time domain, amplitude spectrum, power spectrum and so on. Another way of representing a speech signal is in terms of different kinds of acoustic features. By such a representation, key aspects of the speech signal is preserved and the unimportant aspects are removed leading to a compact representation. Therefore, such features will typically take into account the way humans perceive sound. In the following different features will be presented.

### 2.5.1 Mel Frequency Cepstrum Coefficients

As just mentioned, acoustic features will typically take into account the way in which humans perceive sound. This also includes mapping the normal frequency representation to a representation that is based on how humans perceive tones at different frequencies. One such mapping is the mel scale. A mel is a measure of the perceived frequency of a tone. The relationship between mels and actual frequency is approximate linear below 1 kHz and logarithmic above. One way to carry out this approximation is given by [Deller et al., 1993, p. 380].

$$F_{mel} = \frac{1000}{\log(2)}\left(1 + \frac{F_{Hz}}{1000}\right). \tag{2.9}$$

The mel scale is derived by an experiment where test persons were exposed to a reference tone at 1 kHz and then asked to increase the frequency until they perceived certain tones [Deller et al., 1993, p. 380]. This experiment revealed that humans are better at recognizing the differences between frequencies in the low-end of the frequency scale than the high-end.

Since cepstral analysis was mentioned, in Section 2.1, as a deconvolution tool of the speech model, a brief discussion on the cepstrum is given. The frequency domain representation of the speech model from Eq. (2.2) can be separated into two additional parts by using the logarithm as

$$\log S(f) = \log E(f) + \log V(f). \tag{2.10}$$

Taking the inverse DFT of the log spectrum result in two additional parts in what is now refereed to as the quefrency domain [Deller et al., 1993, p. 360]. Even though the signal is now represented in the quefrency domain, the excitation model and the impulse response of the vocal tract filter is separated. More formally a general expression of the Cepstrum can be defined as the inverse Fourier transform of the log spectrum of a time signal $s(n)$. Mathematically this is expressed as in Eq. (2.11) [Deller et al., 1993, p. 355]

$$c(n) = \mathcal{F}^{-1}\{\log |\mathcal{F}\{s(n)\}|\} \tag{2.11}$$

where $\mathcal{F}\{.\}$ denotes the DFT. The rather strange terminology used for cepstral analysis is based on interchanging consonants, meaning that the word spectrum becomes cepstrum.

The Mel-scale and the cepstrum can be combined when calculating the Mel-Frequency Cepstrum Coefficients (MFCCs). The aim of MFCCs is to extract features from an acoustic signal in a perceptual meaningful way. MFCCs are a commonly used feature in speech processing and in [Healy et al., 2013, p. 3031] they are used as input to a DNN used in a speech enhancement algorithm. The method of extracting the MFCCs can be divided into four main parts as listed below and is based on [Huang et al., 2001, p. 314-316].

- Apply the STFT so an estimate of the power spectrum can be obtained via the periodogram
- Apply a mel-frequency filter bank by a triangular weighting function
- Take the logarithm of the output of the filter bank
- Apply the Discrete Cosine Transform (DCT)

Compared to the definition of the cepstrum in Eq. (2.11) the calculation of the MFCC uses the DCT-2 instead of the inverse DFT. This can be done when the log-energy of the output of the filter bank is an even function [Huang et al., 2001, p. 316]. The proof of this will not be investigated further in this thesis. The effect of the DCT is a decorrelation of the signal which are beneficial since a compact representation is then achieved. The Karhunen-Loève transform is the optimal decorrelator wrt. energy compaction. This means that as much energy as possible is placed in as few coefficient as possible. Unfortunately, the Karhunen-Loève transform is data dependent. The Karhunen-Loève transform can though be estimated by the DCT which is not data dependent [Khayam, 2003, p. 15].

### 2.5.2 Relative Spectral Transform and Perceptual Linear Prediction

As explained in Section 2.1 the speech production is represented by the excitation model and the time varying vocal tract filter shown in Fig. 2.1. This representation of the speech production can be modeled using an all-pole system model given by [Deller et al., 1993, p. 268]

$$\hat{V}(z) = \frac{1}{1 - \sum_{i=1}^{M} \hat{a}(i) z^{-i}} \tag{2.12}$$

One way of estimating the parameters is by use of Linear Prediction (LP) where a number of $P$ previous samples are used to predict the current sample. This can be expressed as [Makhoul, 1975, p.563]

$$\hat{s}(n) = -\sum_{i=1}^{P} a_i s(n-i). \tag{2.13}$$

The prediction of the current sample will introduce some error compared to the actual sample $s(n)$ and it is desired to find a set of coefficients that minimizes this error in some sense. A common choice is to use the squared error measure as cost function. If this cost function is differentiated and set equal to zero, it can be written in terms of the sample auto-correlation sequence of the samples as [Makhoul, 1975, p.563]

$$-R(i) = \sum_{k=1}^{P} a_i R(i-k) \tag{2.14}$$

where the auto-correlation is defined as

$$R(i) = \sum_{n=i}^{N-1} s(n)s(n-i) \tag{2.15}$$

and $N$ is the length of the sequence. The expression in Eq. (2.14) is also called the normal equations and can be formulated in matrix/vector notation as $\mathbf{Ra} = -\mathbf{r}$ where $\mathbf{R}$ is a $P \times P$ sample auto-correlation matrix. Therefore, the approach is named the auto-correlation method. $\mathbf{R}$ is a Toeplitz matrix, meaning all elements along a diagonal is constant and this special structure of the matrix can be utilized to solve the equations by the Levinson–Durbin algorithm. By use of the Levinson–Durbin algorithm the complexity of solving the equations is $O(P^2)$ opposed to the $O(P^3)$ which is the complexity of a naive matrix inversion.

Even though the LP method is widely used it has a disadvantage when used in terms of speech processing. The all-pole model estimated by LP makes an approximation of the power spectrum. The approximation error that is introduced is equally weighted across frequency. This is contrary to the way the human auditory system works where the spectral resolution decreases with increasing frequency from above 800 Hz [Hermansky, 1989, p. 1738]. Hynek Hermansky introduced the Perceptual

Linear Prediction (PLP) that uses different concepts from the auditory system to make modifications to the power spectrum, before performing the LP. The methods is described as follows [Hermansky, 1989]. First the speech signal is divided into frames and multiplied by a Hamming window before a Fast Fourier Transform (FFT) transforms it into the complex frequency domain. The absolute value of the real and imaginary part is squared to obtain the power spectrum $P(\omega)$. Next, the frequency axis of the power spectrum is warped to the Bark frequency $\Omega$ according to the formula [Hermansky, 1989, p. 1739]

$$\Omega(\omega) = 6\ln\Big(\omega/1200\pi + \sqrt{(\omega/1200\pi)^2 + 1}\Big) \tag{2.16}$$

Since the level sensitivity of the human hearing is not equal for all frequencies, an equal loudness preemphasis filter is applied. Furthermore, to simulate the nonlinear relationship between the sound signals intensity and its perceived loudness the cubic-root amplitude compression is applied [Hermansky, 1989, p. 1740]. The inverse DFT is computed to yield the auto-correlation and the LP method described above can be applied.

The PLP methods can also be combined with Relative Spectral (RASTA) filtering. The basic idea of RASTA filtering is to suppress spectral content that changes more slowly or more quickly than what is expected of typical speech. This can be carried out as an Infinite Impulse Response (IIR) filter applied to the power spectrum of a specific T-F unit. The RASTA filter is given by the transfer function [Hermansky, 1994, p. 579]

$$H(z) = 0.1z^4\frac{2 + z^{-1} - z^{-3} - 2z^{-4}}{1 - 0.98z^{-1}}. \tag{2.17}$$

Combining the RASTA filtering with the PLP method yield the Relative Spectral Transform - Perceptual Linear Prediction (RASTA-PLP) features.

### 2.5.3 Amplitude Modulation Spectrogram

Another way of representing a speech signal is in terms of the amplitude modulation. Both neurophysiological and psychoacoustical experiments have shown that amplitude modulation frequencies play an important role in the auditory system [Tchorz and Kollmeier, 2002, p. 3-4]. In terms of digital signal processing, Kollmeier and Koch used these findings to define the Amplitude Modulation Spectrogram (AMS) for a binaural noise suppression method [Kollmeier and Koch, 1993]. The AMS is a two-dimensional representation with center frequencies of some filter bank on the ordinate and modulation frequencies on the abscissa. The overall idea of the AMS representation is to use both spectral and temporal information captured from the speech. The AMS representation can for example be used to make distinctions between acoustical objects such as voiced speech, unvoiced speech and various noise types. In general, a speech signal has a characteristic AMS representation which is different from the AMS representation of most types of noise signals [Tchorz and

Kollmeier, 2003, p. 184]. Therefore, the AMS could be used as a representation that discriminate speech from noise. It should be noted that differences in the AMS representation are more significant for voiced speech versus noise than for unvoiced speech versus noise. For voiced speech, the energy in the AMS along the modulation axis is intense, around the fundamental frequency and it multiples. The energy is likewise intense at formant frequencies along the frequency axis. [Wang and Brown, 2006, p. 88]. Since unvoiced speech lacks harmonicity, there will be no intense activity along the modulation axis. Even though the harmonicity in the AMS representation is a very important element when classifying a sound as speech or noise, the representation will also be useful for discriminating unvoiced speech from noise, hence the AMS cannot be replaced by a pitch detection algorithm where the computational complexity is typically smaller [Tchorz and Kollmeier, 2003, p. 191].

Low modulation frequencies up to 8 Hz have also been shown to be important for speech intelligibility [Tchorz and Kollmeier, 2002, p. 3]. When the AMS is calculated on short frames of perhaps 20 ms, one could questioning the method since the low modulation frequencies, important for speech intelligibility, are not captured. Higher modulation frequencies play an important role when it comes to detection of acoustical objects. In a noisy environment and on short segments of speech, humans have the ability to detect signals as being speech even though the meaning of the spoken words is not captured. The higher modulation frequencies that contain information of the pitch seem to be important for the detection of the speech signal [Tchorz and Kollmeier, 2002, p. 3]. Therefore, such modulation frequencies could be of interest in a speech/interference classification type of problem.

An approach for extracting the AMS features is described in [Hang and Wang, 2012], [Kim et al., 2009] and [Healy et al., 2013]. This method will be used throughout the thesis and is described as follows.

First, the sampled signal is bandpass filtered with a gammatone filter bank. The envelope within each frequency channel is extracted using full wave rectification, i.e., taking the absolute value. The envelope signal is then decimated by a factor of 4. The decimated envelope signal is segmented into overlapping frames of 80 samples with overlap of 40 samples. Each frame in each channel is multiplied with a 20 ms Hanning window. The windowed signal is zero padded to a length of 256 and a 256 point FFT is calculated. The FFT computes the modulation spectrum with a frequency bin spacing of 4000 Hz/ 256 = 15.6 Hz. The magnitudes of the FFT is multiplied with 15 triangular windows uniformly spaced from 15.6-400 Hz. For each individual triangular window the FFT magnitudes are multiplied and summed up to produce one AMS coefficient. Hereby 15 AMS coefficients are calculated.

### 2.5.4 Delta & Delta-Delta Features

The above mentioned features each represent the speech signal in different ways. These representations are carried out on individual T-F units, independent on each other Sometimes it is though also useful to have information about the rate of change within successive frames. Therefore, a new set of coefficients called deltas, and delta-deltas are introduced. In it simple form the delta features is just the difference between T-F units across a number of successive frames. This can be written as

$$\Delta[n,k] = c[n+m,k] - c[n-m,k] \tag{2.18}$$

where $\Delta[n,k]$ is the delta coefficient and $c[n,k]$ is the acoustic feature at the n'th frame and k'th frequency bin. The delta-deltas are calculated in a similar manner just using the delta coefficient instead of the acoustic feature. Sometimes delta coefficients are refereed to as differential coefficients, and delta-delta coefficient are referred to as acceleration coefficients to imply their interpretations. Another and more used way of calculating the delta features is to fit the slope of a straight line to the acoustic features in a number of successive frames. This can be written as [Jensen and Tan, 2015, p.188]

$$\Delta[n,k] = \frac{\sum_{t=-m}^{m} tc[n+m,k]}{\sum_{t=-p}^{p} t^2} \tag{2.19}$$

Compared to Eq. (2.18) this calculation includes more samples and thereby the rate of change within a number of successive frames is found. The delta-deltas are found in a similar way as in Eq. (2.19) just by changing $c[n+m,k]$ by $\Delta[n+m,k]$.

# Chapter 3

# Speech Enhancement Algorithms

As mentioned in the introduction in Chapter 1, speech enhancement is of great interest in many different applications. The basic aim of a speech enhancement algorithm is to remove, or at least attenuate, the noise signal from a noisy observation. This should be done in order to increase the intelligibility and quality. The following chapter contains a description of different types of speech enhancement algorithms. First, Section 3.1 contains a description of the classical methods used for speech enhancement. This includes spectral subtraction, Wiener filter and the spectral amplitude MMSE estimator. The spectral amplitude MMSE estimator is henceforth referred to as just the MMSE estimator. In Section 3.2 the concept of the IBM is introduced. In the same section a description of three machine learning methods, that aim to estimate the IBM, is given. Finally, in Section 3.3 a description of the performance measures used for evaluating intelligibility and quality is given.

## 3.1 Classical Methods

Before going into the description about some classical methods the signal model considered is presented as

$$y(n) = x(n) + n(n) \tag{3.1}$$

where $y(n)$ is the observed noisy signal, $x(n)$ is the speech signal and $n(n)$ is the noise signal. The signals are considered as realizations of a stochastic process and the noise process and speech process are assumed independent. By use of the STFT, Eq. (3.1) can likewise be expressed in the T-F domain as

$$Y(n, \omega) = X(n, \omega) + N(n, \omega) \tag{3.2}$$

where $Y(n, \omega)$, $X(n, \omega)$ and $N(n, \omega)$ is the complex valued T-F domain representation of the noisy signal, the noise-free signal and the noise signal, respectively. Even though the time index $n$ strictly speaking refers to one sample, it will henceforth be referred to as a frame index consisting of several samples without change of notation.

A broad class of speech enhancement algorithms operate in the T-F domain and it turns out that estimates of the noise-free signal is generally a function of the noise PSD and speech PSD [Hendriks et al., 2013, p. 8]. However, in practice these are not available in isolation, and have to be estimated. In the following description of existing speech enhancement algorithms, noise PSD and speech PSD are though assumed to be known. In Section 3.1.4 a short description of how the noise and speech PSD can be estimated, is presented.

In general, the derivation of clean speech estimators is performed under the assumption that both the noise-free and the noise signal is present in the noisy observation. This is not always the case in practice since speech pauses occur. Furthermore, both during speech and speech pauses, some of the transform coefficients from Eq. (3.2) will be close to zero. Therefore, a speech present probability estimator is often included in the system [Hendriks et al., 2013, p. 8]. We will omit a discussion of the speech present probability estimator and just be aware that it is commonly part of a speech enhancement system. Finally, it should be noted that there in general is a trade off between noise suppression and speech distortion in the speech enhancement algorithms [Loizou, 2007, p. 107]

### 3.1.1 Spectral Subtraction

One of the most well known algorithms for noise suppression in speech processing is based on spectral subtraction. The basic idea is to make an estimate of $N(n,\omega)$ and then simply subtract it from $Y(n,\omega)$ to obtain an estimate of $X(n,\omega)$. The estimates of $X(n,\omega)$ and $N(n,\omega)$ are denoted $\hat{X}(n,\omega)$ and $\hat{N}(n,\omega)$, respectively. Now $\hat{N}(n,\omega)$ can be used to obtain $\hat{X}(n,\omega)$ as

$$\hat{X}(n,\omega) = Y(n,\omega) - \hat{N}(n,\omega). \tag{3.3}$$

Alternatively the T-F representation of the noisy speech can be expressed in polar form as

$$Y(n,\omega) = |Y(n,\omega)|e^{j\phi_y(n,\omega)} \tag{3.4}$$

where $|Y(n,\omega)|$ is the magnitude and $\phi_y(n,\omega)$ is the phase of the T-F representation. The T-F representation of the noise can likewise be expressed in polar form as

$$N(n,\omega) = |N(n,\omega)|e^{j\phi_n(n,\omega)} \tag{3.5}$$

where $|N(n,\omega)|$ is the magnitude and $\phi_n(n,\omega)$ is the phase of the T-F representation. It has been shown that in practical applications the noisy phase can be used as a decent estimate for the noise-free phase meaning that $\phi_y(n,\omega) \approx \phi_n(n,\omega)$ [Deller et al., 1993, p. 508]. Using this approximation, Eq.(3.3) can be rewritten as

$$\hat{X}(n,\omega) = (|Y(n,\omega)| - |\hat{N}(n,\omega)|)e^{j\phi_y(n,\omega)}. \tag{3.6}$$

This implies that what have to be estimated is the T-F representation of the noise $|\hat{N}(n,\omega)|$. The spectral subtraction calculation in Eq. (3.6) can also be performed on the PSD as [Loizou, 2007, p. 100]

$$|\hat{X}(n,\omega)|^2 = |Y(n,\omega)|^2 - |\hat{N}(n,\omega)|^2. \tag{3.7}$$

Hereby an enhanced speech signal can be obtained by applying the inverse STFT to Eq. (3.6). One of the major drawbacks of spectral subtraction is the hazard of introducing distortion in the speech signal. When too much is subtracted from a noisy T-F tile, speech is lost and if to little is subtracted noise is still present. Another thing to notice is that when one performs spectral subtraction as in Eq. (3.6) the result can become negative due to estimation error. Since the magnitude spectrum cannot be negative this result is meaningless. A simple way to overcome this is to set $|\hat{X}(n,\omega)|$ to zero, if the calculation become negative. However, this nonlinear approach causes isolated peaks in the spectrum and within subsequently frames, these peaks will occur in different locations of the spectrum. In the time domain this will result in random variations of tones at the frame rate. This generated distortion is referred to as "musical noise" [Loizou, 2007, p. 110]. An example of how the spectrogram is affected by "musical noise", is illustrated in Fig. 3.1. Here a clean speech signal is corrupted by additive white Gaussian noise with a SNR of 5 dB. The top panel shows the spectrogram of the clean signal, the middle panel shows the spectrogram of the noisy signal and the bottom panel shows the spectrogram of the enhanced signal. From the bottom panel the isolated peaks causing the "musical noise" are clearly visible. Furthermore, it is seen that the algorithm also removes part of the clean signal. An audio file with the enhanced signal is available from the file *enhanced_sig_ssub.wav* on the appendix DVDs. The implementation of the Spectral Subtraction algorithm is made by Philipos C. Loizou and the code is available from [Loizou, 2007].

When the SNR decreases the effect of the "musical noise" will be more noticeable. So even though the spectral subtraction approach is relative simple to interpret and implement it has its drawbacks of introducing "musical noise" and distortion. To avoid the effect of "musical noise" different methods have been proposed [Loizou, 2007, p. 112-136]. An investigation of such methods is, however beyond the scope of this thesis.

### 3.1.2 Wiener Filter

The spectral subtraction method discussed in Section 3.1.1 is not derived in a theoretical optimal way. Therefore, the attention is turned toward the Wiener filter which is optimal in a mean-square error sense when certain statistical assumptions, that will be clarified later, is fulfilled. In the following, the Wiener filter model is described together with its asymptotic SNR behavior. Next, the Wiener filter is formulated in the context of speech enhancement. The aim here is to attenuate the additive noise as described from the model in Eq. (3.1). Finally, the wiener filter is formulated in the STFT domain.

The Wiener filter is constrained to be of a linear type and could be a Finite Impulse Response (FIR) as well as an IIR filter. It is however, normal procedure to consider the FIR filter, since it is inherently stable and the solution is relative easy to evaluate. Therefore, only the FIR filter is considered in the following. The setup of a linear filtering problem is depicted in Fig. 3.2. The assumptions made, when

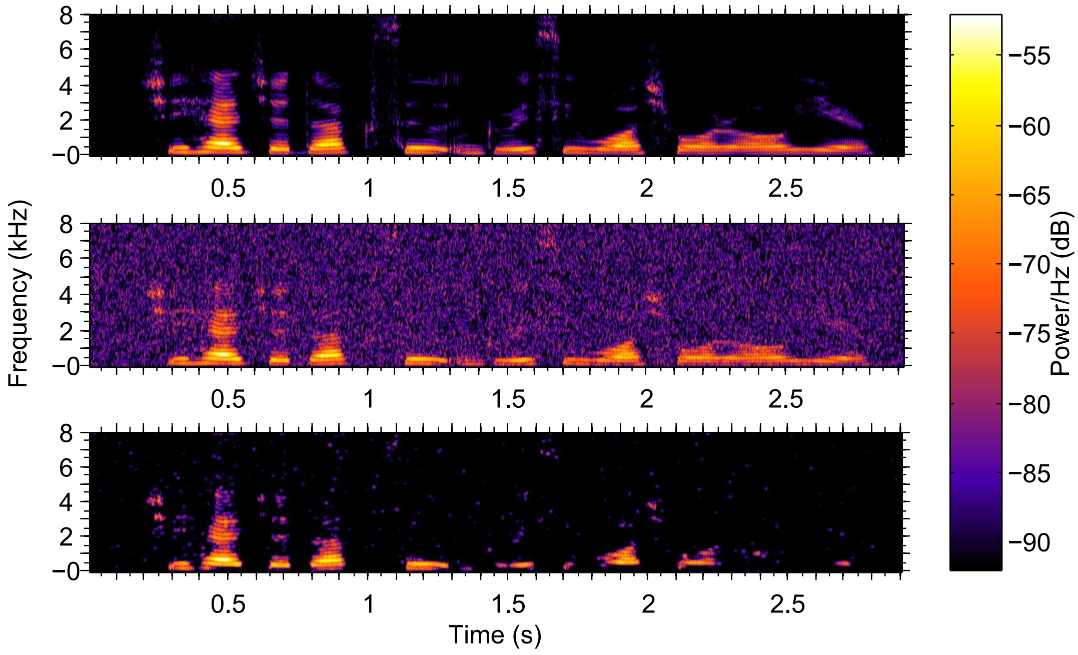


**Figure 3.1:** Example of random locations of peaks in the spectrogram during spectral subtraction. Top figure shows the spectrogram of the noise-free signal, middle figure shows the spectrogram of the noisy signal and bottom figure shows spectrogram of enhanced signal. The "musical noise" is clearly illustrated by the isolated peaks in the bottom figure.

deriving the Wiener filter, is that the input signal $y(n)$ and desired signal $d(n)$ are assumed to be realizations of a jointly WSS process. This means that both $y(n)$ and $d(n)$ are realizations of a WSS stochastic process and their cross-correlation depends only on the time lag.

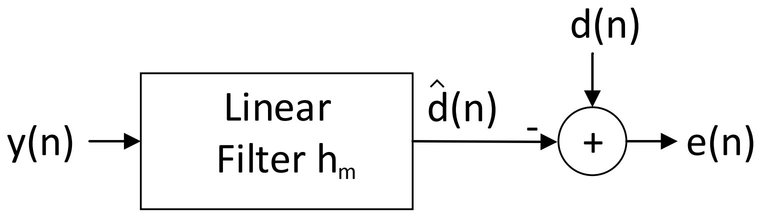


**Figure 3.2:** This figure presents the linear model, which forms the foundation of the Wiener filter.

The filter from Fig. 3.2 is a $M$-tap FIR hence a $M$ length delay line of the input signal is used. The signals included in the Wiener filter can be described as

- $\mathbf{h} = [h_0 \;\; h_1 \ldots h_{M-1}]^T$.
- $\mathbf{y}(n) = [y(n) \;\; y(n-1) \ldots y(n-M-1)]^T$.

Here $\mathbf{h}$ is the filter coefficients and $\mathbf{y}(n)$ is the WSS input signals. The remaining signals in the model shown in Fig. 3.2 can be defined as follows

- $\hat{d}(n) = \mathbf{y}(n)^T\mathbf{h}$ is the output signal.
- $d(n)$ is a zero mean WSS desired signal.
- $e(n) = d(n) - \hat{d}(n)$ is the error signal.

A derivation of the Wiener filter, using the setup in Fig. 3.2, can be found in Appendix A.

Since the Wiener filter is to be used for speech enhancement, the model for a speech signal corrupted by additive noise given in Eq. (3.1) is used as input. Therefore, the desired signal is the clean signal, i.e. $d(n) = x(n)$. In this case, and under the assumption that the noise is zero mean and uncorrelated with the clean signal, the Wiener filter can be expressed as [Loizou, 2013, p. 145]

$$\mathbf{h} = (\mathbf{R}_{xx} + \mathbf{R}_{nn})^{-1}\mathbf{r}_{xx} \tag{3.8}$$

where $\mathbf{R}_{xx}$ is the auto-correlation matrix of the speech signal, $\mathbf{R}_{nn}$ is the auto-correlation matrix of the noise signal and $\mathbf{r}_{xx}$ is the cross-correlation vector.

The solution to the filter coefficients can be rewritten as [Chen et al., 2006, p. 1220]

$$\mathbf{h} = \left[\frac{\mathbf{I}}{SNR} + \hat{R}_{nn}^{-1}\hat{R}_{xx}\right]^{-1} \hat{R}_{nn}^{-1}\hat{R}_{xx}\mathbf{u}_1 \tag{3.9}$$

where the SNR is given by $\sigma_x^2/\sigma_n^2$, $\hat{\mathbf{R}}_{nn} \triangleq \mathbf{R}_{nn}\sigma_n^2$, $\hat{\mathbf{R}}_{xx} \triangleq \mathbf{R}_{xx}/\sigma_x^2$, $\mathbf{I}$ is a $M \times M$ identity matrix and $\mathbf{u}_1$ is the first column of $\mathbf{I}$. From the expression in Eq. (3.9) the asymptotic behavior of the Wiener filter, when the SNR tends to either zero or infinity, can be written as [Chen et al., 2006, p. 1220]

$$\lim_{\text{SNR}\to 0} \mathbf{h} = \mathbf{0} \tag{3.10}$$

and

$$\lim_{\text{SNR}\to \infty} \mathbf{h} = \mathbf{u}_1 \tag{3.11}$$

The interpretation of Eq. (3.11) is that when the SNR tend to a very high level the Wiener filter does not provide any attenuation of the noise since $\mathbf{u}_1^T\mathbf{y} = y(n)$. On the other hand when the SNR tends towards zero the output is highly attenuated resulting in distortion of the speech signal as seen from Eq. (3.10). From Eq. (3.9) it is seen that the Wiener filter coefficient are dependent on $\mathbf{R}_{xx}$ and $\mathbf{R}_{nn}$ which are not directly observable.

Until now the Wiener filter has been defined in the time domain. Most often it is though implemented in the STFT domain so the filter is also presented in this domain. The T-F representation of the Wiener filter can be expressed as [Loizou, 2007, p. 152]

$$H(n,\omega) = \frac{P_{xx}(n,\omega)}{P_{xx}(n,\omega) + P_{nn}(n,\omega)} \tag{3.12}$$

where $P_{xx}(n,\omega)$ and $P_{nn}(n,\omega)$ is the PSD of the clean and noise signal, respectively. Furthermore, the *a priori* SNR that can be defined as [Loizou, 2007, p. 152]

$$\xi(n,\omega) = \frac{P_{xx}(n,\omega)}{P_{nn}(n,\omega)} \tag{3.13}$$

and the Wiener filter can be expressed in terms of an *a priori* SNR

$$H(n,\omega) = \frac{\xi(n,\omega)}{\xi(n,\omega)+1}. \tag{3.14}$$

The T-F representation of the output of the filter is given by

$$\hat{X}(n,\omega) = H(n,\omega)Y(n,\omega) = \frac{\xi(n,\omega)}{\xi(n,\omega)+1}Y(n,\omega). \tag{3.15}$$

A T-F representation of a Wiener filter implementation based on the *a priori* SNR is shown in figure 3.3. This implementation is made by Philipos C. Loizou [Loizou, 2007]. A clean speech signal is corrupted by additive white Gaussian noise with a SNR of 5 dB. The top panel shows the spectrogram of the clean signal, the middle panel shows the spectrogram of the noisy signal and the bottom panel shows the spectrogram of the enhanced signal. Compared to the spectral subtraction method shown in figure 3.1, the Wiener filter still suffers from "musical noise" but more of the clean signal is preserved. An audio file with the enhanced signal is available from the file *enhanced_sig_wiener.wav* on the appendix DVDs.

### 3.1.3 Non-Linear Estimators

The Wiener filter presented in the previous section is a linear estimator since there is a linear relationship between the noisy PSD and the estimate of the clean PSD as shown in Eq. (3.15). Non-linear estimators could potentially improve the performance [Loizou, 2007, p. 209]. A popular choice of such estimator could be the Maximum Likelihood (ML) estimator. Here we have a frame of observations $\mathbf{y} = [y(0)\ y(1) \dots y(N-1)]$ and some unknown but deterministic parameters of interest given by $\boldsymbol{\theta}$. In the context of speech enhancement $\mathbf{y}$ could be a set of noisy observations and $\boldsymbol{\theta}$ could be the clean speech PSD. It is assumed that the Probability Density Function (PDF) $p(\mathbf{y};\boldsymbol{\theta})$ is known and this function is called the likelihood function. The aim is to find the $\boldsymbol{\theta}$, that most likely gave rise to the observations $\mathbf{y}$. This can be formulated as

$$\hat{\boldsymbol{\theta}}_{ML} = \arg\max_{\boldsymbol{\theta}} \quad p(\mathbf{y};\boldsymbol{\theta}) \tag{3.16}$$

where $\hat{\boldsymbol{\theta}}_{ML}$ is called the maximum likelihood estimate. In maximum likelihood estimation it is important to notice that the parameter of interest is deterministic but unknown. The method of ML is also used during the training of a DNN described in Chapter 5.

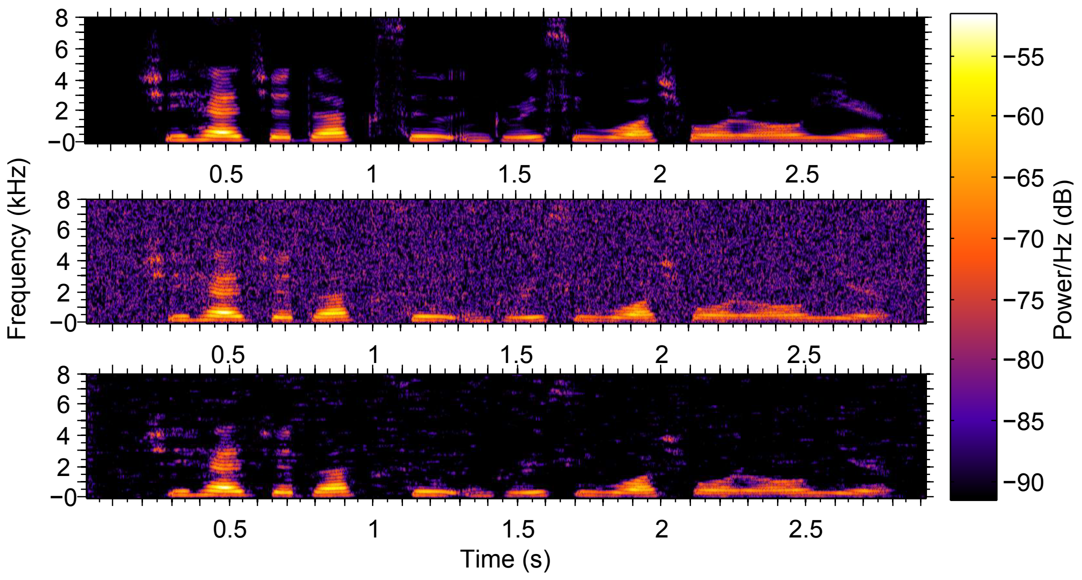


**Figure 3.3:** Speech enhancement performed by Wiener filtering. Top figure shows the spectrogram of the noise-free signal, middle figure shows the spectrogram of the noisy signal and bottom figure shows spectrogram of enhanced signal. Musical noise i still present in the enhanced signal but more of the clean signal is preserved, compared to the spectral subtraction method.

An alternative approach is to perform the estimation in a Bayesian framework. The Bayesian approach considers the parameter of interest as a random variable and hereby prior information can be considered. This is the fundamental difference between classical and Bayesian statistics. One such Bayesian estimator is the Maximum a Posteriori (MAP) estimator which in general can be formulated as

$$\hat{\boldsymbol{\theta}}_{MAP} = \arg\max_{\boldsymbol{\theta}} \quad p(\mathbf{y}|\boldsymbol{\theta})p(\boldsymbol{\theta}) \tag{3.17}$$

Even tough, ML and MAP estimators are popular within estimation theory and leads to relative simple expressions, the attention is turned towards another type of Bayesian estimators called MMSE estimators, since they often have a better performance [Loizou, 2007, p. 219] [Hendriks et al., 2013, p. 18]. MMSE estimators use the mean square error cost function and is expressed as the conditional mean of the parameters of interest given the observations. This can be written as

$$\hat{\boldsymbol{\theta}}_{MMSE} = \mathbb{E}[\boldsymbol{\theta}|\mathbf{y}] = \int \boldsymbol{\theta} p(\boldsymbol{\theta}|\mathbf{y})d\theta. \tag{3.18}$$

In terms of speech enhancement, this type of estimator can in general be expressed as a gain function taking the arguments of an *a posteriori* SNR $\gamma(n,\omega)$ and an *a priori* SNR $\xi(n,\omega)$, applied to the noisy STFT coefficients $Y(n,\omega)$. The *a priori* SNR $\xi(n,\omega)$ is already presented in Eq. (3.13) and the *a posteriori* SNR $\gamma(n,\omega)$ is defined as [Loizou, 2007, p. 224]

$$\gamma(n,\omega) = \frac{|Y(n,\omega)|^2}{P_{nn}(n,\omega)}. \tag{3.19}$$

For simplicity $\xi(n,\omega)$ and $\gamma(n,\omega)$ will be denoted $\xi_{n\omega}$ and $\gamma_{n\omega}$, respectively. The MMSE magnitude estimator is in [Loizou, 2007, p. 225] defined as

$$\hat{X}(n,\omega) = G(\xi_{n\omega}, \gamma_{n\omega})Y(n,\omega). \tag{3.20}$$

Under the assumption that speech and noise STFT coefficients are uncorrelated and both are Gaussian distributed a gain function can be defined as [Loizou, 2007, p. 225]

$$G(\xi_{n\omega}, \gamma_{n\omega}) = \frac{\sqrt{\pi}}{2}\frac{\sqrt{v_{n\omega}}}{\gamma_{n\omega}}\exp\left(-\frac{v_{n\omega}}{2}\right)\left[(1+v_{n\omega})I_0\left(\frac{v_{n\omega}}{2}\right) + v_{n\omega}I_1\left(\frac{v_{n\omega}}{2}\right)\right] \tag{3.21}$$

and $v_{n\omega}$ is defined as

$$v_{n\omega} = \frac{\xi_{n\omega}}{1+\xi_{n\omega}}\gamma_{n\omega}. \tag{3.22}$$

Furthermore, $I_0$ and $I_1$ is the modified Bessel function of zero and first order [Loizou, 2007, p. 581]. The gain function will not be subject to further investigation in this thesis but worth noting is that $G(\xi_{n\omega}, \gamma_{n\omega})$ is a function of the *a priori* and *a posteriori* SNR. Other gain functions have been proposed under different assumptions. For example, estimators derived under the assumption that STFT coefficients are super-Gaussian (sharper peak and longer tails) distributed tends to perform better than the estimators that assumes STFT coefficients to be Gaussian distributed [Hendriks et al., 2013, p. 21]. When implementing an MMSE estimator the following steps should be involved.

- Calculate the T-F representation of the noisy signal $Y(n,\omega)$.
- Calculate the *a posteriori* SNR $\gamma(n,\omega)$ from Eq. (3.19).
- Calculate *a priori* SNR $\xi(n,\omega)$ from Eq. (3.13).
- Calculate the enhanced T-F representation from the noisy signal using the gain function defined in Eq. (3.21).
- Reconstruct the time domain representation of the enhanced signal.

A spectrogram representation of a non-linear MMSE implementation, based on the gain function given in Eq. (3.21), is shown in figure 3.4. This implementation is made by Philipos C. Loizou [Loizou, 2007]. Here a clean speech signal is corrupted by additive white Gaussian noise with a SNR of 5 dB. The top panel shows the spectrogram of the clean signal, the middle panel shows the spectrogram of the noisy signal and the bottom panel shows the spectrogram of the enhanced signal. Compared to the Wiener filter it can be seen that the "musical noise" effect is considerable reduced. It can also be seen that most of the clean speech signal seems to be preserved. An audio file with the enhanced signal is available from the file *enhanced_sigMMSE.wav* on the appendix DVDs.

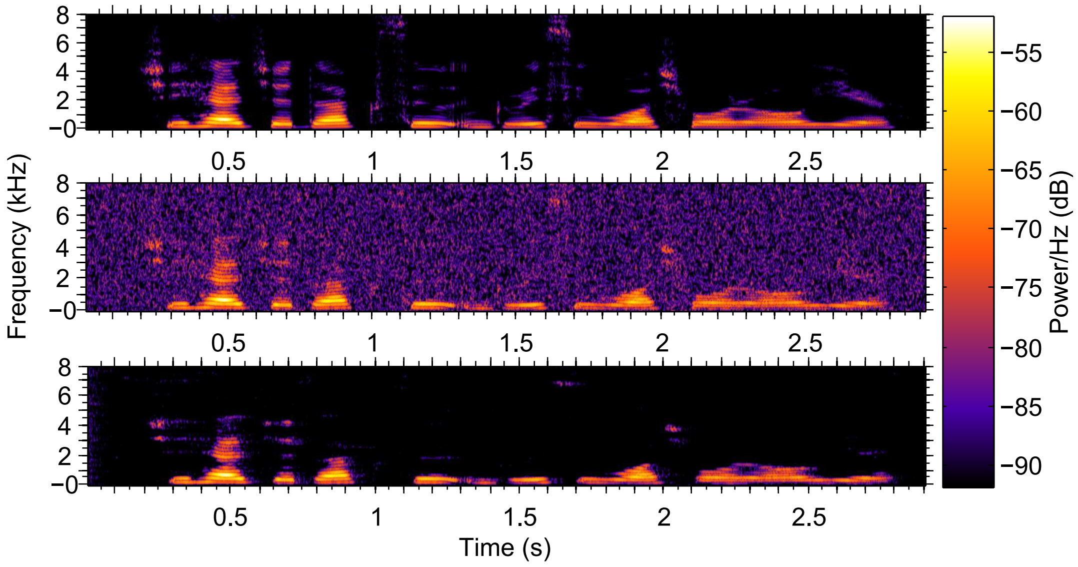


**Figure 3.4:** Speech enhancement performed by spectral magnitude MMSE estimation. Top figure shows the spectrogram of the noise-free signal, middle figure shows the spectrogram of the noisy signal and bottom figure shows spectrogram of enhanced signal. Musical noise is considerably reduced in the enhanced signal and most of the clean signal is preserved.

For the method mentioned so far, it is necessary to have access to both the noise and clean PSD, in isolation. This is obviously not the case in practical speech enhancement scenarios, where only $y(n)$ is observed, so these quantities have to be estimated. The next section will give a brief introduction to some principles on how noise and speech PSD estimators can be designed.

### 3.1.4 Speech and Noise PSD Estimation

Speech and noise PSD estimation is an essential part of a the speech enhancement algorithms presented so far in this chapter. As seen so far the methods depend on the clean spectrogram and (or) the noise spectrogram. The estimates of these quantities will therefore have a major impact on the performance of the different speech enhancement algorithms. Many types of both noise and speech PSD estimators exist. The above discussion should not be seen as an overview of PSD estimators. It should just stress the fact that the aforementioned speech enhancement systems highly depend on the performance of such PSD estimators. In terms of noise estimation, a simple approach is to estimate the noise spectrogram during "silence" periods of the signal, where speech signal is absent and only the noise signal is present. This can be done by use of a Voice Activity Detection (VAD) algorithm that typically makes a binary decision, based on a time frame, whether speech is present or absent. Such methods may work well in an environment with stationary noise. In non-stationary environments, noise estimation based on VAD would not be sufficient, since the noise spectrum changes over time, also during speech activity. Other types of noise estimation is therefore often preferred and VAD algorithms will not be subject to any further investigation in this thesis.

One observation that can be exploited when estimating the noise PSD is that even during speech activity, the energy of the speech is not guaranteed to be present in every frequency band [Hendriks et al., 2013, p. 30]. This also implies that the power of the noisy signal decays to the noise power in the individual frequency bands[Loizou, 2007, p.379]. One can therefore obtain an estimate of the noise PSD by tracking the minimum power in each T-F unit of the noisy signal over a fixed time interval. This type of noise estimation is referred to as minimal-tracking algorithms.

When estimating the speech PSD this can be done by use of maximum likelihood estimation. Here the likelihood function is given by $p(\mathbf{y}; P_{xx})$ where a frame of noisy observations $\mathbf{y}$ is given and the speech PSD have to be estimated. If assumed that the spectral coefficients of the speech and noise signals are complex Gaussian distributed the maximum likelihood estimator is given by [Hendriks et al., 2013, p. 37]

$$\hat{P}_{xx}(n, \omega)_{ML} = |Y(n, \omega)|^2 - P_{nn}(n, \omega). \tag{3.23}$$

If the speech and noise are uncorrelated Eq. (3.23) will yield an unbiased estimate of the speech power. On the other hand this estimator will have a large variance since a sample size of one frame is used. As it is desired to find an estimator with a minimum variance this is obviously not desired. A way to overcome this is to perform smoothing of $L$ successive frames of the noisy signal. However, this requires that the noisy signal is stationary within the $L$ successive frames and this is often not the case in practice. A method based on a previous clean speech estimate is therefore considered. This is called the decision directed approach and is one of the most commonly used speech estimators. This estimator is given by [Hendriks et al., 2013, p. 37]

$$\hat{P}_{xx}(n, \omega) = \max\Big(\alpha \hat{P}_{xx}(n-1, \omega) + (1+\alpha)\Big(|Y(n, \omega)|^2 - P_{nn}(n, \omega)\Big), \beta P_{nn}(n, \omega)\Big) \tag{3.24}$$

where $\alpha$ and $\beta$ is trade off parameters for noise reduction and speech distortion.

## 3.2 Ideal Binary Mask Estimation

What will be discussed in this section is speech enhancement methods based on the concept of an Ideal Binary Mask (IBM). Already from this point it should be made clear that the IBM in itself is not a practical applicable speech enhancement method since it requires that one knows both the clean and the noise signal, in isolation. For practical speech enhancement the goal should be to make a decent estimate of the IBM.

The idea of the IBM arise from Albert Stanley Bregmans proposed model of the human auditory perception, called Auditory Scene Analysis (ASA) [Wang and Brown, 2006, p.2]. Bregman describes the human auditory system to consist of two stages. In the first stage a decomposition of the input signal into T-F units takes place. This is called the segmentation stage. In the second stage, called grouping, the different T-F unit that are likely to come from the same source are grouped. The T-F unit within the same group is collected into a perceptual stream. It is believed that humans, in this way, perform sound source segregation. The model has inspired research within the field of Computational Auditory Scene Analysis (CASA) where the goal is to extract such streams in a computationally manner [Wang and Brown, 2006]. In a CASA system the input signal is first transformed into a T-F representation. It is commonly done by sending a signal through a gammatone filter bank and then divide the output into frames. This T-F representation is referred to as a cochleagram which was explained in Section 2.3. Since the goal is to segregate a speech signal from a noise signal, the IBM has been proposed to decide whether a T-F unit in the cochleagram is dominated by the noise source or by speech. The IBM is given by

$$\mathrm{M}(n,\omega) = \begin{cases} 1, & \text{if } \frac{||x(n,\omega)||^2}{||n(n,\omega)||^2} > \theta \ . \\ 0, & \text{otherwise.} \end{cases} \tag{3.25}$$

where $||x(n,\omega)||^2$ is the energy in a speech T-F unit, $||n(n,\omega)||^2$ is the energy in a noise T-F unit and $\theta$ is a threshold value. Considerable improvements in speech intelligibility have been obtained by use of an IBM [Wang and Brown, 2006, p.23].

The interpretation of the IBM is that the decision rule in Eq.(3.25) should decide whether a given T-F unit is dominated by the speech signal or the noise signal according to some threshold. Hereby, the IBM can be used to eliminate noise dominated T-F units and preserve speech dominated T-F units. To generate the IBM it is of course necessary to have access to both the speech and noise signal in isolation, which is not the case in practical speech enhancement scenarios. Therefore, an estimate of the IBM should be obtained in some way and a binary classification problem is faced. There are several ways to solve such a problem including different machine learning methods. In recent work, the problem of estimating the IBM for speech enhancement is performed by use of Gaussian Mixture Models (GMMs) [Kim et al., 2009], Support Vector Machines (SVMs) [Hang and Wang, 2012] and DNNs [Healy et al., 2013]. These three papers report promising results in terms of classification performance and to gain insight into the three machine learning methods, a description is presented in the following.

### 3.2.1 Gaussian Mixture Models

An approach to handle the estimation of the IBM is in terms of Gaussian mixture models, as in [Kim et al., 2009]. Here it is assumed that two somewhat distinct clusters appear in some feature space, one for noise dominated T-F units and one for speech dominated T-F units. These two clusters can each be modeled as a mixture of Gaussians. In general a mixture of $K$ Gaussians on some variable $\mathbf{x}$ is given by

[Bishop, 2009, p.111]

$$p(\mathbf{x}) = \sum_{k=1}^{K} \pi_k \mathcal{N}(\mathbf{x}|\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k) \tag{3.26}$$

where $\boldsymbol{\mu}_k$ is the mean vector and $\boldsymbol{\Sigma}_k$ is the covariance matrix for the $k^{th}$ Gaussian component. The parameter $\pi_k$ is called a mixing coefficient, and is the probability that a sample originate from the $k^{th}$ Gaussian component. The approach presented in [Kim et al., 2009] is based on training two different mixture models, each consisting of a 256-mixture Gaussian model. One model for noise dominated T-F units and one for speech dominated T-F units. In the training phase both clean and noise signals are available, in isolation. In the test phase a binary decision rule will, based on the probabilities of the two different mixture models, classify T-F units as noise dominated or speech dominated. An often used method for training GMMs is the Expectation-Maximization (EM) algorithm that will give the maximum likelihood estimate of the parameters. This is also the approach used in [Kim et al., 2009]. The EM algorithm is carried out through the use of a latent variable. The derivation of the Gaussian mixture model expressed in terms of such latent variable is described in Appendix B, together with the EM algorithm.

Commonly the K-means algorithm is used to initiate the parameters of the Gaussian mixture model and this is also the case in [Kim et al., 2009]. The results from [Kim et al., 2009] reveals that the GMM based method gave rise to a large intelligibility improvements, especially at low SNR levels.

### 3.2.2 Support Vector Machines

The problem of estimating the IBM can also be performed through the use of SVMs as in [Hang and Wang, 2012]. The SVM is as a binary classifier with a supervised learning approach. When used for estimating the IBM the two classes correspond to noise dominated T-F unit and speech dominated T-F units. It is for now assumed that the two classes are linear separable in some feature space and there will exist a decision hyperplane that separate those two classes. One such hyperplane is not unique but the aim of the SVM is to find the hyperplane with the maximum margin. This is illustrated in Fig. 3.5.

From figure 3.5 it is seen that three hyperplanes are defined. The hyperplanes $H_1$ and $H_2$ contains the support vectors which are the data points closest to the decision hyperplane $H$. Furthermore, the vector $\mathbf{w}$ is perpendicular to the hyperplanes. During the training phase, the training data for the SVM is given by $\{\mathbf{x}_i, y_i\}$ where $y_i \in \{-1, +1\}$, $\mathbf{x}_i \in \mathbb{R}^{\mathrm{d}}$. This means that $y_i$ is one of the two class labels denoted -1 and +1 and $\mathbf{x}_i$ is a sample from some feature space. Since all samples should belong to one of the two classes the following must be satisfied

$$\mathbf{x}_i^T \mathbf{w} + b \geq +1 \quad \text{for } \; y_i = +1 \tag{3.27}$$

$$\mathbf{x}_i^T \mathbf{w} + b \leq -1 \quad \text{for } \; y_i = -1 \tag{3.28}$$

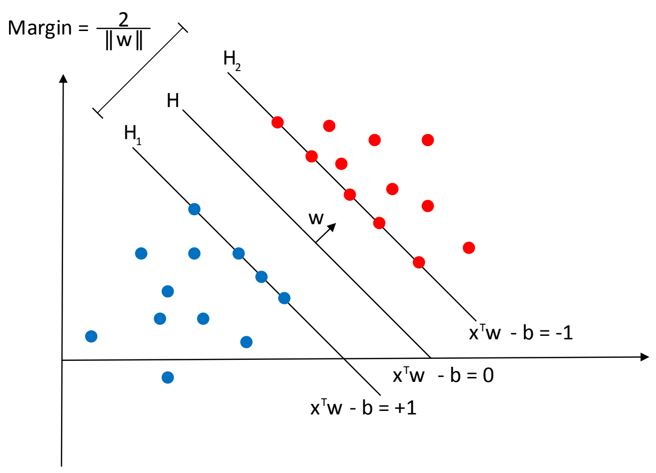


**Figure 3.5:** The SVM illustrated in a linear separable feature space. The blue and red dots illustrate data from the two classes and the data points located on the hyperplanes $H_1$ and $H_2$ are called support vectors.

where b is a bias term. If Eq. (3.27) and (3.28) is combined the following expression is obtained

$$y_i(\mathbf{x}_i^T \mathbf{w} + b) - 1 \geq 0. \tag{3.29}$$

This constraint should therefore be fulfilled for all $i$ when one is to maximize the margin. The margin is found to be $2/\|w\|$ and since the objective is to maximize this margin it is equivalent to minimizing $\|w\|$. This can be formulated as the optimization problem

$$\begin{aligned} &\underset{\mathbf{w},b}{\arg\min} && \tfrac{1}{2}\|\mathbf{w}\|^2 \\ &\text{subject to} && y_i(\mathbf{x}_i^T \mathbf{w} + b) - 1 \geq 0 \end{aligned} \tag{3.30}$$

Hereby a trained model can be used and the decision function is given by [Bishop, 2009, p. 329]

$$f(\mathbf{x}) = \sum_{i=1}^{N} \alpha_i y_i \mathbf{x}_i^T \mathbf{x} + b \tag{3.31}$$

where there is a $\alpha_i$ for each training point and $\alpha_i > 0$ correspond to the support vectors. For other points, the value of $\alpha_i$ is 0. In order to classify test data the sign of the decision function is evaluated.

Until now it has been assumed that the data is linear separable in feature space. If this is not the case, a kernel trick can be applied to map $x_i$ into a higher dimensional feature space. This means that $\mathbf{x}_i^T\mathbf{x}$ can be substituted by some non-linear kernel function. There exist different such kernel functions and popular choices are the polynomial and radial basis kernel function. One drawback of the SVM is its relative long training time where one is faced with the curse of sample size. The use of SVMs for classification of the IBM discussed in [Hang and Wang, 2012] shows an HIT minus

False Alarm (HIT-FA) improvement compared to the GMM approach described in Section 3.2.1.

### 3.2.3 Deep Neural Networks

The methods of estimating the IBM can also be carried out by using a DNN as in [Healy et al., 2013]. This method has shown very promising results as it, in terms of HIT-FA rates, outperforms both the GMM and SVM approaches discussed so far.

As mentioned in the introduction in Chapter 1 the scope of this thesis is to investigate the performance of such methods. A detailed description of DNNs is hereby given in Chapters 4 and 5.

## 3.3 Performance Measures

In order to evaluate the performance of the different speech enhancement algorithms, and compare them against each other, it must be decided which performance measure to use. In the speech enhancement literature the criteria of quality and intelligibility are typically used. Quality refers to how "good" or "poor" a given speech signal is perceived and is highly subjective since different people have different opinions on what a "good" and "poor" speech signal sounds like. Intelligibility on the other hand is objective in nature since it relates to how well the information in the speech signal is perceived by the listener [Loizou, 2013, 439]. Measuring speech quality or speech intelligibility is ideally done using subjective listening test using a representative collection of test subjects, but unfortunately such listening test are both time consuming and expensive. For this reason researchers have investigated objective evaluation methods with the aim of being highly correlated with the real life listening tests.

Many such methods exist, for both speech quality and intelligibility, but the Perceptual Evaluation of Speech Quality (PESQ) [Rix et al., 2001] and the Short-Time Objective Intelligibility (STOI) [Taal et al., 2010] measures are two commonly used objective performance measures used for assessing quality and intelligibility, respectively.

For evaluating performance of binary classifiers, the classification accuracy is the intuitive measure to consider but a measure known as HIT minus False Alarm (HIT-FA) is commonly used for IBM based speech enhancement methods. The performance measures PESQ, STOI and HIT-FA are further described in the following section.

### 3.3.1 Perceptual Evaluation of Speech Quality (PESQ)

The PESQ measure is a performance measure originally invented to evaluate the degradation of speech quality when a speech signal has been sent through a telecommunication network. The computation of the PESQ measure requires the clean speech signal and the degraded speech signal and the algorithm accounts for time alignment and gain equalization and is based on a spectral comparison of the clean and degraded speech signal where perceived loudness and masking effects are considered [Loizou, 2013, p. 491]. The PESQ measure has been found to be highly correlated to subjective listening tests [Rix et al., 2001]. The implementation of PESQ used in this thesis is available from [Loizou, 2013].

### 3.3.2 Short-Time Objective Intelligibility (STOI)

The STOI measure is an Objective Intelligibility Measure (OIM), which is designed to be used in conjunction with T-F weighted speech, which encompasses IBM and other processing types [Taal et al., 2010]. The STOI measure is designed from the assumption that intelligibility is correlated with the similarity between neighboring T-F units of clean speech and T-F units of processed noisy speech. The computation of STOI can be divided into three different stages: T-F transformation, Intermediate intelligibility measure and the Final intelligibility measure.

The T-F transformation is performed by dividing the signal under examination into overlapping frames, which are hamming windowed. The frames are then appropriately zero-padded and transformed into frequency by a DFT. The frequency transformed frames are then divided into 15 one-third octave bands, which cover a frequency range from approximately 150 Hz to 3.8 kHz. STOI assumes a sampling frequency of 10 kHz so signals not fulfilling this requirements must be resampled.

The intermediate intelligibility measure is computed for each T-F unit and is defined as the linear correlation coefficient between consecutive T-F units within the same one-third octave band in the clean speech signal and the IBM processed noisy speech signal.

The final intelligibility measure is computed as the average between all intermediate intelligibility measures across time and frequency. This means that the STOI measure is a number between -1 and +1, with +1 being the highest intelligibility score. Studies have revealed that the STOI measure is highly correlated to human intelligibility supporting the use of STOI as an OIM [Taal et al., 2010] [Loizou, 2013, p. 566]. The implementation of STOI used in this thesis is available from [Taal, 2010].

### 3.3.3 Hit - False Alarm (HIT-FA)

An intuitive performance measure when dealing with classifiers is the classification accuracy computed as the ratio between the total number of correctly classified and incorrectly classified labels. This measure however does not differentiate between the different kind of classification errors, which, in the binary case, are the False Alarm (FA) and Miss Error (ME). The FA is defined as the percentage of the total number of noise dominated T-F units mistakenly classified as speech dominated and ME is defined as the percentage of the total number of speech dominated T-F units mistakenly classified as noise dominated. It has been shown that a FA error is more damaging regarding speech intelligibility than a ME, which lead to the HIT-FA performance measure. This measure is aimed at IBM based speech enhancement methods [Wang, 2008] [Loizou, 2013, p. 647]. The HIT rate is defined as the percentage of the total number of correctly classified speech dominated T-F units. The difference between the HIT rate and the FA rate, i.e. the HIT-FA rate, has been reported to be highly correlated with speech intelligibility [Loizou, 2013, p. 647].

# Chapter 4

# Neural Networks

This chapter presents the basic theory regarding Artificial Neural Networks (ANNs). ANNs form the foundation of the Deep Neural Network (DNN) methods used for speech enhancement in chapter 6. ANNs are a class of machine learning methods used for solving regression and classification problems and are motivated by research on the human brain. The way the human brain solves problems is very different from how a typical PC, based on a Von Neumann or Harvard architecture, solves problems. A typical silicon based PC consists of relatively few processing units and some memory and is capable of computing several hundred billion of Floating-point Operations Per Second (FLOPS) in a primarily sequential manner. The computations are executed based on machine instructions, which are representations of algorithms designed, by a human, to solve certain problems. From this perspective PCs are very well suited for solving well-defined problems which can be exactly described by an algorithm. This could be problems such as searching through a database or computing the constant $\pi$ with a billion digits. Problems which are fairly simple and well understood. Problems such as recognizing objects within an image or recognizing noisy speech are on the other hand rather complicated. Manually designing generic algorithms that effectively solves such problems are very difficult. The human brain on the other hand solves such problems both faster and better than computers are capable of today [Haykin, 1999, p. 1]. This sort of paradox and the aim of understanding human cognition lead to connectionism, which is a machine learning paradigm aiming to understand human intellectual abilities using ANNs. Connectionist research lead to the use of ANNs in engineering applications such as, pattern recognition, speech recognition and speech enhancement. [Stanford Encyclopedia, 2014].

## 4.1 Biological Neural Networks

To fully understand the motivation behind ANNs a basic understanding of the human brain, and especially Biological Neural Networks (BNNs), is required. The human brain weighs on average 1.375 kg and consists of a network of approximately $100 \times 10^9$ interconnected brain cells called neurons, hence the term Neural Network (NN) [Moos and Møller, 2010, p. 13]. A neuron consists primarily of three structures as illustrated in Fig. 4.1. The dendrites, the cell body and the axon. The dendrites serves as an

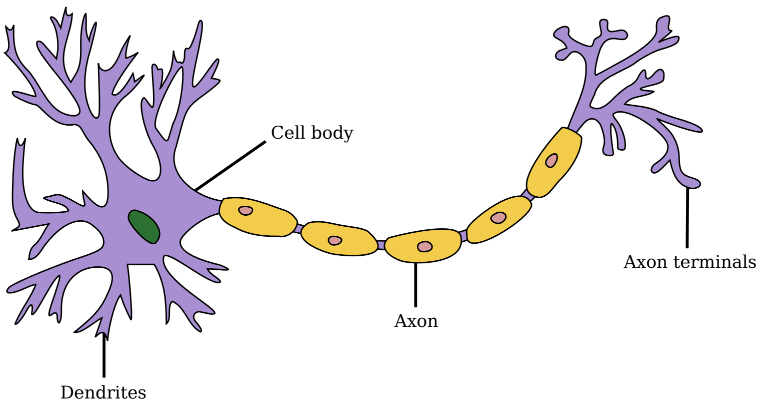


**Figure 4.1:** Biological neuron with its three primary components. The dendrites, the cell body and the axon with its axon terminals. The neuron receives signals on its dendrites which are accumulated in the cell body and if the signals accumulate above a certain threshold a new signal is fired onto the axon. Source: `http://commons.wikimedia.org/wiki/Neuron#mediaviewer/File\protect\kern+.2222em\relaxDerived_Neuron_schema_with_no_labels.svg` License: CC BY-SA 3.0

input leading chemical or electrical signals to the cell body and the axon serves as an output, leading signals away from the cell body [Moos and Møller, 2010, pp. 57-59]. The axon has several axon terminals which are connected to the dendrites on other neurons, hence forming the NN. The connections between neurons, i.e. connections between axon terminals and dendrites, are referred to as synapses. There can be from a few hundred to several hundred thousand synaptic connections between one neuron and other neurons in the NN. These synaptic connections make up an extremely large and complicated network with several trillion synaptic connections [Haykin, 1999, p. 6]. The neurons are also connected to other parts of the body such as the optic nerves involved in vision or the motor nerves involved in muscle contractions. These parts act, respectively as input and output for the NN.

The general perception is that the human brain cannot produce neurons so when the brain stops growing, a few years after birth, the brain starts to degenerate and it looses on average 85.000 neurons everyday. However, the organization of the neurons, i.e. their interconnections, are believed to change or at least have the capability of doing so. This is studied within the field of brain plasticity [Gougoux et al., 2009]. For example is it shown that the visual cortex on blind people are processing hearing signals in cooperation with the auditory cortex [Bach and Kercel, 2003]. The visual cortex is the part of the brain processing vision on normal people and the auditory cortex is normally the only place in the brain hearing is processed, so these results indicate that the visual cortex has learned itself to process hearing.

What task a certain part of a BNN can perform and how a signal propagates, does not only depend on the physical organization of neurons but also on the sensitivity of the individual neurons within the network. Whether or not a neuron "fires" a signal onto its axon and further into the NN, depends on the magnitude and the number of input signals on its dendrites. The threshold for firing a signal is not fixed and can be changed by learning. It has been shown that the more times a given input has been given to a certain area of a BNN the more sensitive the neurons become and hereby more likely to generate signals on its axons. If an input has been provided enough times the neurons have been so sensitive that activities inside the brain can activate the same neurons resulting in the same output as if the input was present. This phenomena is called neural facilitation and is involved in how the brain learns [Guyton and Hall, 2011, p. 545].

These findings indicate that BNNs have capabilities of learning and handling very complicated problems using a relatively simple structure consisting of billions of interconnected neurons. The idea behind the ANN is to mimic the BNN exploiting its problem solving capabilities.

## 4.2 Artificial Neural Networks

It has in the former section been found that the BNN has capabilities of learning and from learning, solving rather complicated tasks such as object and speech recognition. The capabilities of the BNN are what motivates the developing of an ANN such that these, in a similar way, can be trained to solve complex engineering problems. This section will introduce the basic theory behind the ANN and how it can be trained. ANNs can be used to solve both classification problems and regression problems but since this project is regarding classification of IBMs, ANNs will be presented with the aim of solving classification problems. The ANN consists of primarily two things. The artificial neuron and the network architecture are further described in the following.

### 4.2.1 The Artificial Neuron

The artificial neuron, henceforth referred as the neuron, can be divided into several components. The inputs, the weights, the accumulator, an activation function and some outputs [Haykin, 1999, pp. 10-11]. A generalized neuron is illustrated in Fig. 4.2. The inputs, denoted $x_{1,\dots,I}$, are typically either a feature vector, such as pixels or MFCCs, or outputs from other neurons in a network. The weights, $w_{1,\dots,I}$, are used to scale the contribution from the input or from other neurons and can be seen as a way of indicating the importance or relevance of individual features or neurons. The weights are sometimes referred to as synaptic weights since they model the sensitivity of the synaptic connection in a biological neuron. The accumulator accumulates all the weighted inputs to the neuron. This is done to acquire the total amount of input potential $z$, which has a direct influence on whether or not the current neuron will change state and propagate a signal to other neurons in the network. The bias function $\theta$ is used to make an affine transformation of the input to the activation function $\varphi$. The activation function is the mechanism that decides

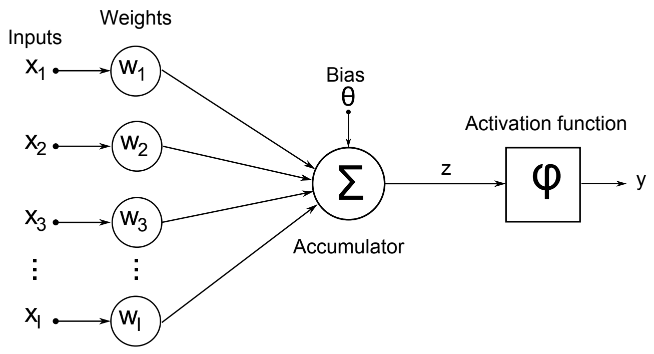


**Figure 4.2:** A block diagram of an artificial neuron with the inputs given by $x_{1,\dots,I}$ the weights given by $w_{1,\dots,I}$ and the activation function $\varphi$. The input to the activation function is given by $z$ and the output of the neuron is given by $y$.

whether or not the neuron should "fire" and if so, how large the magnitude of the signal should be. The accumulator, the bias and the activation function is a model of the cell body of a biological neuron. A mathematical description of the artificial neuron is shown in Eq. (4.1) and Eq. (4.2).

$$z = \sum_{j=0}^{N} w_j x_j \tag{4.1}$$

$$y = \varphi(z) \tag{4.2}$$

Equation (4.1) shows how the input potential is computed and Eq. (4.2) shows how the output is computed based on the input potential $z$ and the activation function $\varphi$. In Eq. (4.1) the bias term $\theta$ has been included in the summation by adding an extra input $x_0 = 1$ with a weight fixed at $w_0 = \theta$.

There exists five commonly used activation functions [Haykin, 1999, pp. 12-14]. A threshold or binary function, a piecewise linear function, the sigmoid function, the hyperbolic tangent function and the softmax function. The first four activation functions are illustrated in Fig. 4.3. The threshold or binary function is a discrete step function which changes state from either zero or positive one depending on the input value and a threshold value. The piecewise linear function is either zero or positive one outside some predefined interval and within the interval the function is linear. This means that the piecewise linear function can take any value between zero and positive one. The sigmoid function is a continuous, smooth and non-linear, but differentiable, function which takes any value from zero to positive one. The hyperbolic tangent function is also a continuous, smooth, non-linear and differentiable function but it takes on values in the interval from negative one to positive one. The softmax function is similar to the sigmoid function in that way that it has a continuous output from zero to one. The softmax differs from the sigmoid function by having normalized outputs with respect to all other outputs, such that the output of the neural network represents probabilities [Bishop, 2009, p. 236].

What activation function to use depends on the application but the sigmoid and softmax function are the most used. This is primarily due to its simple form and its derivative and because it leads to a distributed representation within the network, which for the latter implies robustness [Haykin, 1999, p. 14] [Duda et al., 2001, p. 307]. The mathematical description of the sigmoid function is given as

$$\varphi(z) = \frac{1}{1+e^{-z}}. \tag{4.3}$$

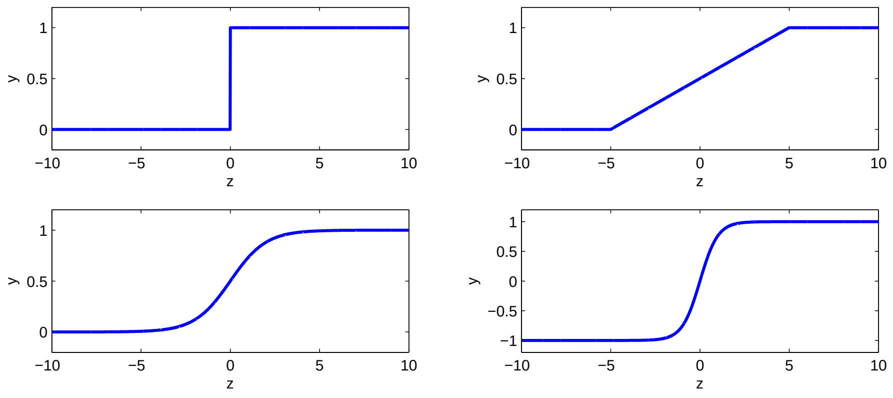


**Figure 4.3:** The four commonly used activation functions. Upper left: The binary activation function. Upper right: The piecewise linear activation function. Lower left: The sigmoid activation function. Lower right: The hyperbolic tangent activation function. The softmax activation function is not displayed since it is merely just a normalization.

### 4.2.2 Artificial Neural Network Architectures

When several neurons are connected together they form a network and three main network architectures exist [Haykin, 1999, pp. 21-29]. The FNN, the Recurrent Neural Network (RNN) and the Convolutional Neural Network (CNN). ANNs are typically represented as directed graphs and can be both cyclic or acyclic and fully and partially connected. The property of being fully connected are not meant as in the strict definition from graph theory. In the context of ANNs the property of being fully connected means that all inputs are connected to all units in the first hidden layer and all units in that layer are connected to all units in the next layer and so on. There are typically no connections between neurons in the same layer in ordinary ANNs. The simplest neural network and also the most used, is a FNN as illustrated by the directed acyclic and fully connected graph in Fig. 4.4 [Bishop, 2009, pp. 225-229]. In Fig. 4.4 a two layer FNN is presented. In general, the terminology for ANNs is that the layer of inputs is not counted since only layers where computing takes place are counted. The second layer and potentially all layers beyond, except the output layer, is referred to as the hidden layers. The neurons within a hidden

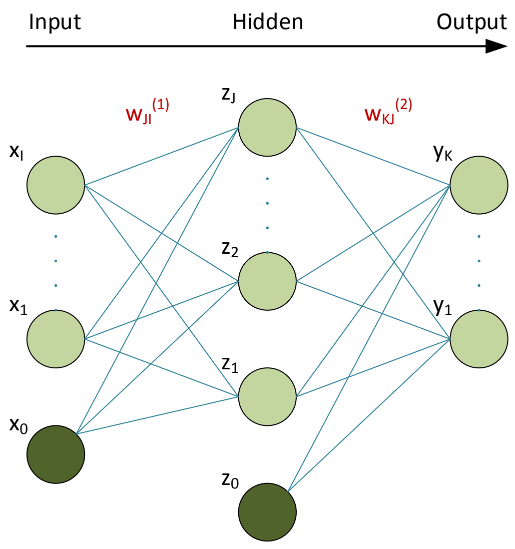


**Figure 4.4:** 2-Layer Feed-forward Neural Network with an input layer I consisting of I+1 input units. A hidden layer J consisting of J+1 hidden units and an output layer K consisting of K output units. The weights ${w_{JI}}^{(1)}$ correspond to the weight in layer one from input unit I to hidden unit J and ${w_{KJ}}^{(2)}$ correspond to the weights in layer two from hidden unit J to output unit K.

layer are referred to as hidden units. The term "hidden" comes from the fact that they are not being observed. The last layer in an ANN is referred to as the output layer and since the output layer consists of neurons, that layer is included in the number of layers. Therefore, the FNN illustrated in Fig. 4.4 has two layers. The FNN has $I + 1$ input units, $J + 1$ hidden units and $K$ output units. The +1 term in the input layer and the hidden layer are due to the bias terms which are connected to a fixed input of +1 illustrated by the dark green units on Fig. 4.4. The ${w_{JI}}^{(1)}$ and ${w_{KJ}}^{(2)}$ illustrated in red represents the weights and should be interpreted in the following way. The first subscript indicates the unit for which the weight belong and the second subscript indicate which node the weight is connected to. The superscript determines what layer the weights belong to. For example, ${w_{21}}^{(2)}$ is the weight on the arc between the first hidden unit and the second output unit in layer 2. The output from the $k^{th}$ output unit from the FNN in Fig. 4.4 is expressed mathematical as

$$y_k(\mathbf{x}, \mathbf{w}^{(1)}, \mathbf{w}^{(2)}) = \varphi_1\left(\sum_{j=0}^{J} w_{kj}^{(2)} \varphi_2\left(\sum_{i=0}^{I} w_{ji}^{(1)} x_i\right)\right) \tag{4.4}$$

where $y_k$ is the $k^{th}$ output, $\mathbf{x}$ and $\mathbf{w}$ are the input vector and weight matrix respectively, $\varphi_1$ and $\varphi_2$ are the activation functions and $w_{ji}^{(1)}$ and $w_{kj}^{(2)}$ are the weights for respectively layer one and two.

In a BNN it is found that the signals normally propagate in only one direction, i.e. from dendrite to axon [Guyton and Hall, 2011, p. 543]. These findings motivated the FNN architecture. However, it is also found that feed-back occurs in BNNs which inspired the RNN architecture [Haykin, 1999, p. 18] [Freeman, 1975, p. 272]. The RNN has feed-back elements implementing memory in the network. These networks are represented as directed cyclic fully or partially connected graphs. Also the CNN has been inspired from the BNN by the concept of receptive fields within the visual cortex [LISA lab, 2014, p. 45]. CNNs are found to be powerful to solve tasks such as object recognition since CNNs are less sensitive to certain transformations in the input pattern such as translations, scaling and rotations [Bishop, 1995, p. 267] [Duda et al., 2001, p. 326]. The CNN is called a CNN since the weights are shared between neurons. This means that the "activation potential" for a given neuron is a convolutional sum of the weights and the inputs to the neuron. The CNN is represented as a partially connected cyclic or acyclic graph.

### 4.2.3 Design of Artificial Neural Networks

Until now an ANN has been defined as a network of artificial neurons consisting of weights and activation functions. The challenge when designing an ANN is to identify the numerical values of the weights, the types of activation functions, the number of hidden units and the number of layers. A few things are given directly from the application. The number of inputs must match the length of the feature vector and the output must match the number of classes the network should classify, if a classification problem is solved. For a MNIST classification problem there could be 784 input units equal to a $28 \times 28 = 784$ pixels input vector and the ANN would have 10 output units corresponding to the 10 classes of digits from $0 - 9$.

Regarding the architecture there are no well defined rules. Whether it should be a FNN, a RNN or a CNN is primarily based on heuristics [Haykin, 1999, p. 28]. The same applies to the number of hidden units and the type of activation functions for which the optimal is unknown. However, it can be proven by Kolmogorov's theorem that any continuous function can be represented, with an arbitrary accuracy, by a 2-layer ANN, given a sufficient number of hidden units, weights and non-linearities [Bishop, 1995, p. 137] [Duda et al., 2001, p. 307]. This means that any continuous decision boundary can be represented arbitrarily well by an ANN. Unfortunately, the theorem requires a quadratic number of hidden units relative to the number of inputs and it does not tell what activation functions to use but it motivates that a 2-layer ANN, such as the one illustrated in Fig. 4.4, using non-linear sigmoid activation functions and a relatively large number of hidden units should be able to deliver a good starting point on an ANN design [Duda et al., 2001, p. 288].

### Hidden Units & Layers

Regarding the number of hidden units and the number of layers, no method for computing these quantities exists, and in practice it is often based on an educated guess and experimentation [Duda et al., 2001, p. 310] [Haykin, 1999, p. 191]. However, there is some intuition which can be used to get an idea of the consequences if the number is either too small or too large. The hidden layers in an ANN can be seen as feature extractors, which extract features from the input from the preceding layer. The input to the first hidden layer is a vector represented in the input space but the output of the hidden layers are represented in a feature space, which can be either smaller or larger than the input space. The more hidden units there are in a layer, the more complex decision boundaries the layers can construct, hence separating the features more effectively. If the class clusters in the feature space are well separated or evenly linearly separated, only a few hidden units are required since the decision boundary is relatively simple. If the clusters however are highly entangled a large amount of hidden units are required in order to represent the decision boundary properly.

Empirical knowledge shows that the number of hidden units also relates to the generalization capability via the size of the training set. It is known that if a large number of hidden units are used relative to the number of training cases there is a risk of overfitting, which results in poor generalization capabilities [Duda et al., 2001, p. 288]. It is also to be expected that too few hidden units will give poor performance since a too simple decision boundary is constructed. It is suggested that good generalization capabilities are achieved if the number of training examples $N$ is equal to the number of hidden units $w$ divided by the fraction of the classification errors $\epsilon$ as shown in Eq. (4.5) [Haykin, 1999, p. 208] [Duda et al., 2001, p. 310].

$$N = \frac{w}{\epsilon} \tag{4.5}$$

Equation (4.5) can be used to estimate the classification error or propose the number of hidden unit in a given ANN architecture based on the training data available. Equation (4.5) also indicates that using too many hidden units has a negative effect on the performance of the ANN. Another reason to keep the number of hidden units as low as possible is due to the computational and memory complexity of the ANN and the time it takes to train the network. In the Backpropagation algorithm presented in the next section it is shown that the learning is based on gradients of an error function taking the weight vector as input. If there are too many hidden units in the ANN there are plateaus in the error space which results in slow convergence of the learning algorithm [Duda et al., 2001, p. 313].

Another aspect in the architecture of the ANN is the number of hidden layers. As mentioned earlier Kolmogorov's theorem states that one hidden layer is sufficient to approximate any decision boundary. However, it is found that it is easier for an ANN with several hidden layers to learn advanced features compared to an ANN with a single hidden layer [Duda et al., 2001, p. 317] [Haykin, 1999, p. 212]. As mentioned, the hidden layers should be interpreted as feature extractors and in a multi hidden layer ANN, one hidden layer improves the features provided from the layer below. This means that the features from the first hidden layer are local compared to features from subsequent layers, which gradually becomes more global hence increasing the capability of class separation of the ANN. One disadvantage of using several hidden layers is that the ANN is more prone to be caught in local minima during training, which of course decreases the performance of the ANN [Bengio, 2009, p. 32].

### 4.2.4 Backpropagation Algorithm

The last thing needed in order to make an ANN practically applicable is to find suitable weights. The weights are the key component and they give the ANN the power of solving complex problems. The development of an efficient training algorithm for multilayer networks has been one of the big hurdles in the ANN community. Even though ANNs was described in the literature in 1943 it was not until 1986 an efficient learning algorithm, capable of learning multilayer FNN, were discovered [Haykin, 1999, pp. 38-43] [Rumelhart et al., 1986]. The algorithm was named the backpropagation algorithm and it is capable of finding suboptimal weights in multilayer FNN. The backpropagation algorithm described in this section is limited to FNN but can with modifications also be used to train CNNs and RNNs [Bishop, 2009, p. 269] [Haykin, 1999, p. 750].

The philosophy behind the backpropagation algorithm is that the FNN is trained to model a function which maps a known training input to a desired output. This form of learning is referred to as supervised learning and is the primary way shallow ANNs are trained. It is assumed that if enough training data is present and a proper FNN architecture is used, the weights in the network will learn the underlying statistics that generated the training data. In this way the network can generalize and provide a correct output based on an unknown input.

In order to train a FNN, a performance measure must be decided such that it can be measured how the performance of the FNN changes relative to the weights. Typical two such measures exist, the sum-of-squares measure and the cross-entropy measure. It can be shown that minimizing the sum-of-squares is equivalent to maximizing the likelihood when a regression problem, involving Gaussian distributed random variables, are solved. Likewise, it can be shown that minimizing the cross-entropy is equivalent to maximizing the likelihood when a binary classification problem is solved [Bishop, 2009, pp. 232-235]. These findings are supported by [Simard et al., 2003] and should be considered when deciding on the error function. When the error function $E(\mathbf{w})$ is decided, the optimal weights can theoretically be found by setting the gradient of the error function to zero and solve for $\mathbf{w}$. Unfortunately, this cannot be done in closed-form. Even if it could it would only give globally optimal weights if the error function was convex, which is typically not the case [Bishop, 2009, p. 237]. A typical way to approach this problem is to use iterative methods such as gradient descent which iteratively finds a set of sub-optimal weights using only the gradient of the error function. Training a single layer FNN using the gradient descent method has been known since the 1960s and is known as the delta rule [Haykin, 1999, p. 40]. However, training a multilayer FNN is a more difficult task and was subject to the "credit assignment problem" until 1986 when the backpropagation algorithm was discovered [Duda et al., 2001, p. 291]. The credit assignment problem refers to the task of assigning responsibility of an error on the output of a FNN to the weight of a hidden unit, which lies one or more layers below the output unit. This problem is elegantly solved by the backpropagation algorithm. A derivation of the backpropagation algorithm using sigmoid activation functions and the sum-of-squares error function, is presented in Appendix C.

Even though, the backpropagation algorithm solved the problem of making ANNs practical applicable there were still some limitations and practicalities which limited the potential of ordinary ANNs. These limitations motivated the work on DNNs, which will be presented in the next chapter.

# Chapter 5

# Pre-training Neural Networks

As mentioned in section 4.2 there were some impracticalities with FNNs that limited the complexity of the problems the FNN could successfully solve, when it was trained using the backpropagation algorithm. These impracticalities are described as follows. When a complex classification problem is to be solved the ANN architecture must be decided accordingly and as stated by Kolmogorov's theorem any decision boundary can be represented by a 2-layer FNN, which indicates that a good solution exists [Bishop, 1995, p. 137]. However, representing this decision boundary would require an exponential number of hidden units, with respect to the visible units, which could make the FNN non-practically applicable if the complexity of the problem is sufficiently high.

A way to circumvent this problem is to construct a Deep Neural Network (DNN) by increasing the depth of the network in order to gain a more compact representation of the decision boundary, hence reducing the number of hidden units [Bengio, 2009, p. 18]. However, this solution also has a drawback, because the backpropagation algorithm cannot efficiently train these DNNs if they are initialized with random weights [Bengio, 2009, p. 32]. It has been observed that the backpropagation algorithm has a tendency to get stuck in local minima such that a DNN trained with the backpropagation algorithm performing worse than a shallow FNN having 1 or 2 hidden layers [Bengio, 2009, p. 32].

Another impracticality is the aspect of supervised learning. When training an FNN with the backpropagation algorithm, labeled data is required since this is required in the computation of the error. This means that the designer of the network must construct these labels, which in practice can be a very difficult and time consuming task. Also, as illustrated by Eq. (4.5) good generalization is dependent on the amount of training data, so it is fair to say that the more training data the better. However, unlabeled data on the other hand is fairly easy to acquire since no manually classification process is required after the data has been recorded. And today, with YouTube and Google, there is easy access to enormous amount of unlabeled data. These impracticalities, motivated the work on machine learning methods involving unsupervised learning and a breakthrough within the field of DNNs was done in 2006 by Geoffrey Hinton et al. by introducing the Deep Belief Network (DBN) [Bengio, 2009, p. 6] [Hinton et al., 2006]. This breakthrough is what initiated the hype about Deep Learning as described in the introduction in Chapter 1. The DBN is a generative model, which in combination with a DNN has shown promising results solving classification problems as well as performing dimensionality reduction [Hinton et al., 2006] [Hinton and Salakhutdinov, 2006]. The main component of a DBN is the Restricted Boltzmann Machine (RBM) which, when stacked on top of each other, forms a DBN. The RBM is described further in the next couple of sections and the DBN as a whole, is described in Section 5.4.

## 5.1 Restricted Boltzmann Machines

The RBM is a generative model and is used to represent probability distributions and are often represented as a probabilistic graphical model as illustrated in Fig. 5.1. All the nodes in a graphical model are random variables and their dependence on each other is illustrated via arcs. Graphical models can be either directed or undirected. Directed graphical models are known as Bayesian networks and undirected graphical models are known as Markov Random Fields (MRFs) [Bishop, 2009] [Murphy, 2012]. The RBM is a special type of a broader class of MRFs known as Boltzmann Machines (BMs) [Bengio, 2009]. The BM distinct itself from RBMs by being a fully connected network in the correct sense meaning that all nodes are interconnected. The RBM is not fully connected and has a structure as illustrated by Fig. 5.1. The definition of a RBM is that it consists of two layers. One known as the visible layer and one known as the hidden layer. The visible layer consists of visible units $v_m$ and the hidden layer consists of hidden units $h_n$. A weight $w_{nm}$ is associated with all arcs in the network. Each visible unit and hidden unit is also associated with a bias term, which for simplicity reasons, is not illustrated on Fig. 5.1 .

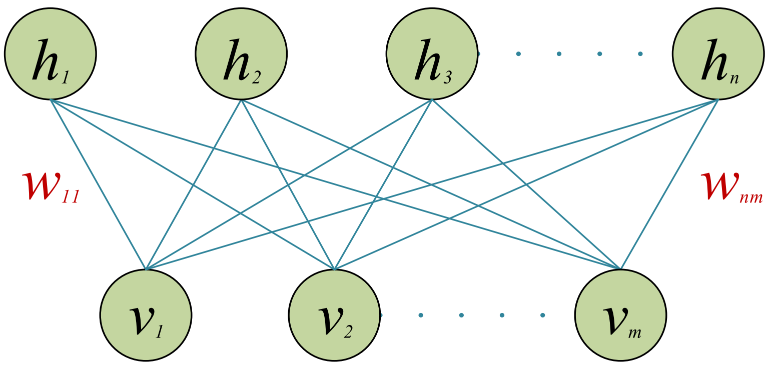


**Figure 5.1:** Restricted Boltzmann Machine represented as an undirected graphical model. The RBM has M visible units and N hidden units with N × M weights between each visible-hidden pair. The biases related to the visible and hidden units are not illustrated to keep the illustration simple. There exist a bias term for each visible unit and hidden unit.

The terminology of the visible and hidden units are similar to the terminology of the NN explained in Section 4.2. That is, the visible units are observed and the hidden units are not. The hidden units are, in a similar way as with the NN, used to capture the dependencies between the visible units and are intended to provide the model with the capability of proper generalization [Fischer and Igel, 2014, p. 10] [Krizhevsky, 2009]. The major difference between a RBM and a one-hidden-layer-NN is that the RBM does not have any specified inputs or outputs and then both the visible and hidden units are considered as being random variables.

Another key difference between RBMs and NNs is that the RBM is a generative model whereas a NN, when used for classification, is a discriminative model. By generative it is meant that the RBM models a joint probability distribution, which is different from the NN, which models a conditional distribution. The idea of using RBMs in DBNs is to achieve a DNN with good initial weights, meaning that the DNN as a generative model has a good representation of the underlying statistics of the data. If such a model can be achieved, backpropagation can effectively be applied making the generative model into a discriminative model, hence circumventing the impracticality of using backpropagation on randomized weights, as described in the former section. Several different variants of the RBM exists, which are the Bernoulli-Bernoulli RBM (BBRBM), the Gaussian-Bernoulli RBM (GBRBM) and the Gaussian-Gaussian RBM (GGRBM) [Hinton, 2012] [Bengio et al., 2007]. Instead of using the term Bernoulli, the term binary is also used in the literature [Hinton, 2012] [Fischer and Igel, 2012], but in this report Bernoulli is the term of choice.

The BBRBM is named as such because both the visible units and the hidden units are distributed according to a Bernoulli distribution, meaning that both the visible and hidden units only take values as $(\mathbf{v}, \mathbf{h}) \in \{0, 1\}^{m+n}$. The GBRBM differs from the BBRBM by having the visible units distributed according to a normal distribution, meaning that the visible units are continuous, i.e., $\mathbf{v} \in (-\infty, \infty)^m$ and $\mathbf{h} \in \{0, 1\}^n$. The GGRBM differs from the former RBMs by having both the visible and the hidden units distributed according to a normal distribution, i.e., $(\mathbf{v}, \mathbf{h}) \in (-\infty, \infty)^{m+n}$.

Which RBM architecture to choose depends on the application, but in the case of speech processing using the IBM, a common choice in the literature is to use both a BBRBM and a GBRBM [Mohamed et al., 2012] [Healy et al., 2013]. The GBRBM will form the input layer of the DBN, as described in Section 5.4, since the speech features used for the DBN are real valued. The Bernoulli-Bernoulli RBMs will be used to form the subsequent layers of the DBN, which is primarily related to a trade off in model complexity and training capability. The design of the DBN will be further described in Section 5.4. The BBRBM and the GBRBM will be described in more details in the following sections.

### 5.1.1 Bernoulli-Bernoulli RBM

As mentioned, a BBRBM consists of visible units and hidden units and together they form a generative model. The probability distribution that this model represents, is given by the Gibbs distribution, also known as the Boltzmann distribution. This distribution is given by Eq. (5.1) [Fischer and Igel, 2012] [Bishop, 2009, 387] [Hinton, 2012].

$$p(\mathbf{v}, \mathbf{h}) = \frac{1}{Z} e^{-E(\mathbf{v}, \mathbf{h})} \tag{5.1}$$

where

$$E(\mathbf{v}, \mathbf{h}) = -\sum_{i=1}^{n} \sum_{j=1}^{m} w_{ij} h_i v_j - \sum_{j=1}^{m} b_j v_j - \sum_{i=1}^{n} c_i h_i \tag{5.2}$$

and

$$Z = \sum_{\mathbf{h}, \mathbf{v}} e^{-E(\mathbf{h}, \mathbf{v})}. \tag{5.3}$$

It can be shown that a BBRBM can represent any discrete probability distribution arbitrarily well given enough hidden units [Bengio, 2009, p. 57]. Equation (5.2) is known as the energy function and Eq. (5.3) is known as the partition function and is a normalization factor used to ensure that Eq. (5.1) is a true probability distribution [Fischer and Igel, 2014]. In Eq. (5.2) $h_i$ and $v_j$ denote the state of the $i^{\text{th}}$ hidden unit and $j^{\text{th}}$ visible unit, respectively. Furthermore, $w_{ij}$ is the weight between them. Similarly, $c_i$ and $b_j$ are the bias terms related to the $i^{\text{th}}$ hidden unit and $j^{\text{th}}$ visible unit, respectively.

As mentioned, a RBM is a special type of a MRF. MRFs have some very convenient conditional independence properties known as the local Markov property, the pairwise Markov property and the global Markov property [Murphy, 2012, 661] [Fischer and Igel, 2012]. These properties cause the conditional probabilities of a MRF to factorize. This factorization makes Gibbs sampling from MRFs easy. The next couple of paragraphs introduces these properties and argue that they can be applied to RBMs such that the conditional probabilities of a RBM factorize as given by Eq. (5.4) and Eq. (5.5).

The local Markov property states that a node is conditional independent of the rest of the nodes in a MRF given its Markov blanket. The Markov blanket is defined as the nodes that cause a given node independent of all other nodes and is given by the immediate neighbours of the given node. The pairwise Markov property states that two nodes are independent if no direct path are between them. Finally, the global Markov property states that two disjoint subsets $A$ and $B$ of a MRF are conditionally independent of the subset $C$ if all paths from $A$ to $B$ goes through $C$, i.e., $A \perp\!\!\!\perp B|C$. However, if the joint distribution given by the MRF is strictly positive, the pairwise Markov property implies the global Markov property [Murphy, 2012, 661] [Fischer and Igel, 2012].

The conditional independence of the MRF and the fact that Eq. (5.2) is a strictly positive function, implies that the conditional independence properties of the MRF holds for the RBM. Using these properties it is seen from Fig. 5.1 that all hidden units are conditional independent on each other when conditioned on the visible units. Likewise, all visible units are independent on each other when conditioned on the hidden units. They are so because no inter-visible or inter-hidden connections are present, hence the Markov properties apply. This is also why the RBM is known as "restricted", since paths are restricted to occur only between layers and not within the layers [Bengio, 2009, 55] [Fischer and Igel, 2012, 11]. From these arguments the conditional distributions $p(\mathbf{v}|\mathbf{h})$ and $p(\mathbf{h}|\mathbf{v})$ of a RBM factorizes as given by Eq. (5.4) and Eq. (5.5) respectively.

$$p(\mathbf{v}|\mathbf{h}) = \prod_{j=1}^{m} p(v_j|\mathbf{h}), \tag{5.4}$$

and

$$p(\mathbf{h}|\mathbf{v}) = \prod_{i=1}^{n} p(h_i|\mathbf{v}) \tag{5.5}$$

Furthermore, the factors in Eq. (5.4) and Eq. (5.5) can be shown to be given by Eq. (5.6) and Eq. (5.7), respectively [Fischer and Igel, 2012] [Bengio, 2009].

$$p(v_j = 1|\mathbf{h}) = \varphi\left(b_j + \sum_{i=1}^{n} w_{ij} h_i\right) \tag{5.6}$$

$$p(h_i = 1|\mathbf{v}) = \varphi\left(c_i + \sum_{j=1}^{m} w_{ij} v_j\right) \tag{5.7}$$

where

$$\varphi(z) = \frac{1}{1 + e^{-z}}. \tag{5.8}$$

Equation (5.8) is the sigmoid function first presented in Eq. (4.3), which is an interesting result since it allows a Bernoulli-Bernoulli RBMs to be interpreted as a stochastic NN with sigmoid activation functions [Fischer and Igel, 2012, 11]. The simple form of the conditionals in Eq. (5.6) and Eq. (5.7) are very convenient since it makes inference from the RBM easy using block Gibbs sampling, which is further described in Section 5.2.

### 5.1.2 Gaussian-Bernoulli RBM

The GBRBM also uses the Boltzmann distribution as given by Eq. (5.1), but the GBRBM differs from the BBRBM by having continuous visible units and binary hidden units, i.e., $\mathbf{v} \in (-\infty, \infty)^m$ and $\mathbf{h} \in \{0, 1\}^n$. This configuration is more suited when RBMs are used to model statistics of natural images or speech [Hinton, 2012]

[Melchior, 2012] [Krizhevsky, 2009]. The GBRBM also differs from the BBRBM by having a different energy function given by Eq. (5.9) [Hinton, 2012].

$$E(\mathbf{v}, \mathbf{h}) = \sum_{j=1}^{m} \frac{(v_j - b_j)^2}{2\sigma_j^2} - \sum_{i=1}^{n}\sum_{j=1}^{m} w_{ij} h_i \frac{v_j}{\sigma_j^2} - \sum_{i=1}^{n} c_i h_i \tag{5.9}$$

The parameter $\sigma_j^2$ is the variance of the visible units and is typically set to one, conditioned on that the visible units are normalized [Wang and Wang, 2013] [Hinton, 2012]. However, it has been shown that learning the variance as well, improve the performance of the RBM [Cho et al., 2011]. It has further been shown that Gaussian-Bernoulli RBMs, when used as the first layer in DBNs with binary hidden units, possess a universal approximation property, meaning that Gaussian-Bernoulli RBMs used in DBNs can model any strictly positive probability density arbitrarily well [Krause et al., 2013]. Since the GBRBM represents a mixed distribution containing both discrete and continuous variables, the partition function changes slightly, since integration must be applied instead of summation over the visible units, as given by

$$Z = \int \sum_{\mathbf{h}} e^{-E(\mathbf{h},\mathbf{v})} d\mathbf{v}. \tag{5.10}$$

Since the probability distribution of the GBRBM, equivalently to the BBRBM, is strictly positive, the same properties apply: In particular, the conditional distributions given by Eq. (5.4) and Eq. (5.5) also apply for the GBRBM. From this observation the factors in Eq. (5.4) and Eq. (5.5) can for the GBRBM be shown to be given by Eq. (5.11) and Eq. (5.12) [Melchior, 2012] [Krizhevsky, 2009].

$$p(v_j|\mathbf{h}) = \mathcal{N}\left(v_j; b_j + \sum_{i=1}^{n} w_{ij} h_i, \sigma_j^2\right) \tag{5.11}$$

and

$$p(h_i = 1|\mathbf{v}) = \varphi\left(c_i + \sum_{j=1}^{m} w_{ij} \frac{v_j}{\sigma_j^2}\right). \tag{5.12}$$

## 5.2 Sampling from Restricted Boltzmann Machines

Sampling from a RBM is beneficial in primarily two ways. At first it is needed for constructing the estimate of the log-likelihood function, which is used for training the RBM, as explained in Section 5.3, and secondly samples from the RBM distribution can be used for evaluating if the RBM captures the correct statistics by simply looking at the samples [Bengio, 2009, p. 58].

A commonly used sample method for RBMs is the Gibbs sampler, which is a Markov Chain Monte Carlo (MCMC) method. Other methods such as the Inverse Transform Sampling (ITS) or the Metropolis-Hastings method could probably also be used but due to the simple formulation of the conditional distributions of the RBM, the Gibbs sampler is the obvious choice. The ITS would require the Cumulative Distribution Function (CDF) and the inverse of the joint probability function to exist and the Metropolis-Hastings would require the choice of a proposal distribution and its parameters. The Gibbs sampler on the other hand only requires the conditional distributions [Bishop, 2009, p. 542].

MCMC methods are named as such because they are based on Markov chains and the principal behind the MCMC sampling method is to construct a reversible and ergodic Markov chain and run it for a sufficient amount of time. That the Markov chain is reversible and ergodic means that regardless of the initial state it will converge to a steady state or equilibrium distribution [Bishop, 2009, p. 542], [Bengio, 2009, p. 599].

The basic idea of Gibbs sampling is to construct a Markov chain with as many variables as hidden and visible units in the RBM. Each variable is then iteratively updated based on the conditional distribution of a given variable, conditioned on all the other variables. Fortunately, the RBM has no inter-hidden or inter-visible connections meaning that all hidden variables and all visible variables can be sampled at the same time. This can be done due to the conditional independence properties given by Eq. (5.4) and Eq. (5.5), which state that a hidden unit and a visible unit are only dependent on their Markov blanket, i.e., their immediate neighbours.

From this argument the Gibbs sampler for a RBM can be expressed graphically as Fig. 5.2 and mathematical as given by Eq. (5.13).

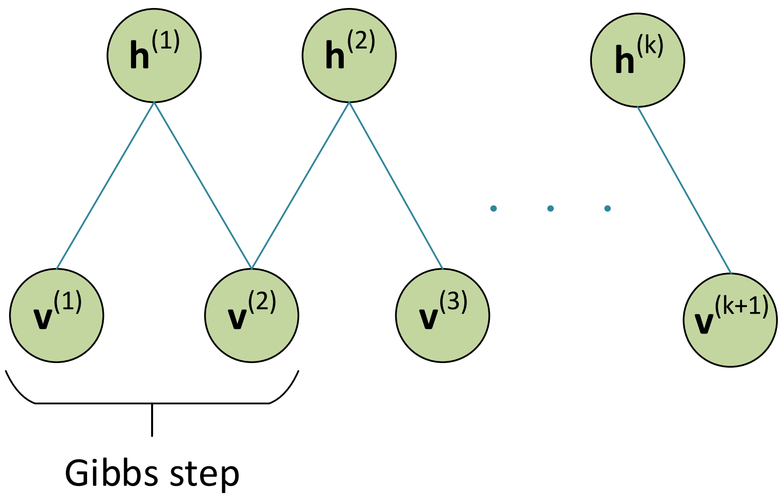


**Figure 5.2:** Gibbs sampling of a RBM. $\mathbf{v}^{(1)}$ is a vector containing the state at time 1 for all M visible units and $\mathbf{h}^{(1)}$ is likewise a vector containing the state at time 1 for all N hidden units. $\mathbf{v}^{(2)}$ is hereby the vector with the state of all visible units after one Gibbs step.

$$\begin{aligned}
\mathbf{v}^{(1)} &\sim \hat{P}(\mathbf{v}) \\
\mathbf{h}^{(1)} &\sim P(\mathbf{h}|\mathbf{v}^{(1)}) \\
\mathbf{v}^{(2)} &\sim P(\mathbf{v}|\mathbf{h}^{(1)}) \\
\mathbf{h}^{(2)} &\sim P(\mathbf{h}|\mathbf{v}^{(2)}) \\
&\vdots \\
\mathbf{v}^{(k+1)} &\sim P(\mathbf{v}|\mathbf{h}^{(k)})
\end{aligned} \tag{5.13}$$

In Eq. (5.13) $\hat{P}$ is the probability distribution of the training data and a sample from this distribution i.e., a training example, is typically used to initialize the Gibbs chain [Bengio, 2009] [Fischer and Igel, 2014]. An argument for doing so is that during training of the RBM the RBM model becomes more and more accurate hence $\hat{P}$ becomes more similar to $P$, so using a training example as the initial state can increase the rate of convergence of the Gibbs chain [Bengio, 2009].

Even though the Gibbs sampler is the most popular choice regarding sampling from RBMs it has been shown that other methods exist, which perform better without introducing a computational overhead [Brügge et al., 2013].

## 5.3 Training Restricted Boltzmann Machines

Since RBMs are generative models, they are trained in an generative fashion as well, i.e. unsupervised. The purpose of the training is to learn the RBM the underlying statistics of the training data. A RBM models the Boltzmann distribution as given by Eq. (5.1), but the distribution of interest is really the marginal distribution $p(\mathbf{v})$ since this is the distribution of the visible variables. This distribution is given by

$$p(\mathbf{v}) = \sum_{\mathbf{h}} p(\mathbf{v}, \mathbf{h}) = \frac{1}{Z} \sum_{\mathbf{h}} e^{-E(\mathbf{v},\mathbf{h})}. \tag{5.14}$$

The distribution given by Eq. (5.14) is parametrized with the parameters given by either Eq. (5.2) for a BBRBM or Eq. (5.9) for a GBRBM. The parameters are the weights $\mathbf{w}$, the biases $\mathbf{b}$ and $\mathbf{c}$ and the variance $\sigma^2$. In the following they are referenced by the variable $\boldsymbol{\theta}$ to simplify the derivation of the training algorithms. The goal of training a RBM is to find parameters $\boldsymbol{\theta}$ that maximizes the probability of the data, which makes the training data more likely to have been produced by the RBM. If $p(\mathbf{v}_{train})$, where $\mathbf{v}_{train}$ is a specific training example, is large, it is assumed that the distribution of the training data $\hat{p}(\mathbf{v})$ is similar to the distribution of the RBM $p(\mathbf{v})$. A typical approach to find estimates on such parameters is the method of ML using the log-likelihood function, which for the BBRBM is given by Eq. (5.15).

$$ln\ \mathcal{L}(\boldsymbol{\theta}; \mathbf{v}) = ln\ p(\mathbf{v}; \boldsymbol{\theta}) = ln \frac{1}{Z} \sum_{\mathbf{h}} e^{-E(\mathbf{v},\mathbf{h})} = ln \sum_{\mathbf{h}} e^{-E(\mathbf{v},\mathbf{h})} - ln \sum_{\mathbf{v},\mathbf{h}} e^{-E(\mathbf{v},\mathbf{h})} \tag{5.15}$$

It can be shown that maximizing the log-likelihood corresponds to minimizing the Kullback-Leibler Divergence (KLD) between $p(\mathbf{v}_{train})$ and $p(\mathbf{v})$ [Fischer and Igel, 2014], which means that by maximizing the log-likelihood, $p(\mathbf{v})$ becomes more similar with $p(\mathbf{v}_{train})$. The optimal parameters are then found by maximizing Eq. (5.15). This can be done exactly if an analytical solution exist or by an estimate if numerical methods are applied. No analytical ML solution exist for Eq. (5.15) and typically the parameters are estimated using gradient ascent with the parameters being updated according to Eq. (5.16) based on the change in parameters given by Eq. (5.17) [Fischer and Igel, 2014] [Bengio, 2009].

$$\boldsymbol{\theta}^{(t+1)} = \boldsymbol{\theta}^{(t)} + \Delta\boldsymbol{\theta}^{(t)} \tag{5.16}$$

$$\Delta\boldsymbol{\theta}^{(t)} = \eta \frac{\partial}{\partial \boldsymbol{\theta}^{(t)}} \Big( ln\ \mathcal{L}(\boldsymbol{\theta}|S) \Big) - \lambda \boldsymbol{\theta}^{(t)} + \gamma \Delta\boldsymbol{\theta}^{(t-1)} \tag{5.17}$$

In Eq. (5.17) $S = \{\mathbf{v}_1, \ldots, \mathbf{v}_l\}$ is the set of all training examples, $\eta$ is a learning rate, $\lambda$ is a weight decay factor and $\gamma$ is a momentum factor. The learning rate defines how large steps, in parameter space, the algorithm should take during each parameter update. The weight decay factor is used to penalize large weights and the momentum factor is used to prevent oscillations during training by introducing some memory [Fischer and Igel, 2014]. These parameters have influence on the training and their values are primarily based on empirical knowledge [Hinton, 2012].

In order to apply the gradient ascent method as given by Eq. (5.16) and Eq. (5.17) the gradient of the log-likelihood is needed, which for the BBRBM is given by Eq. (5.18) [Fischer and Igel, 2014].

$$\begin{aligned} \frac{\partial}{\partial\theta} ln\ \mathcal{L}(\boldsymbol{\theta}|\mathbf{v}) &= \frac{\partial}{\partial\theta}\left( ln \sum_{\mathbf{h}} e^{-E(\mathbf{v},\mathbf{h})}\right) - \left( ln \sum_{\mathbf{v},\mathbf{h}} e^{-E(\mathbf{v},\mathbf{h})}\right) \\ &= -\frac{1}{\sum\limits_{\mathbf{h}} e^{-E(\mathbf{v},\mathbf{h})}} \sum_{\mathbf{h}} e^{-E(\mathbf{v},\mathbf{h})} \frac{\partial E(\mathbf{v},\mathbf{h})}{\partial\theta} \\ &\quad + \frac{1}{\sum\limits_{\mathbf{v},\mathbf{h}} e^{-E(\mathbf{v},\mathbf{h})}} \sum_{\mathbf{v},\mathbf{h}} e^{-E(\mathbf{v},\mathbf{h})} \frac{\partial E(\mathbf{v},\mathbf{h})}{\partial\theta} \end{aligned} \tag{5.18}$$

By applying the relation of conditional probability as given by (5.19) [Kay, 2006, p. 75]

$$p(\mathbf{h}|\mathbf{v}) = \frac{p(\mathbf{v},\mathbf{h})}{p(\mathbf{v})} = \frac{\frac{1}{Z} e^{-E(\mathbf{v},\mathbf{h})}}{\frac{1}{Z}\sum\limits_{\mathbf{h}} e^{-E(\mathbf{v},\mathbf{h})}} = \frac{e^{-E(\mathbf{v},\mathbf{h})}}{\sum\limits_{\mathbf{h}} e^{-E(\mathbf{v},\mathbf{h})}} \tag{5.19}$$

and recalling the definition of the joint probability $p(\mathbf{v},\mathbf{h})$ given by Eq. (5.1) , Eq. (5.18) can be reduced to

$$\frac{\partial}{\partial\theta} ln\ \mathcal{L}(\boldsymbol{\theta}|\mathbf{v}) = -\sum_{\mathbf{h}} p(\mathbf{h}|\mathbf{v}) \frac{\partial E(\mathbf{v},\mathbf{h})}{\partial\theta} + \sum_{\mathbf{v},\mathbf{h}} p(\mathbf{v},\mathbf{h}) \frac{\partial E(\mathbf{v},\mathbf{h})}{\partial\theta}. \tag{5.20}$$

From Eq. (5.20) it should be noticed that the first term is the conditional expectation of the gradient of the energy function given the visible units and likewise the second term is the expectation of the gradient of the energy function under the joint distribution the BBRBM models. This observation simplifies the evaluation of Eq. (5.20) as presented next. Equation (5.20) also apply for Gaussian-Bernoulli RBMs if the summation over **v** is replaced by integration and the correct energy function is used.

### 5.3.1 Contrastive Divergence for Bernoulli-Bernoulli RBMs

Equation (5.20) represents the log-likelihood gradient for a BBRBM and it will now be presented how the log-likelihood gradient for a BBRBM can be estimated with respect to the parameters **w**, **b** and **c** using the principle of Contrastive Divergence (CD). The log-likelihood gradient with respect to a particular weight $w_{ij}$ is found by evaluating Eq. (5.20) with parameter $w_{ij}$ and for a single training example **v** it is given by Eq. (5.21). $\langle\,.\,\rangle$ denotes expectation.

$$\begin{aligned}\frac{\partial}{\partial w_{ij}} ln\ \mathcal{L}(\boldsymbol{\theta}|\mathbf{v}) &= -\sum_{\mathbf{h}} p(\mathbf{h}|\mathbf{v}) \frac{\partial E(\mathbf{v},\mathbf{h})}{\partial w_{ij}} + \sum_{\mathbf{v},\mathbf{h}} p(\mathbf{v},\mathbf{h}) \frac{\partial E(\mathbf{v},\mathbf{h})}{\partial w_{ij}} \\ &= \sum_{\mathbf{h}} p(\mathbf{h}|\mathbf{v}) h_i v_j - \sum_{\mathbf{v},\mathbf{h}} p(\mathbf{v},\mathbf{h}) h_i v_j \\ &= \langle h_i v_j \rangle_{p(\mathbf{h}|\mathbf{v})} - \langle h_i v_j \rangle_{p(\mathbf{v},\mathbf{h})}\end{aligned} \tag{5.21}$$

From Eq. (5.21) it is seen that the log-likelihood gradient with respect to a weight $w_{ij}$ is given as the difference between two expectations, which is the equivalent to the difference between two correlations of $h_i v_j$. The first term is the conditional expectation of $h_i v_j$ given the visible units, which is the same as the correlation between $h_i v_j$ evaluated with the conditional distribution $p(h_i|\mathbf{v})$. The second term is the expectation of $h_i v_j$ under the RBM probability distribution or the correlation of the same term evaluated at the joint distribution of the RBM. Since the visible units are given by the training example, $v_j$ is deterministic hence, $v_j$ can be taken outside the expectation. What is left is the conditional expectation of the hidden unit $h_i$ given the visible units, which is given by Eq. (5.12). From this argumentation Eq. (5.21) can be rewritten into Eq. (5.22) [Krizhevsky, 2009].

$$\frac{\partial}{\partial w_{ij}} ln\ \mathcal{L}(\boldsymbol{\theta}|\mathbf{v}) = p(h_i = 1|\mathbf{v}) v_j - \langle h_i v_j \rangle_{p(\mathbf{v},\mathbf{h})} \tag{5.22}$$

The last term in Eq. (5.22) is a bit more tedious to evaluate since it quickly becomes intractable, for regular sized RBMs when the sum over all possible configurations of **v** and **h** is evaluated. To circumvent this obstacle Geoffrey Hinton proposed the Contrastive Divergence (CD) algorithm, which basically consists of two approximations of the last term in Eq. (5.21) [Hinton, 2002] [Bengio, 2009, 59]. The first approximation is that the expectation is approximated by a sample of $h_i$ and a sample of

$v_j$ from the joint distribution $p(\mathbf{v}, \mathbf{h})$. This can be achieved by, for example, Gibbs sampling as described in Section 5.2. However, using Gibbs sampling requires that the Markov chain is running long enough, to ensure that it has converged to its equilibrium distribution, before a true sample from the joint distribution can be sampled. This is where Geoffrey Hinton proposed the second approximation by only running the Markov chain for $k$-steps and usually only one [Hinton, 2002]. Using these two approximation Eq. (5.21) can be rewritten into Eq. (5.23) where the superscript $(k)$ indicates the k$^{\text{th}}$ sample from the Gibbs chain given by Eq. (5.13).

$$\frac{\partial}{\partial w_{ij}} ln\ \mathcal{L}(\boldsymbol{\theta}|\mathbf{v}) \approx p(h_i = 1|\mathbf{v}^{(0)})v_j^{(0)} - p(h_i = 1|\mathbf{v}^{(k)})v_j^{(k)} \tag{5.23}$$

In a similar fashion as just described for the log-likelihood gradient with respect to a weight, the log-likelihood gradient for the biases can be defined and they are given by Eq. (5.24) and Eq. (5.25) for the visible biases and hidden biases respectively.

$$\begin{aligned}\frac{\partial}{\partial b_j} ln\ \mathcal{L}(\boldsymbol{\theta}|\mathbf{v}) &= -\sum_{\mathbf{h}} p(\mathbf{h}|\mathbf{v})\frac{\partial E(\mathbf{v},\mathbf{h})}{\partial b_j} + \sum_{\mathbf{v},\mathbf{h}} p(\mathbf{v},\mathbf{h})\frac{\partial E(\mathbf{v},\mathbf{h})}{\partial b_j} \\ &= \sum_{\mathbf{h}} p(\mathbf{h}|\mathbf{v})v_j - \sum_{\mathbf{v},\mathbf{h}} p(\mathbf{v},\mathbf{h})v_j \\ &= \langle v_j\rangle_{p(\mathbf{h}|\mathbf{v})} - \langle v_j\rangle_{p(\mathbf{v},\mathbf{h})} \\ &\approx v_j^{(0)} - v_j^{(k)}\end{aligned} \tag{5.24}$$

$$\begin{aligned}\frac{\partial}{\partial c_i} ln\ \mathcal{L}(\boldsymbol{\theta}|\mathbf{v}) &= -\sum_{\mathbf{h}} p(\mathbf{h}|\mathbf{v})\frac{\partial E(\mathbf{v},\mathbf{h})}{\partial c_i} + \sum_{\mathbf{v},\mathbf{h}} p(\mathbf{v},\mathbf{h})\frac{\partial E(\mathbf{v},\mathbf{h})}{\partial c_i} \\ &= \sum_{\mathbf{h}} p(\mathbf{h}|\mathbf{v})h_i - \sum_{\mathbf{v},\mathbf{h}} p(\mathbf{v},\mathbf{h})h_i \\ &= \langle h_i\rangle_{p(\mathbf{h}|\mathbf{v})} - \langle h_i\rangle_{p(\mathbf{v},\mathbf{h})} \\ &\approx p(h_i = 1|\mathbf{v}^{(0)}) - p(h_i = 1|\mathbf{v}^{(k)})\end{aligned} \tag{5.25}$$

Equations (5.23), (5.24) and (5.25) can now be inserted into Eq. (5.16) and Eq. (5.17) to update the parameters of a BBRBM, hence learning it to represent the statistics underlying the training data. If $\eta = 1$ and $\lambda = \gamma = 0$, in Eq. (5.16) and Eq. (5.17), the change in weights and biases for the visible and hidden units are given by Eq. (5.26), Eq. (5.27) and Eq. (5.28) respectively.

$$\Delta w_{ij} = p(h_i = 1|\mathbf{v}^{(0)})v_j^{(0)} - p(h_i = 1|\mathbf{v}^{(k)})v_j^{(k)} \tag{5.26}$$

$$\Delta b_j = v_j^{(0)} - v_j^{(k)} \tag{5.27}$$

$$\Delta c_i = p(h_i = 1|\mathbf{v}^{(0)}) - p(h_i = 1|\mathbf{v}^{(k)}) \tag{5.28}$$

It should be mentioned that setting $\eta = 1$ and $\lambda = \gamma = 0$ is done for illustrative purposes only to keep the expressions simple. Algorithm 1 illustrates the pseudo code for a batch learning algorithm using CD for a BBRBM during one epoch.

**Algorithm 1:** k-step Contrastive Divergence

```
Data: RBM hidden and visible units and training batch $S$
Result: Gradient estimates : $w_{ij}, b_j$ and $c_i \quad for \quad i = 1, \ldots, n, \;\; j = 1, \ldots, m$
initialization $w_{ij} = b_j = c_i = 0 \quad for \quad i = 1, \ldots, n, \;\; j = 1, \ldots, m$;
forall the $v \in S$ do
    for $t$ = 0,...,k-1 do
        for $i$ = 1,...,$n$ do $h_i^{(t)} \sim p(h_i|\mathbf{v}^{(t)})$;
        for $j$ = 1,...,$m$ do $v_j^{(t+1)} \sim p(v_j|\mathbf{h}^{(t)})$;
    end
    for $i$ = 1,..., $n$, $j$ = 1,...,$m$ do
        $\Delta w_{ij} = \Delta w_{ij} + p(h_i = 1|\mathbf{v}^{(0)})v_j^{(0)} - p(h_i = 1|\mathbf{v}^{(k)})v_j^{(k)}$
    end
    for $j$ = 1,...,$m$ do
        $\Delta b_j = \Delta b_j + v_j^{(0)} - v_j^{(k)}$
    end
    for $i$ = 1,..., $n$ do
        $\Delta c_i = \Delta c_i + p(h_i = 1|\mathbf{v}^{(0)}) - p(h_i = 1|\mathbf{v}^{(k)})$
    end
end
```

### 5.3.2 Contrastive Divergence for Gaussian-Bernoulli RBMs

CD for Gaussian-Bernoulli RBMs is slightly different than CD for Bernoulli-Bernoulli RBMs. This is because the joint distribution $p(\mathbf{v}, \mathbf{h})$ is a mixed distribution due to the visible units being Gaussian distributed. Since $p(\mathbf{v}, \mathbf{h})$ is a mixed distribution the gradient of the log-likelihood function changes slightly and is given by Eq. (5.29) [Melchior, 2012] [Cho et al., 2011] [Yamashita et al., 2014].

$$\frac{\partial}{\partial \theta} ln \; \mathcal{L}(\boldsymbol{\theta}|\mathbf{v}) = -\sum_{\mathbf{h}} p(\mathbf{h}|\mathbf{v}) \frac{\partial E(\mathbf{v}, \mathbf{h})}{\partial \theta} + \int \sum_{\mathbf{h}} p(\mathbf{v}, \mathbf{h}) \frac{\partial E(\mathbf{v}, \mathbf{h})}{\partial \theta} d\mathbf{v} \tag{5.29}$$

In a similar manner as with the BBRBM, Eq. (5.29) can be evaluated with the weights and the biases using the energy function given by Eq. (5.9), hence giving the gradient estimates of the log-likelihood function for the GBRBM. This is for **w**, **b** and **c** given by Eqs. (5.30), (5.31) and (5.32) respectively [Melchior, 2012]. The corresponding update rules for $w_{ij}$, $b_j$ and $c_i$ with $\eta = 1$ and $\lambda = \gamma = 0$ are given by Eqs. (5.33), (5.34) and (5.35) respectively.

$$
\begin{aligned}
\frac{\partial}{\partial w_{ij}} ln\ \mathcal{L}(\boldsymbol{\theta}|\mathbf{v}) &= -\sum_{\mathbf{h}} p(\mathbf{h}|\mathbf{v}) \frac{\partial E(\mathbf{v},\mathbf{h})}{\partial w_{ij}} + \int \sum_{\mathbf{h}} p(\mathbf{v},\mathbf{h}) \frac{\partial E(\mathbf{v},\mathbf{h})}{\partial w_{ij}} d\mathbf{v} \\
&= \sum_{\mathbf{h}} p(\mathbf{h}|\mathbf{v}) \frac{1}{\sigma_j^2} h_i v_j - \int \sum_{\mathbf{h}} p(\mathbf{v},\mathbf{h}) \frac{1}{\sigma_j^2} h_i v_j d\mathbf{v} \\
&= \frac{1}{\sigma_j^2} \left( \langle h_i v_j \rangle_{p(\mathbf{h}|\mathbf{v})} - \langle h_i v_j \rangle_{p(\mathbf{v},\mathbf{h})} \right) \\
&\approx \frac{1}{\sigma_j^2} \left( p(h_i = 1|\mathbf{v}^{(0)}) v_j^{(0)} - p(h_i = 1|\mathbf{v}^{(k)}) v_j^{(k)} \right)
\end{aligned}
\tag{5.30}
$$

$$
\begin{aligned}
\frac{\partial}{\partial b_j} ln\ \mathcal{L}(\boldsymbol{\theta}|\mathbf{v}) &= -\sum_{\mathbf{h}} p(\mathbf{h}|\mathbf{v}) \frac{\partial E(\mathbf{v},\mathbf{h})}{\partial b_j} + \int \sum_{\mathbf{h}} p(\mathbf{v},\mathbf{h}) \frac{\partial E(\mathbf{v},\mathbf{h})}{\partial b_j} d\mathbf{v} \\
&= \sum_{\mathbf{h}} p(\mathbf{h}|\mathbf{v}) \frac{(v_j - b_j)}{\sigma_j^2} - \int \sum_{\mathbf{h}} p(\mathbf{v},\mathbf{h}) \frac{(v_j - b_j)}{\sigma_j^2} d\mathbf{v} \\
&= \frac{1}{\sigma_j^2} \left( \sum_{\mathbf{h}} p(\mathbf{h}|\mathbf{v})(v_j - b_j) - \int \sum_{\mathbf{h}} p(\mathbf{v},\mathbf{h})(v_j - b_j) d\mathbf{v} \right) \\
&= \frac{1}{\sigma_j^2} \left( \langle v_j - b_j \rangle_{p(\mathbf{h}|\mathbf{v})} - \langle v_j - b_j \rangle_{p(\mathbf{v},\mathbf{h})} \right) \\
&\approx \frac{1}{\sigma_j^2} \left( (v_j^{(0)}) - (v_j^{(k)}) \right)
\end{aligned}
\tag{5.31}
$$

$$
\begin{aligned}
\frac{\partial}{\partial c_i} ln\ \mathcal{L}(\boldsymbol{\theta}|\mathbf{v}) &= -\sum_{\mathbf{h}} p(\mathbf{h}|\mathbf{v}) \frac{\partial E(\mathbf{v},\mathbf{h})}{\partial c_i} + \int \sum_{\mathbf{h}} p(\mathbf{v},\mathbf{h}) \frac{\partial E(\mathbf{v},\mathbf{h})}{\partial c_i} d\mathbf{v} \\
&= \sum_{\mathbf{h}} p(\mathbf{h}|\mathbf{v}) h_i - \int \sum_{\mathbf{h}} p(\mathbf{v},\mathbf{h}) h_i d\mathbf{v} \\
&= \langle h_i \rangle_{p(\mathbf{h}|\mathbf{v})} - \langle h_i \rangle_{p(\mathbf{v},\mathbf{h})} \\
&\approx p(h_i = 1|\mathbf{v}^{(0)}) - p(h_i = 1|\mathbf{v}^{(k)})
\end{aligned}
\tag{5.32}
$$

$$
\Delta w_{ij} = \frac{1}{\sigma_j^2} \left( p(h_i = 1|\mathbf{v}^{(0)}) v_j^{(0)} - p(h_i = 1|\mathbf{v}^{(k)}) v_j^{(k)} \right) \tag{5.33}
$$

$$
\Delta b_j = \frac{1}{\sigma_j^2} \left( v_j^{(0)} - v_j^{(k)} \right) \tag{5.34}
$$

$$
\Delta c_i = p(h_i = 1|\mathbf{v}^{(0)}) - p(h_i = 1|\mathbf{v}^{(k)}) \tag{5.35}
$$

It is seen from Eq. (5.33) and Eq. (5.34) that the update rule for the weights and the biases for the visible units of a GBRBM is just a scaled version of the update rule for the weights and visible biases of a BBRBM, as given by Eq. (5.26) and Eq. (5.27), with $\frac{1}{\sigma_j^2}$ as the scaling factor. The update rule for the biases for the hidden units does not change compared to the BBRBM as seen by Eq. (5.35), since they only influence the hidden variables, which are still binary.

### 5.3.3 RBM Example

To illustrate the concept of RBMs two examples have been constructed, one for the BBRBM and one for the GBRBM. The examples are based on the graph shown in Fig. 5.3, which consists of two visible units and two hidden units. This graph can be interpreted as both a BBRBM and a GBRBM. The small size is chosen to make it possible to plot the Probability Mass Function (PMF)/PDF as well as to compute all conditional probabilities analytically.

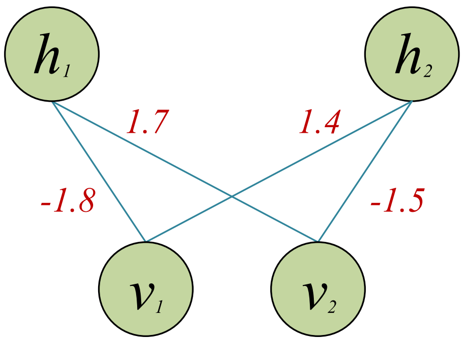


**Figure 5.3:** A RBM consisting of two visible units and two hidden units which makes it tractable to compute all probabilities represented by the RBM analytically.

#### Bernoulli-Bernoulli RBM Example

This example is intended to illustrate that a BBRBM based on Fig. 5.3 represents a Bernoulli distribution and that one can sample from the marginal PMF, using Gibbs sampling. All possible configurations of **v** and **h** as well as the joint and marginal PMF of the BBRBM is presented in Tab. 5.1.

Since the BBRBM consists of only two visible units and two hidden units there are a total of 16 different configurations of **v** and **h** as seen from Tab. 5.1. This number scales exponentially with respect to the number of visible units $M$ and hidden units $N$ so that the total number of configurations is given by $2^M 2^N$.

This is the primary reason why the computation of the last term in the log-likelihood function given by Eq. (5.15) is intractable for large networks, because it

| $\mathbf{v}$ | $\mathbf{h}$ | $p(\mathbf{v},\mathbf{h})$ | $p(\mathbf{v})$ |
|---|---|---|---|
| 1 1 | 1 1 | 0.038 | 0.1693 |
| 1 1 | 0 1 | 0.042 | |
| 1 1 | 1 0 | 0.042 | |
| 1 1 | 0 0 | 0.046 | |
| 0 1 | 1 1 | 0.057 | 0.3694 |
| 0 1 | 0 1 | 0.010 | |
| 0 1 | 1 0 | 0.255 | |
| 0 1 | 0 0 | 0.046 | |
| 1 0 | 1 1 | 0.031 | 0.2748 |
| 1 0 | 0 1 | 0.189 | |
| 1 0 | 1 0 | 0.007 | |
| 1 0 | 0 0 | 0.046 | |
| 0 0 | 1 1 | 0.046 | 0.1866 |
| 0 0 | 0 1 | 0.046 | |
| 0 0 | 1 0 | 0.046 | |
| 0 0 | 0 0 | 0.046 | |

**Table 5.1:** Probabilities of a Bernoulli-Bernoulli RBM with two visible units and two hidden units as shown by Fig. 5.3

has to iterate over $2^M 2^N$ configurations. The joint PMF $p(\mathbf{v},\mathbf{h})$ are computed using Eqs. (5.1), (5.2) and (5.3). Again, it should be noticed that computing the partition function requires $2^M 2^N$ iterations. The marginal PMF $p(\mathbf{v})$ is computed by summing over all configurations of $\mathbf{h}$ as given by Eq. (5.14).

From the marginal distribution it is seen that the four configurations of $\mathbf{v}$ have different probabilities ranging from 0.16 to 0.37, where $\mathbf{v} = [0\ 1]$ is the configuration with the highest probability. If samples were drawn from $p(\mathbf{v})$, one would expect that 37% of the samples were equal to [0 1]. To confirm this claim, Gibbs chains are constructed using the conditional probabilities given by Eq. (5.6) and Eq. (5.7). Using the procedure given by Eq. (5.13) samples from the marginal distribution $p(\mathbf{v})$ of the BBRBM are drawn. To investigate how the Gibbs chain converges, 10 Gibbs chains have been constructed. The first Gibbs chain runs one Gibbs step, the next Gibbs chain runs two Gibbs steps, and so on, up to 10 Gibbs steps has been performed. Each Gibbs chain is initialized with a sample from a bivariate Bernoulli distribution with $P = 0.5$ and repeated $500 \times 10^3$ times, such that $500 \times 10^3$ samples are drawn. Finally, the occurrence of the four possible states is computed and plotted as shown by Fig. 5.4.

From Fig. 5.4 it is seen that the probability without any Gibbs step is 0.25, which is expected since the probability of any outcome of a Bivariate Bernoulli random process with $p = 0.5$ is $p^2 = 0.25$. Next, it is seen that the Gibbs chain is converging and after a few Gibbs step it appears that the Gibbs chain has reached it equilibrium distribution, which as seen from Fig. 5.4 is the correct PMF with an accuracy of three

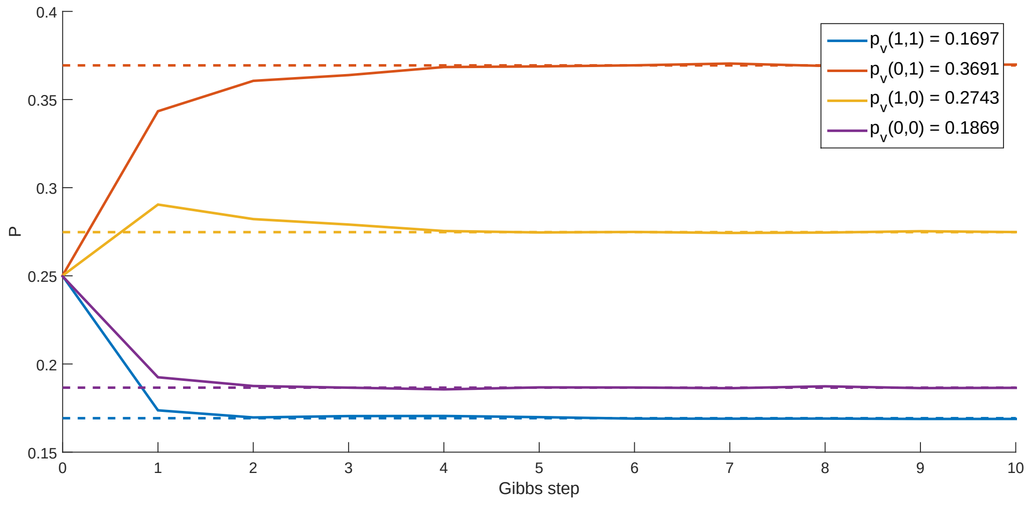


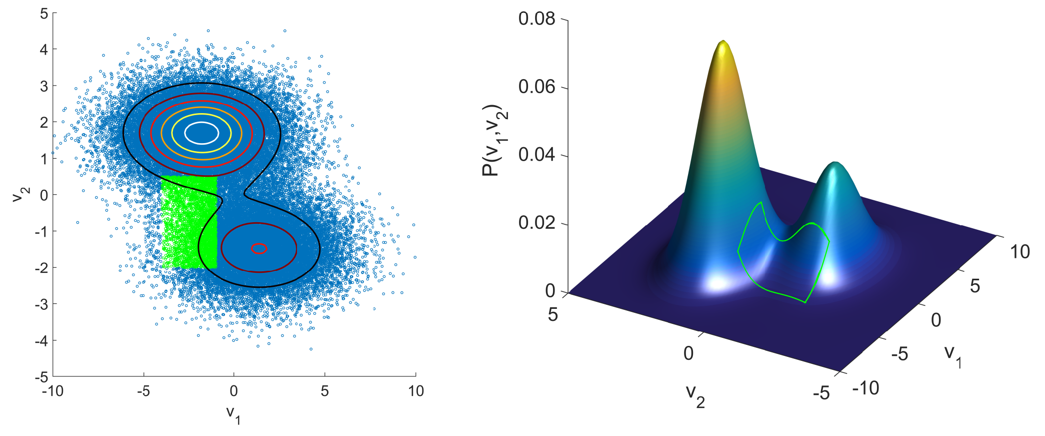
**Figure 5.4:** Convergence characteristic for a Gibbs chain for a BBRBM

significant digits.

### Gaussian-Bernoulli RBM Example

This example is intended to illustrate that a GBRBM based on Fig. 5.3 has a Gaussian marginal PDF, from which samples can be drawn using Gibbs sampling. Since the GBRBM is a mixed distribution, i.e., both consist of discrete and continuous random variables, integration must be used instead of summation, when evaluating the marginal PDF $p(\mathbf{v})$. To validate that the Gibbs chain is functioning correctly and to investigate the marginal PDF represented by the GBRBM, the marginal PDF $p(\mathbf{v})$ is computed. $p(\mathbf{v})$ has been computed using Eqs. (5.1), (5.9) and (5.10). The variances in the energy function given by Eq. (5.9) have been chosen to the following values: $\sigma_1^2 = 5$ and $\sigma_2^2 = 0.5$. The Gibbs chain is constructed using Eqs. (5.11), (5.12) and sampling is following the procedure given by Eq. (5.13).

$500 \times 10^3$ samples have been drawn from $p(\mathbf{v})$ each based on 100 iterations of Gibbs sampling. The Gibbs chain was initialized with two Independent and Identically Distributed (IID) Gaussian distributed random samples with zero mean and unit variance. The drawn samples are plotted in Fig. 5.5a with the contours of $p(\mathbf{v})$ plotted as an overlay. The PDF of $p(\mathbf{v})$ is plotted in Fig. 5.5b. From the figures it is seen that the marginal distribution $p(\mathbf{v})$ is a bivariate and bimodal Gaussian distribution and it appears that the samples are distributed very similar to the contours of $p(\mathbf{v})$. To investigate the immediate accuracy of the Gibbs samples the specific number of samples within a certain range is computed and related with the analytical value of the area under the marginal distribution of $p(\mathbf{v})$, given the same range. One would expect that the number of samples within a given range is approximately equivalent to the area under $p(\mathbf{v})$ given the same range. The area chosen is illustrated by the green enclosing on Fig. 5.5b and the samples within the area are colored green as illustrated on Fig. 5.5a.



**(a)** Samples drawn from $p(\mathbf{v})$ **(b)** Marginal PDF $p(\mathbf{v})$

**Figure 5.5:** In Fig. 5.5a samples drawn from $p(\mathbf{v})$ using 100 iterations of Gibbs sampling is illustrated. The contour plot is the contours of the real PDF $p(\mathbf{v})$ as shown in Eq. 5.14. The green samples in Fig. 5.5a represents 5.63% of the total number of samples which is similar to the area under $p(\mathbf{v})$, for the same area as illustrated by the green enclosing in Fig. 5.5b, which is 0.0562.

To compute the expected number of samples within the green enclosure the function given by Eq. (5.14) is integrated in the interval $v_1 \in (-4, -1)$ and $v_2 \in (-2, 0.5)$ as given by Eq. (5.36).

$$p_{enclosed}(\mathbf{v}) = \int_{-2}^{0.5} \int_{-4}^{-1} \frac{1}{Z} \sum_{\mathbf{h}} e^{-E(\mathbf{v},\mathbf{h})} dv_1 dv_2 = 0.0562 \tag{5.36}$$

From Eq. (5.36) it is seen that the probability of being inside the green enclosing is 0.0562. By computing the ratio of green samples with respect to the blue samples it is found that 5.63% lies within the green enclosing. This means that the Gibbs samples coincide with the real $p(\mathbf{v})$ with two significant digits of accuracy.

## 5.4 Deep Belief Networks

Until now the focus, in this chapter, has been on training and sampling from individual RBMs, either having binary or continuous visible units. As mentioned in the introduction to this chapter, RBMs form the basic building blocks for, what is known as, DBNs. A DBN is a special kind of graphical model, which can be interpreted as an ordinary, but deep, neural network. If the DBN is properly trained as a generative model, its DNN interpretation will form a DNN where supervised training can be effectively applied using backpropagation. This is because the weights are initialized in a way that makes backpropagation more effective, even though the network is deep. This is what Geoffrey Hinton showed in [Hinton et al., 2006], which lead to a renewal of the interest in ANN.

Since a DBN is basically a collection of RBMs, the representational power of a DBN is greater than each of the RBMs individually and it can be shown that under some circumstances the generative model that the DBN represents is guaranteed to improve by adding additional layers [Bengio et al., 2007].

A DBN can be illustrated as a combination of directed and undirected probabilistic graphical models. The directed part of a DBN consists of all but the two topmost layers. These layers form an undirected graph as illustrated in Fig. 5.6. The joint distribution of a DBN is given by Eq. (5.37) [Bengio et al., 2007].

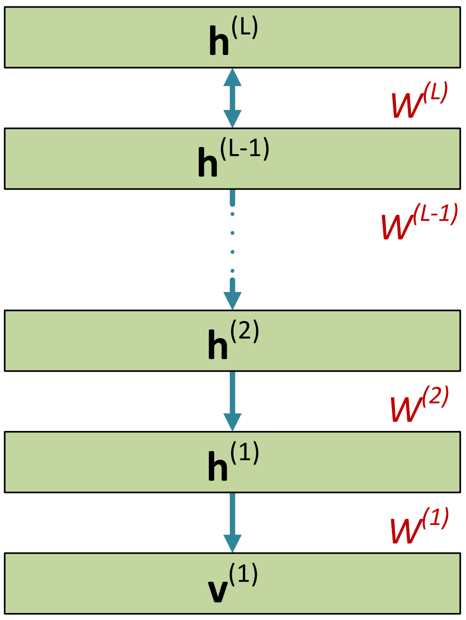


**Figure 5.6:** A $L$ layer Deep Belief Network with $\mathbf{v}^{(1)}$ as the visible layer and $\mathbf{h}^{(1)}$ to $\mathbf{h}^{(L)}$ as the hidden layers. Each layer can consist of an arbitrary number of element. The layers from $\mathbf{h}^{(1)}$ to $\mathbf{h}^{(L-1)}$ form at directed network and the two topmost layers form a RBM

$$p(\mathbf{v}, \mathbf{h}^{(1)}, \mathbf{h}^{(2)}, \ldots, \mathbf{h}^{(L)}) = p(\mathbf{v}|\mathbf{h}^{(1)})p(\mathbf{h}^{(1)}|\mathbf{h}^{(2)}) \ \ldots \\ p(\mathbf{h}^{(L-2)}|\mathbf{h}^{(L-1)})p(\mathbf{h}^{(L-1)}, \mathbf{h}^{(L)}) \tag{5.37}$$

where

$$p(\mathbf{h}^{(k)}|\mathbf{h}^{(k+1)}) = \prod_i p(h_i^{(k)}|\mathbf{h}^{(k+1)}) \tag{5.38}$$

and

$$p(h_i^{(k)} = 1|\mathbf{h}^{(k+1)}) = \varphi\left(\mathbf{c}_i^{(k)} + \sum_j w_{ij}^{(k)}\mathbf{h}_j^{(k+1)}\right) \tag{5.39}$$

In Fig. 5.6 an $L$ layer DBN is depicted. The undirected part of the DBN is an ordinary RBM, where the lower directed layers form a Bayesian Network also known as a Belief Network [Neal, 1992]. Equation (5.37) shows the joint distribution of a DBN with $L$ hidden layers $\mathbf{h}^{(k)}$ for $k = 1, \ldots, L$ and Eq. (5.38) shows how the conditional distribution of each layer factorizes into the conditional distribution given by Eq. (5.39), which is equivalent to the factorization of the joint distribution of the BBRBM given by Eq. (5.7). The visible layer can be represented by $\mathbf{v} = \mathbf{h}^{(0)}$. Furthermore $c_i^{(k)}$ and $w_{ij}^{(k)}$ are respectively the bias vector and weight matrix for layer $k$.

It is the constellation of both directed and undirected graphical models that make DBNs practical applicable since both sampling and inference is easy from such combined models. This is because Eq. (5.38) factorizes, which conditional distributions in Belief Networks under some circumstances do not due to the phenomena of "Explaining Away" [Hinton et al., 2006] [Bishop, 2009, p. 372]. The reason why "Explaining Away" is not of a concern in DBNs is due to the RBM in the top layer, which acts as a prior distribution that approximately counteracts the effects of explaining away [Hinton et al., 2006]. To understand why this is so the phenomena of "Explaining Away" has to be further explained. Keep in mind that the overall goal is to construct a deep network, which can be trained efficiently. So the question is how to construct such a network. Large MRFs are difficult to train due to the computation of the partition function, which is intractable to compute due to an exponential number of computations with respect to the number of units in the network. This drawback is somewhat circumvented by the RBM and the learning algorithm CD, although an RBM is a shallow network, it will later become clear that the RBM is the solution to the problem of "explaining away" in Belief Networks.

A Belief Network is a directed network and differs from the MRFs by having all of its conditional distributions normalized without the need of a partition function. This means that Belief Network does not posses the drawback of the intractable computation of a partition function such as the MRF. Instead Belief Networks posses the drawback of "explaining away" [Bishop, 2009, p. 377]. In MRFs the joint distribution factorizes into conditional distributions when conditioned on the Markov blanket, as earlier described in Section 5.1.1. This is an important property when learning parameters of a MRF as described for the RBM in Section 5.1.1. With Belief Networks this factorization does not occur for some graph structures due to "explaining away".

The phenomena of "explaining away" relates to the fact that two independent nodes in a Belief Network becomes dependent on each other if a shared child node is observed. When nodes are dependent they do not factorize, which makes inference difficult. For an in depth explanation of "explaining away" we refer to [Bishop, 2009, p. 377].

As seen from Fig. 5.6 a DBN consists of a bottom layer of observed variables and $L$ layers on top consisting of hidden variables. One would then expect that when the variables in the first layer are observed, the hidden variables in the first hidden layer become dependent, hence being hard to train since the conditional distribution over hidden variables given the observed variables do not factorize due to "explaining away". This is in fact also correct, but for Sigmoid Belief Networks, which are Belief Networks with binary units, the "explaining away" can be counteracted by a concept proposed by Geoffrey Hinton known as "complementary priors" [Hinton et al., 2006].

A complementary prior is a prior distribution which is used to counteract the effect of "explaining away" hence making an apparent non-factorial distribution factorial. In a DBN the complimentary prior is defined as the RBM at the top of the DBN and is given by the last $p(\mathbf{h}^{(L-1)}, \mathbf{h}^{(L)})$ term in Eq. (5.37). This RBM can be viewed as a perfect complimentary prior if the DBN consists of only one hidden layer below the RBM. If additional hidden layers are introduced, with different wights, the RBM does not provide a correct complimentary prior and the conditional distributions forming the lower layers will not factorize properly. However, it can be shown that having a RBM as the top two layers of a DBN has the effect that the conditional distributions of all the layers below can be considered approximately factorial, which leads to a greedy layer-wise training procedure of deep belief nets as presented next [Hinton et al., 2006].

## 5.4.1 Generative Training of Deep Belief Networks

The main purpose of the DBN is to train the network in an unsupervised fashion such that its weights can be used as initialization of a DNN, on which supervised training can be conducted. This unsupervised training can be conducted in a greedy layer-wise fashion, one layer at a time, due to the concept of complementary priors, which makes the conditional distribution of each layer approximately factorial. The concept of the greedy layer-wise training procedure is to stack RBMs on top of each other and then train them individually using the state of one layer as the data for the next layer. In Fig. 5.7 a three layer DBN is illustrated. The DBN illustrated in Fig. 5.7 consists of one layer of visible units $\mathbf{v}^{(1)}$ and three layers of hidden units $\mathbf{h}^{(1)}$, $\mathbf{h}^{(2)}$ and $\mathbf{h}^{(3)}$. As a direct consequence, the DBN also consists of three sets of weights $\mathbf{W}^{(1)}$, $\mathbf{W}^{(2)}$ and $\mathbf{W}^{(3)}$. The procedure of the greedy layer-wise training is to split the DBN into three individual RBMs, as shown in Fig. 5.7, and then train the

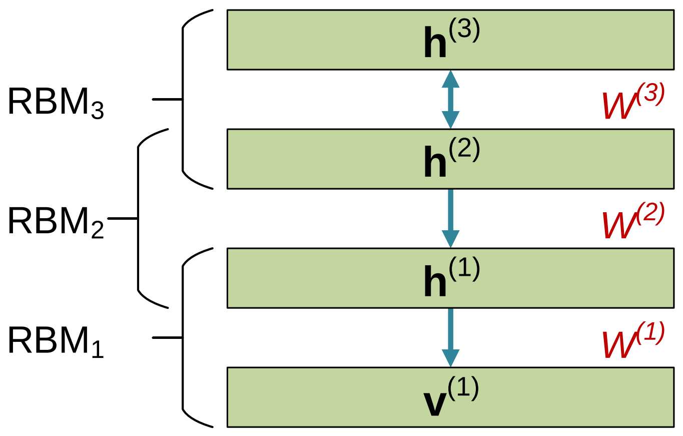


**Figure 5.7:** Training a DBN using the procedure of greedy layer-wise training where each pair of layers are considered as a RBM each being trained using CD.

RBMs one after another using for example CD.

In the current case three RBMs are defined. One RBM consists of $\mathbf{v}^{(1)}$ and $\mathbf{h}^{(1)}$ having $\mathbf{v}^{(1)}$ as the visible units and $\mathbf{h}^{(1)}$ as the hidden units. In a similar fashion the second and third RBM is defined. The first RBM will use the actual training data when being trained, but the second and third RBM will be using the state of $\mathbf{h}^{(1)}$ and $\mathbf{h}^{(2)}$ respectively as observed data.

Training a DBN in this manner can be seen as having feature detectors represented by RBMs each gradually constructing more and more abstract features based on the features from the RBM below. In this way the first RBM detects basic and low level features, where the topmost RBM construct abstract features [Bengio, 2009, p. 5] Bengio et al. [2007] Larochelle et al. [2009]. When all the RBMs have been individually trained the DBN represents a generative model suitable for fine tuning using supervised learning such as backpropagation as described in the next section.

### 5.4.2 Discriminative Training of Deep Belief Networks

When a DBN has been trained in an unsupervised fashion, as described in the former section, the DBN can be interpreted as a DNN with good initial weights [Bengio, 2009, p. 32]. These weights generally serve as a better starting point, than random initialized weights, hence resulting in better classification performance when trained in a supervised fashion using backpropagation [Hinton et al., 2006].

Since the DBN has no output layer an output layer has to be added before it can be considered an ordinary DNN. A typical choice of activation function for the units in the output layer is either the sigmoid activation function or the softmax activation function. The DBN with the added output layer and interpreted as a deterministic and ordinary DNN is illustrated in Fig. 5.8. The network in Fig. 5.8 is now a DNN

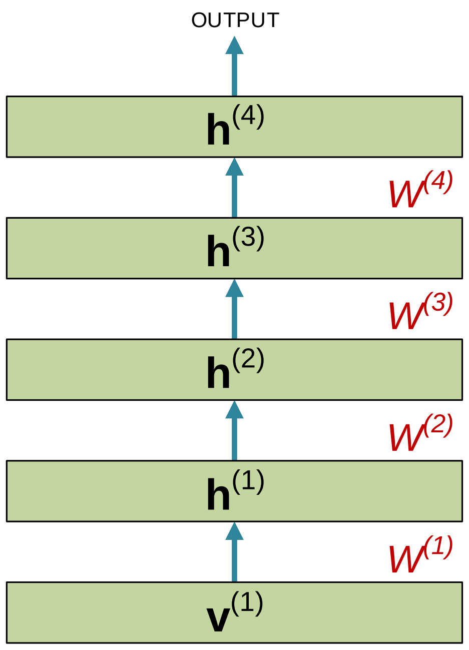


**Figure 5.8:** A deep neural network having one input layer and three hidden layers and one output layer. The weights in the three hidden layers are produced by generatively training a DBN. The weights in the output layer are found by discriminative training using backpropagation.

initialized with the weights from the DBN. The output layer has introduced some weights which are adjusted using backpropagation. The output layer can be viewed as a single layer classifier that is used on some advanced features produced by the layers below [Wang and Wang, 2013].

Using the procedures described in this chapter a DNN can be constructed and trained with a semi-supervised training procedure. In this way, non-labeled data can be efficiently utilized in an unsupervised training fashion and labeled data can likewise be efficiently utilized in a supervised fine tuning process. A similar approach was used in [Healy et al., 2013] to perform speech enhancement and a similar approach will be used in the subsequent chapters in order to investigate the performance of DNNs, when used for speech enhancement.

Chapter 6

# Speech Enhancement Using Deep Neural Networks

This chapter presents a speech enhancement technique based on the method of Ideal Binary Masks (IBMs) and Deep Neural Networks (DNNs). The chapter covers the design as well as the implementation and evaluation of the system using these techniques. Additionally, an algorithmic analysis of the system is given in the end of the chapter. The speech enhancement technique is inspired by the approach described in [Healy et al., 2013]. Figure 6.1 shows a system overview on how this technique performs speech enhancement, as well as being trained.

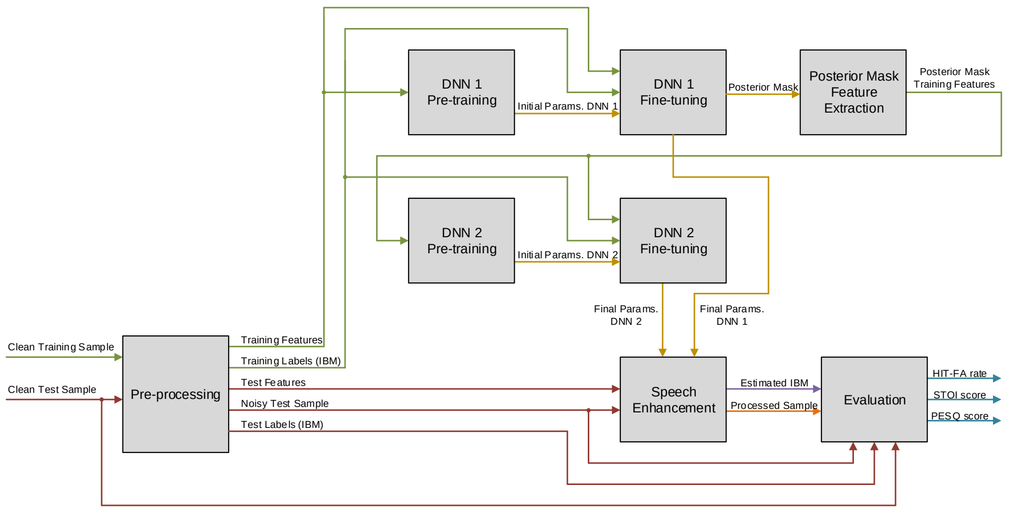


**Figure 6.1:** This figure presents a system overview of the speech enhancement presented in this thesis. The system is based on the method of the IBM and DNNs are used as classifiers for estimating the IBM

## 6.1 System Overview

As seen from Fig. 6.1 the speech enhancement system consists of eight major subsystems. A pre-processing subsystem, two pre-training subsystems, two fine-tuning subsystems, a posterior mask feature extraction subsystem, a speech enhancement subsystem and an evaluation subsystem.

The system works as follows. First, in the pre-processing subsystem clean speech is converted into noisy speech, by adding noise, and then transformed into speech features and labels. The training features are then used to pre-train and fine-tune DNNs as illustrated by the *DNN 1 Pre-training* and *DNN 1 Fine-tuning* subsystems. The output of the *DNN 1 Fine-tuning* subsystem is a posterior mask and DNN parameters such as weights and biases. The parameters are passed to the *Speech Enhancement* subsystem and the posterior mask is passed to the *Posterior Mask Feature Extraction* subsystem. The *Posterior Mask Feature Extraction* subsystem generates features, which are used to pre-train and fine-tune a second DNN as illustrated by the *DNN 2 Pre-training* and *DNN 2 Fine-tuning* subsystems. The output of the *DNN 2 Fine-tuning* subsystem are DNN parameters, which are passed to the *Speech Enhancement* subsystem. The *Speech Enhancement* subsystem uses the model parameters from the *DNN 1 Fine-tuning* and *DNN 2 Fine-tuning* subsystems to initialize DNNs, which are then used to estimate the IBM based on a test sample. Finally, the performance of the system is evaluated by the *Evaluation* subsystem using the performance measures presented in Section 3.3.

When a training or test sample is referred to, a speech utterance from the Hearing In Noise Test (HINT) speech corpus is meant. This corpus is further described in Section 6.2. Each of the subsystems just presented will now be further elaborated in the next couple of sections.

### 6.1.1 Pre-processing

The purpose of the preprocessing block is to construct noisy training and test samples. Moreover, the purpose of this block is to compute features and labels from the clean and noisy samples. The features and labels are then either sent to the speech enhancement subsystem or to the pre-training and fine-tuning subsystems.

As seen from Fig. 6.2 the pre-processing subsystem has one input an two outputs and it consists of a noise generation block, T-F transform blocks, a feature extraction block and an IBM block.

The pre-processing of a training sample and a test sample is similar, hence the pre-processing block shown in Fig. 6.2 only has one input and two outputs and not two inputs and four outputs as shown in Fig. 6.1. The output labeled *Features* consists of a feature representation of a noisy speech sample, being either a training or test sample. The output labeled *IBM* consists of an IBM made out of the clean speech sample and the noise sample generated by the noise generator.

The noise generator adds noise to the clean speech samples at a certain SNR. The type of noise and the SNR varies depending on the scenario that is being investigated

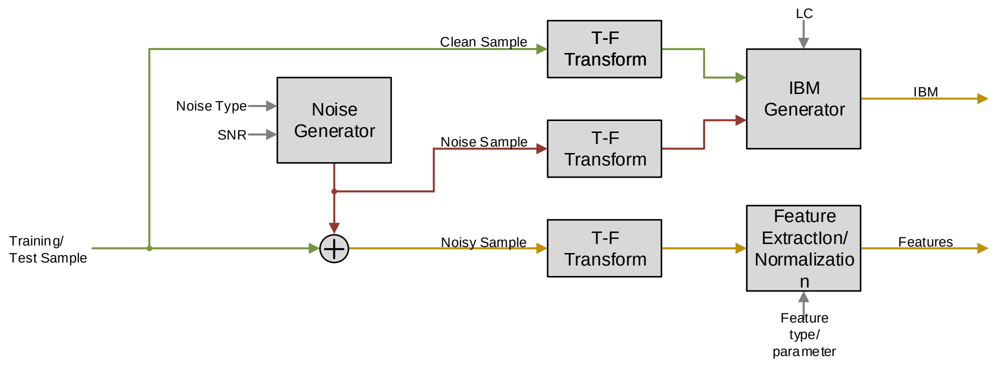


**Figure 6.2:** This figure presents the preprocessing subsystem, which is a subsystem of the speech enhancement system presented in Fig. 6.1. The preprocessing subsystem constructs features and labels for training the DNNs

and is elaborated further in Section 6.2. The noisy sequences are constructed such that the noise begins approximately 140 ms prior to the beginning of the speech signal and ends approximately 140 ms after the end of the speech signal [Healy et al., 2013]. The clean speech signal is padded with 140 ms of zeros in the beginning and in the end in order to ensure that the clean signal and the noisy signal are equal in length.

To construct the data for the output labeled *Features* a T-F representation of the noisy speech sample is generated by the T-F transform block. The T-F representation is made using a gammatone filter bank as the one illustrated in Fig. 2.4 and each frequency channel is divided into overlapping frames. The noisy speech signal is now in a time-frequency representation where each frame, in each frequency channel, consists of a number of filtered noisy speech samples. The number of frequency channels used, the frame length as well as overlap can differ and is a trade off between temporal and spectral resolution and computational complexity. A typical choice is to use from 25 to 64 frequency channels with a frame length from 20 to 32 ms and an overlap from 10 to 16 ms. [Healy et al., 2013] [Kim et al., 2009] [Hang and Wang, 2012]. The speech enhancement system presented in this chapter has been based on 64 frequency channels and a frame length of 20 ms with 10 ms overlap.

The T-F representation of the noisy speech sample is then processed by the feature extraction block, which generates a feature vector for each T-F unit in the T-F representation. The features used are MFCCs, AMSs and RASTA-PLP as described in Section 2.5, which are similar to the once used in [Healy et al., 2013]. The feature vectors across each frequency channel is then normalized to unit variance as assumed by Eq. (5.9) with $\sigma_j^2 = 1$. The features are also normalized to have zero mean as suggested in [Hinton, 2012]. This representation is now ready to be used for either training a DNN or performing speech enhancement using an already trained DNN.

To construct the data for the output labeled *IBM*, a T-F transformation, similar to the one created of the noisy speech sample, is produced from the clean speech sample as well as the noise signal in isolation. Using these two T-F representations an IBM is created based on a Local Criterion (LC) as described in Section 3.2. In the case of a training sample the IBM is used as labels for the discriminative training of a DNN and in the case of a test sample the IBM is used for evaluating the IBM estimation performance of a DNN.

## 6.1.2 Pre-training

The pre-training subsystems, as illustrated in Fig. 6.3, is where DBNs are trained, one for each frequency channel. In the *DNN 1 Pre-training* subsystem the training data is a noisy speech sample, represented as feature vectors in a T-F representation, and in the *DNN 2 Pre-training* subsystem the training data is feature vectors consisting of posterior probabilities generated by the *Posterior Mask Feature Extraction* subsystem.

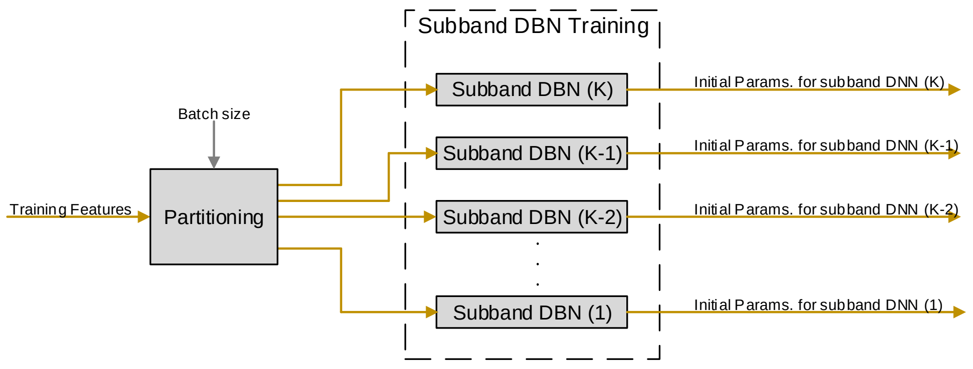

Subband DBN Training
Batch size
Training Features
Partitioning
Subband DBN (K)
Subband DBN (K-1)
Subband DBN (K-2)
Subband DBN (1)
Initial Params. for subband DNN (K)
Initial Params. for subband DNN (K-1)
Initial Params. for subband DNN (K-2)
Initial Params. for subband DNN (1)

**Figure 6.3:** This figure presents the Pre-training subsystem, which is a subsystem of the speech enhancement system presented in Fig. 6.1. The pre-training subsystem consists of DBNs, which are trained and used for initializing the parameters of the DNNs shown in Fig. 6.1

As seen from Fig. 6.3 the pre-training subsystem consists of a partitioning block and some subband DBN training blocks. The partitioning block divides the training data into batches or mini-batches, which are used to train each of the DBNs. Each of the subband DBNs consists of several layers of hidden units and are trained and constructed in a similar way as described in Section 5.3 and Section 5.4. When the pre-training is completed a set of initial model parameters have been created, which are transfered to the fine-tuning block for training each of the $K$ subband DNNs.

### 6.1.3 Fine-tuning

The fine-tuning subsystems, as illustrated in Fig. 6.4, is where the weights from the trained subband DBNs are used as initial weights in subband DNNs. Since the DBNs have no output layer, the DNNs are one layer deeper than the DBNs due to the output layer of the DNNs. The output layer consists of 2 output units, one for a speech dominated T-F unit and one for a noise dominated T-F unit. The activation function for the output units is the softmax function such that the output for both output units sum to one. This means that it can be treated as a probability. The discriminative training or fine-tuning of the subband DNNs are performed using the backpropagation algorithm using the cross-entropy error function. The output from the *DNN 1 Fine-tuning* subsystem is posterior mask probabilities, which are used to train a second DNN. The output from the *DNN 2 Fine-tuning* subsystem is a set of DNN weights and biases, which are sent to the *Speech Enhancement* subsystem.

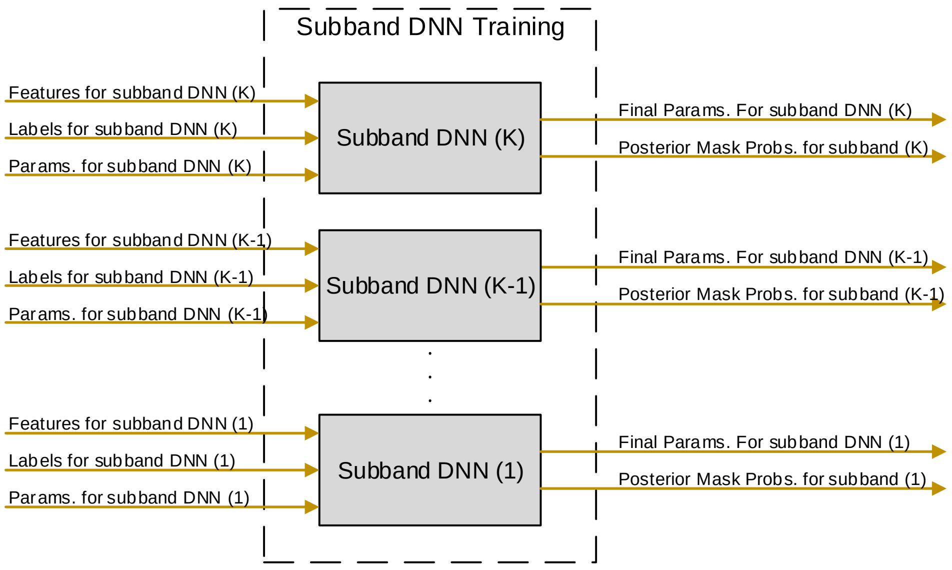

Subband DNN Training
Features for subband DNN (K)
Labels for subband DNN (K)
Params. for subband DNN (K)
Subband DNN (K)
Final Params. For subband DNN (K)
Posterior Mask Probs. for subband (K)
Features for subband DNN (K-1)
Labels for subband DNN (K-1)
Params. for subband DNN (K-1)
Subband DNN (K-1)
Final Params. For subband DNN (K-1)
Posterior Mask Probs. for subband (K-1)
Features for subband DNN (1)
Labels for subband DNN (1)
Params. for subband DNN (1)
Subband DNN (1)
Final Params. For subband DNN (1)
Posterior Mask Probs. for subband (1)

**Figure 6.4:** This figure presents the Fine-tuning subsystem, which is a subsystem of the speech enhancement system presented in Fig. 6.1. The fine-tuning subsystem consists of DNNs, which are trained using the backpropagation algorithm and used for estimating IBMs.

### 6.1.4 Posterior Mask Feature Extraction

The *Posterior Mask Feature Extraction* subsystem is used to convert a mask consisting of posterior probabilities into feature vectors representing T-F units. The posterior mask generated by the *DNN 1 Fine-tuning* subsystem is identical in size to the corresponding IBM and each T-F unit in the posterior mask is a probability indicating how likely a given T-F unit is to be speech dominated. The idea of using features based on a posterior mask is to incorporate information about the spectro-temporal patterns, which are present in human speech [Healy et al., 2013]. The features incorporating this spectro-temporal information is constructed by row-wise concatenation of the probabilities within a window spanning 5 time frames and 17 frequency channels of the posterior mask. The label that corresponds to the feature vector of posterior probabilities is the state of the T-F unit in the IBM that corresponds to the coordinate of the T-F unit in the center of the window. The concept of the posterior mask and how a given feature vector is constructed is illustrated in Fig. 6.5. In Fig. 6.5 a subsection of a fictitious posterior mask and a correspond-

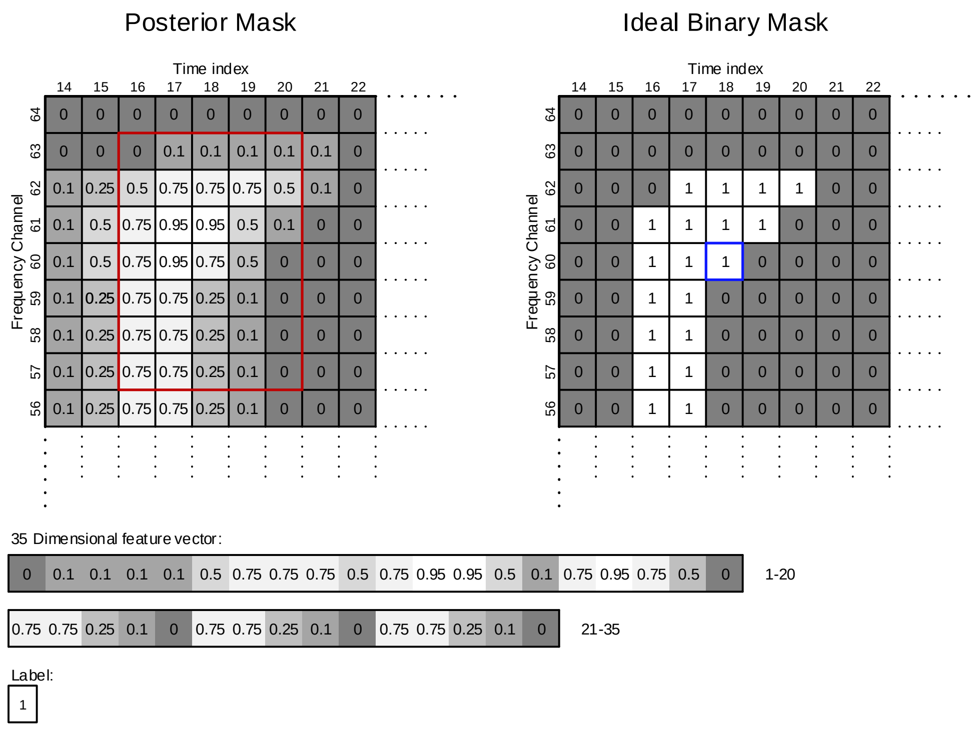


**Figure 6.5:** This figure presents the *Posterior Mask Feature Extraction* subsystem, which is a subsystem of the speech enhancement system presented in Fig. 6.1. The *Posterior Mask Feature Extraction* construct features of posterior probabilities based on the output of DNNs.

ing fictitious IBM is illustrated. The window which encapsulates the probabilities, making up a given feature vector, is represented by the red square on the posterior mask and the corresponding feature vector is shown in the bottom of Fig. 6.5. The window size is reduced to cover only 7 frequency channels instead of 17 to keep the illustrations and examples simple. Hereby, the feature vector has dimension 35 and is constructed by row-wise concatenation of the probabilities encapsulated by the window starting from the top row. The label corresponding to the feature vector is seen from the IBM to the right where it is encapsulated by a blue square, which is the position corresponding to the center of the red window. Feature vectors representing the whole IBM is constructed by sliding the window one T-F unit at the time covering all T-F units in the posterior mask except at the border conditions, which are at the three lowest and highest frequency channels and at the two first and last time indexes.

The first border condition is handled by altering the shape and size of the window such that the row in the window otherwise "exceeding" the posterior mask is ignored. This means that the size of the feature vectors for the T-F units in frequency channel 3 and 62, 2 and 63 and 1 and 64 have dimension 30, 25 and 20, respectively. This is considered acceptable since the size and the shape of the windows are constant in each frequency channel.

The second border condition is at the two first and two last time indexes. This is handled by ignoring the first two and last two T-F units in the posterior mask, hence slightly reducing the number of features. This is primarily done to ensure that the window size and shape is constant within each frequency channel and because the first 13 time units in each frequency band is guaranteed to be noise dominated. Therefore, leaving out the first two and last two time indexes in the posterior mask, is considered acceptable.

### 6.1.5 Speech Enhancement

The Speech Enhancement subsystem as illustrated in Fig. 6.6 is where a noisy speech signal is being processed using the method of the IBM and DNNs.

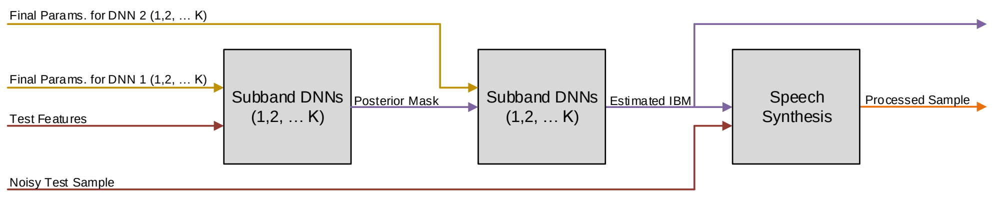

Final Params. for DNN 2 (1,2, ... K)
Final Params. for DNN 1 (1,2, ... K)
Test Features
Noisy Test Sample
Subband DNNs (1,2, ... K)
Posterior Mask
Subband DNNs (1,2, ... K)
Estimated IBM
Speech Synthesis
Processed Sample

**Figure 6.6:** This figure presents the speech enhancement subsystem, which is a subsystem of the speech enhancement system presented in Fig. 6.1. The speech enhancement subsystem is where an IBM is estimated using DNNs and afterwards applied to a noisy speech sample to attenuate the noise.

The IBM is estimated using two sets of subband DNNs. The first set of subband DNNs estimate the posterior mask by forward propagating acoustic feature vectors representing T-F units of noisy speech. The second set of subband DNNs estimates the IBM by forward propagating feature vectors representing a T-F unit via a window of posterior mask probabilities. The output of a subband DNN is a probability indicating how likely a given T-F unit, represented as a feature vector, is to be noise dominated or speech dominated. Whether or not a given T-F unit is classified as noise or speech dominated is based on the output having the largest probability.

When all T-F units have been classified the IBM has been estimated. To apply the estimated IBM to the noisy speech signal each entry in the estimated IBM, being either zero or one, is multiplied with all noisy speech samples within the corresponding frequency channel and frame. If 20 ms frames with 10 ms overlap and 16 kHz sampling frequency are used, this is equivalent to 320 samples which are either preserved or discarded by the estimated IBM. Due to the overlap each frame shares 160 samples with their neighbor frames.

From this procedure ideally all noise dominated segments are discarded having only the speech dominated segments left. The processed T-F representation of the noisy speech is then synthesized using the overlap and add technique, which is the addition of all the frames across frequency and time windowed using an appropriate window function as described in [Wang and Brown, 2006, 23]. The speech signal is then synthesized into a time-domain signal and is ready to be evaluated by the evaluation subsystem.

When the system is tested using the soft mask approach as described in the introduction in Chapter 1, the classification stage just described is omitted and the output of the DNNs are directly multiplied to all samples of the T-F representation of the noisy speech signal.

### 6.1.6 Evaluation

The Evaluation subsystem, as illustrated in Fig. 6.7, is where the performance of the speech enhancement block is evaluated, which is done using HIT-FA rates, PESQ and STOI objective performance measures. The PESQ measure is known to correlate well to speech quality and the STOI measure is known to correlate well to speech intelligibility as elaborated in Section 3.3. From these three measures the system will be evaluated in terms of both quality and intelligibility. In order to have a reference for STOI and PESQ computed for the estimated IBM processed sample, these measures are also computed for the IBM processed sample as well as the noisy sample. This enables the STOI and PESQ score from the noisy speech signal, processed by the estimated IBM, to be related properly. In total this gives nine different scores per test sample. The HIT, FA and HIT-FA rates and the STOI and PESQ scores for the noisy signal, the IBM processed signal and the estimated IBM processed signal as shown on Fig. 6.7.

## 6.2 Training & Test Data

Before describing how the different subsystems are being implemented and tested a short description of training and test data as well as different noise types is given.

Training and testing the system illustrated by Fig. 6.1 requires some speech data. Training data is required to train the system and test data is required to investigate

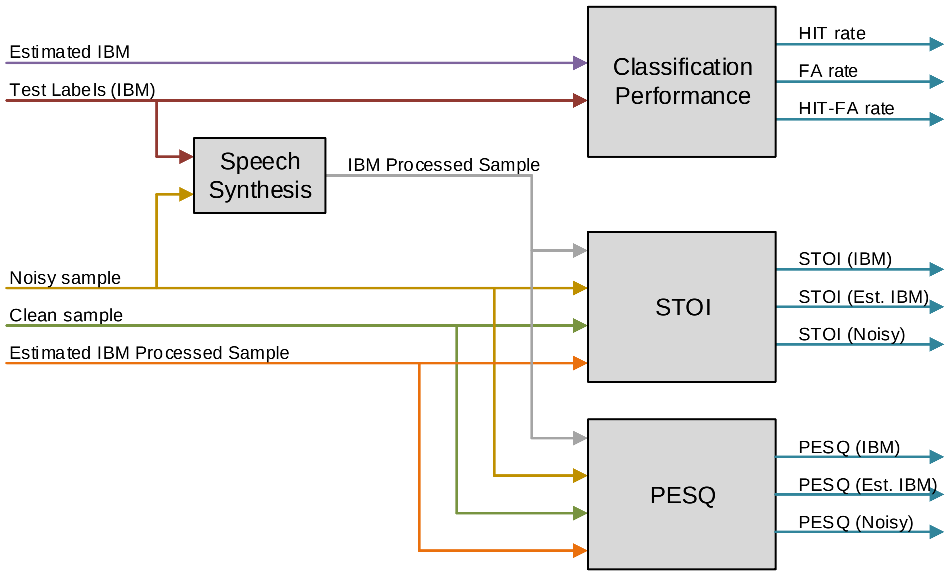


**Figure 6.7:** This figure presents the evaluation subsystem, which is a subsystem of the speech enhancement system presented in Fig. 6.1. The evaluation subsystem is where the speech enhancement system is evaluated using the HIT rate, FA rate, HIT-FA rate as well as STOI and PESQ.

how well the system generalizes to unseen input. In speech processing applications, datasets known as speech corpora are commonly used. A speech corpus is a dataset consisting of spoken sentences where both the sentences themselves as well as the speakers are carefully chosen to fulfill some specific requirements.

Regarding the speaker these requirements can be related to the gender of the speaker, the age and the dialect and or nativity. Regarding the sentences there can be requirements such that only voiced or unvoiced speech is present or it can be such that the phonetic content is balanced in relation to the distribution of normal conversational speech at the current language [Loizou, 2013, p. 445]. It is also important, especially regarding intelligibility testing, that the predictability of the words are considered since predictability and intelligibility can be difficult to separate in the scoring from a listening test.

Since speech enhancement algorithms aim at being effective at enhancing noisy conversational speech, phonetically balanced corpora are typically used for training and testing. A commonly used speech corpus is the Texas Instruments Massachusetts Institute of Technology (TIMIT) corpus. The TIMIT corpus consists of a total of 6300 spoken sentences divided between 630 speakers, 70% male and 30% female, each speaking 10 sentences. The speakers cover 8 major dialect regions of the United States and a total of 2342 different sentences are used [Garofolo et al., 1993].

Another commonly used approach in the speech enhancement literature is to use own recordings of speakers but based on a common accepted list of sentences. Two commonly used of such lists are the HINT sentences and the IEEE sentences, which are also known as the Harvard sentences [Nilsson et al., 1994] [Rothauser et al., 1969]. HINT consists of 260 sentences and IEEE consists of 720 sentences both of which are considered to be phonetically balanced.

In order to construct training and test data which are applicable for training and testing machine learning based speech enhancement algorithms, noise sources must be decided. The noise source must be realistic and match the scenario one wants to investigate. A commonly used noise corpus is Aurora, which consists of different real life noise recordings [Pearce et al., 2000]. Another approach is to construct a noise source based on filtered white noise such as Speech Shaped Noise (SSN) or by mixing audio files from speech corpora such as TIMIT making competing talker noise or, if more speakers are present, babble noise.

The general difficulty when constructing training and test data, for machine learning based techniques, is to generate enough data such that the training set and test set are completely unique regarding both the speech signal and the noise signal and still large enough to train and test the algorithm properly. In several papers the limitation of especially noise material lead to the use of the same noise in both the training and testing data [Healy et al., 2013] [Hang and Wang, 2012] [Kim et al., 2009]. Using the same noise sequence for both training and testing may enable the classifier to learn the noise sequence which will lead to an artificially high classification performance since the real generalization capability of the classifier is not being tested. This is also pointed out in [May and Dau, 2014] where it is shown that the performance reported by [Kim et al., 2009] only applies for known noise realizations and not unseen noise realizations.

Since the speech enhancement system illustrated in Fig. 6.1 is inspired by [Healy et al., 2013] the same speech and noise material will be used for evaluation and initial testing in order to be as close to the system described in [Healy et al., 2013] as possible. This is done only to be able to compare the results produced by the system from Fig. 6.1 with the results reported in [Healy et al., 2013]. When it has been verified that the implementation of the system from Fig. 6.1 is correct, more realistic training and testing material will be used to evaluate the performance of the system.

In [Healy et al., 2013] the 260 HINT sentences are used which has been spoken by a male. All 260 sentences have been normalized to have unit RMS power [Healy et al., 2013]. The noise is based on a 5 s babble sequence and a 10 s SSN sequence which are used for both training and test data. In [Healy et al., 2013] the term "looped noise" is assumed to mean that the noise sequence is just used repeatedly, i.e. replicated, until all clean training and test samples have been corrupted by noise. This means that if 4000 s of training and test data is used the babble noise sequence is replicated 1000 times in order to fit the length of the data set. Similarly, this applies when the 10 s SSN sequence is used.

This also means that in the rest of the chapter, when looped noise is mentioned, it is the 5 s babble noise sequence which is replicated an appropriately number of times to fit the size of the training and test set. The 5s babble noise sequence and the spoken HINT sentences are public available from [Wang, 2013] and will form the foundation for the training and test data in the rest of this chapter.

### 6.2.1 Unseen Noise Realizations

In order to train and test the system in Fig. 6.1 in a more realistic scenario, i.e., without the use of looped noise, unseen noise realizations must be constructed. It has been decided to use the same kind of noise as in [Healy et al., 2013], which are babble noise and SSN.

#### Babble Noise

Babble noise is a noise source which consists of competing talkers, male or female. Babble noise is a way to simulate the scenario of a cocktail party and is considered as a difficult speech enhancement problem since the noise source is similar to the target speaker as well as being non-stationary, i.e., the PSD of consecutive 20 ms frames might differ.

The babble noise source used for non-looped noise realizations in the simulations presented in Chapter 7 is constructed from the TIMIT corpus. The TIMIT corpus consists of 6300 sentences or roughly 300 min. of speech and the babble noise sequence has been constructed by randomly dividing these 6300 sentences into six groups each consisting of 1050 sentences or roughly 50 min. of speech. These six groups are then added together forming a 6-speaker babble sequence with a duration of approximately 50 min. In [Healy et al., 2013] a similar approach was used but as mentioned in the former section, only a babble sequence of 5 s was constructed, which led to the use of looped noise. Having a babble noise sequence of 50 min. it is possible to construct training and test data having the same size as in [Healy et al., 2013] but with unique and unseen, i.e., non-looped noise realizations.

In [Kim et al., 2009] a 20-speaker babble sequence was used and in [Healy et al., 2013] an 8-speaker babble sequence was used. In [Healy et al., 2013] it is argued that an 8-speaker babble sequence is a more difficult noise signal than a 20-speaker babble sequence. Accordingly, it is assumed that the 6-speaker babble sequence constructed by the approach just presented must be even more difficult. However, since only a limited amount of speech material is available it is decided to use the 6-speaker babble sequence based on the TIMIT corpus as the noise source for simulations using non-looped noise.

#### Speech Shaped Noise

Speech Shaped Noise (SSN) is a term used to describe a noise sequence which has a PSD similar to the long term PSD of speech. SSN is constructed by filtering a white Gaussian noise sequence with a filter similar to the all-pole system given by

Eq. (2.12). The coefficients of the filter are found by minimizing a LP error term, which is equivalent to solving the normal equations as given by Eq. (2.14).

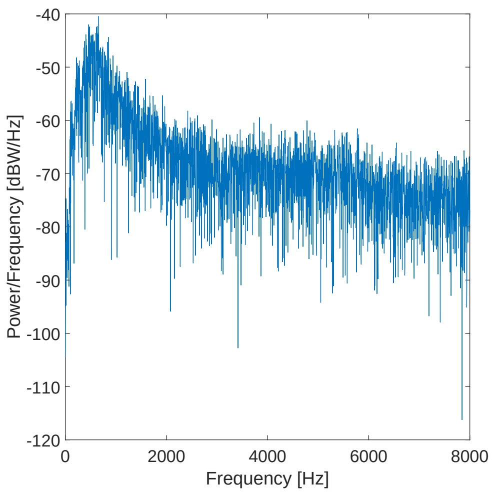
The SSN noise source used for non-looped noise realizations in the simulations presented in Chapter 7 is constructed in MATLAB using the `lpc`, `wgn` and `filter` functions. First 12 LP coefficients are found based on 100 randomly chosen TIMIT sentences using the `lpc` function. Then a 1-hour white Gaussian noise sequence with unit variance is generated using the `wgn` function. To avoid aliasing the white Gaussian noise is band-limited to 8 kHz using a 6'th order Butterworth filter. This 8 kHz band limited Gaussian noise sequence is then filtered by an all-pole filter using the 12 LP coefficients and the `filter` function. The filtered noise is then having a periodogram similar to the long term periodogram of the 100 TIMIT sentences and is hereby known as SSN. Plots showing the different stages are depicted in Fig. 6.8. Figure 6.8a shows the periodogram of the 100 TIMIT sentences and the corresponding all-pole system generated using LP is shown in Fig. 6.8b. It is seen that the all-pole system has captured the characteristics of the spectrum in Fig. 6.8a. In Fig. 6.8c the periodogram of the 1 hour 8 kHz band-limited white Gaussian noise signal is depicted and in Fig. 6.8d the same signal is shown after it has been filtered using the all-pole system of Fig. 6.8b. It is seen that the periodogram of the filtered white Gaussian noise is "shaped" very similar to the periodogram of the speech signal based on the 100 TIMIT sentences, which is exactly what is desired when constructing SSN.

## 6.3 System Implementation & Evaluation

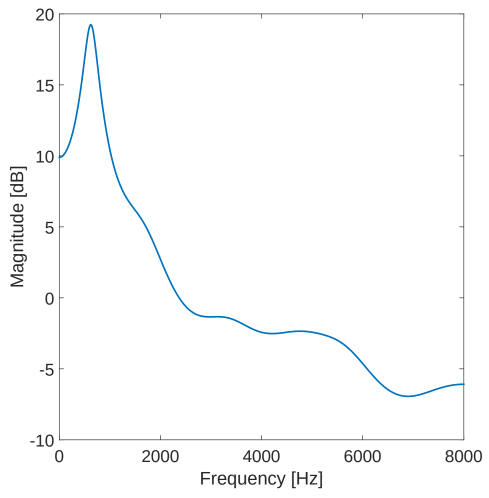
This section describes how the speech enhancement system presented in Section 6.1 is implemented and tested. The speech enhancement system is implemented in MATLAB and the implementation is a combination of public available code, toolboxes, and code written by the authors. The MATLAB code can be found on the appendix DVDs and an overview of the implementation can be found in Appendix D.

In order to minimize the risk of software errors the implementation of the speech enhancement system is, as far as possible tested on a unit level. The code and toolboxes, which are public available and was not modified in any way, are used as is. This means that no effort has been put into verifying their implementation. This relates to the extraction of MFCC and RASTA-PLP features as well as the objective performance measures PESQ and STOI. MFCC and RASTA-PLP features are extracted using the toolbox RASTAMAT, which is public available from [Ellis, 2005]. This toolbox is also used by DeLiang Wang in [Wang et al., 2012] for extracting MFCCs and RASTA-PLP features. PESQ is available from [Loizou, 2013] and STOI is public available from [Taal, 2010]. It is hereby assumed that the aforementioned implementations are correct.

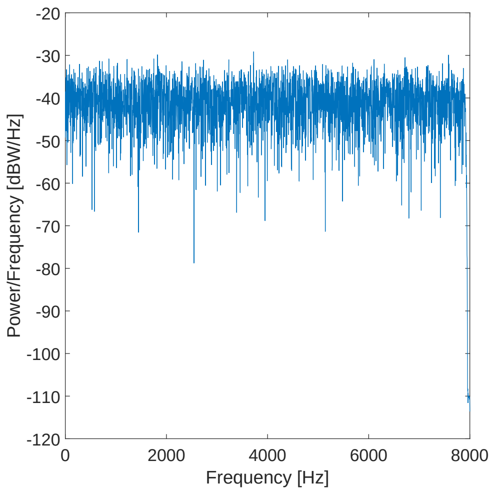
What is left of the speech enhancement system is the extraction of the AMS features, the IBM creation, speech analysis and synthesis and DNN training and pre-training. These parts of the speech enhancement system have been written partly or



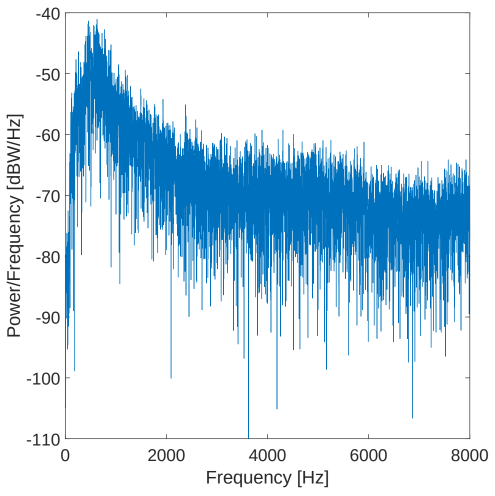
**(a)** Periodogram of 5 min. speech signal based on 100 TIMIT sentences



**(b)** Amplitude Reponse of LP all-pole system



**(c)** Periodogram of 0 dBW, 8 kHz band-limited white Gaussian noise



**(d)** Periodogram of 1 hour speech shaped noise signal

**Figure 6.8:** This figure presents the stages involved when constructing speech shaped noise. In Fig. 6.8a the periodogram of 100 TIMIT sentences is shown and the corresponding all-pole system generated using LP is shown in Fig. 6.8b. It is seen that the all-pole system has captured the characteristics of the spectrum in Fig. 6.8a. In Fig. 6.8c a 1 hour 8 kHz band-limited white Gaussian noise signal is depicted and in Fig. 6.8d the same signal is depicted when it has been filtered using the all-pole system shown in Fig. 6.8a. It is seen that the periodogram of the filtered white noise is "shaped" similar to the periodogram of the speech signal based on the 100 TIMIT sentences, hence a speech shaped noise source has been constructed.

entirely by the authors and hereby require thorough testing, which are described in the next couple of sections.

### 6.3.1 IBM Creation, Speech Analysis & Synthesis Evaluation

This section will describe how the implementation of the IBM construction as well as the procedure of speech analysis and synthesis is performing. This code is primarily written by the authors except for the synthesis part, where an implementation partly written by DeLiang Wang from the department of Computer Science and Engineering, Ohio State university, is used [Jin and Wang, 2007b].

One way to investigate if the speech analysis and speech synthesis functionality has been implemented correctly is to compare a speech signal before and after it has been analyzed and synthesized, i.e., without binary mask processing. If the implementation is working correctly it is assumed that the original signal and the processed signal are almost identical if a sufficiently large number of frequency channels are used in the gammatone filter bank.

The job of the analysis part is to process the speech signal by a gammatone filter bank followed by framing the frequency channels into 20 ms frames with 10 ms overlap. The job of the synthesis part is then to convert the framed T-F representation back to a continuous time signal, which is done using a sum-and-add approach as described in [Wang and Brown, 2006, p. 24].

A natural criterion to measure the similarity of the processed and non-processed speech signal is the sum of squared residuals but since intelligibility is also of a concern, STOI will also be used. To test the implementation three TIMIT sentences and three HINT sentences are being processed with a varying number of frequency channels, varying from 1 to 96.

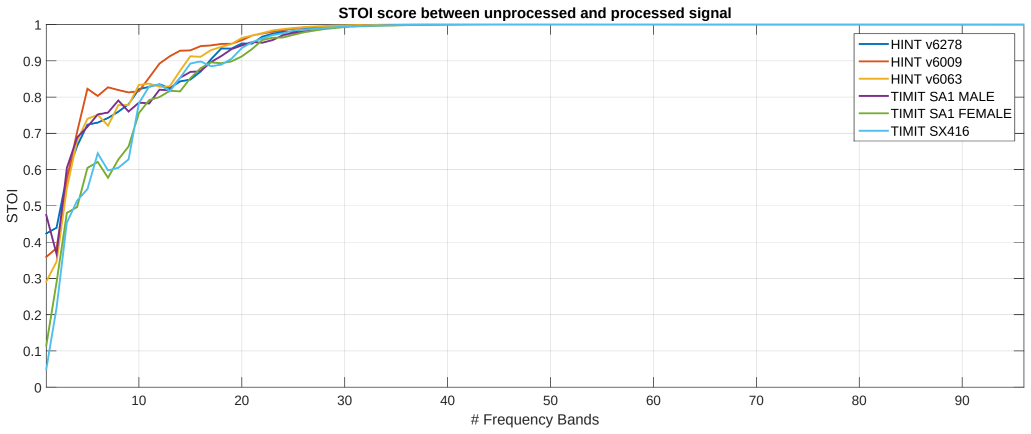
Since 64 frequency channels is used in [Healy et al., 2013] the magnitude response of the whole analyze-synthesize system using 64 frequency channels is simulated. The results are shown in Fig 6.9. From Fig 6.9 the STOI score and the sum of squared residuals is presented for three TIMIT sentences and three HINT sentences. The sum of squared residuals is represented as a SNR computed as

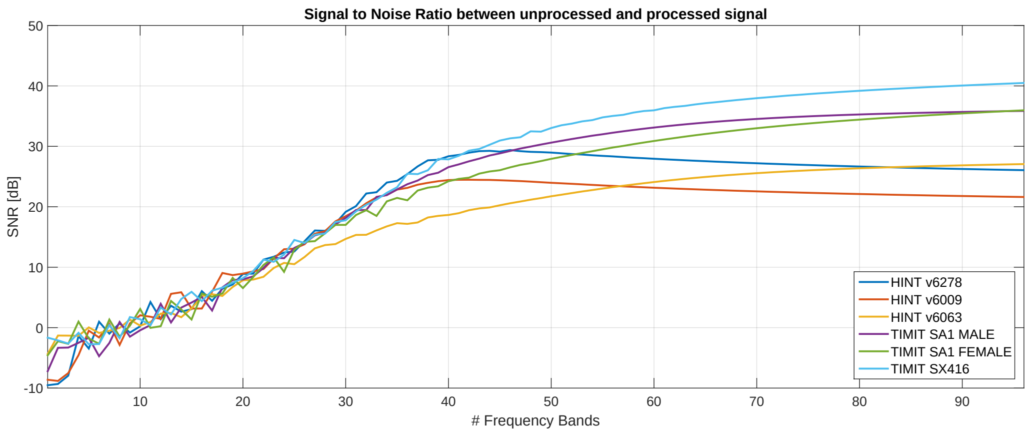
$$SNR = 10 \times log_{10}\left(\frac{\sum_{n=0} x(n)^2_{clean}}{\sum_{n=0}(x(n)_{clean} - x(n)_{processed})^2}\right) \tag{6.1}$$

where $x(n)_{clean}$ is the clean unprocessed speech signal and $x(n)_{processed}$ is the signal which has been processed by the analysis and synthesis stages. It is seen from Fig 6.9 that STOI begins at zero but approaches one as the number of frequency channels goes up and is approximately one, when the number of frequency channels is larger than 40.

Likewise, is it seen that the sum of squared residuals starts low but increases steadily as the number of frequency channels go up and approximately stagnates when more than 60 frequency channels are used.

From the frequency response of the analyze-synthesize system using 64 frequency channels, it is seen that the magnitude response is relatively flat up to 7 kHz with





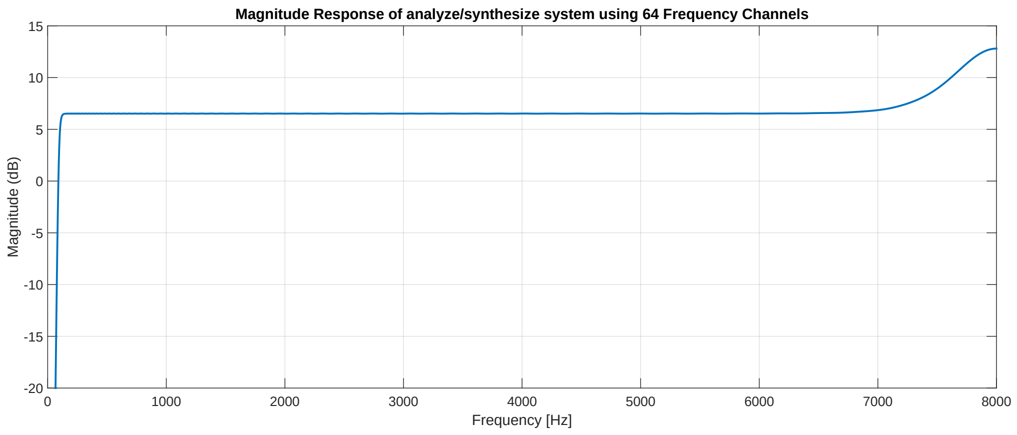


**Figure 6.9:** This figure shows how the speech analysis and synthesis functionality is performing by comparing the a clean speech signal to a signal which has been sent through the gammatone filter bank and afterwards has been re-synthesized. Three TIMIT and three HINT sentences has been analyzed and synthesized using a gammatone filter bank having from 1 to 96 frequency channels. The STOI measure (top figure) and the sum of squared residuals (middle figure) has been used for evaluation and it is seen that after approximately 40 frequency channels a STOI of 1 is achieved and similarly the sum of squared residuals starts to flatten out after approximately 50-60 channels. From the bottom figure the magnitude response of the analyze-synthesize system using 64 frequency channels is presented and it is seen that the response is relatively flat up to 7 kHz with a 6 dB gain. After 7 kHz an unexplainable gain increase occur.

a constant gain of 6.5 dB. Also, it is seen that an approximately 6 dB gain increase is introduced from 7-8 kHz. It has not been possible to identify what causes this gain increase but it has been observed that if the bandwidth of the filter bank is changed this gain increase changes as well. It is hereby assumed that it is due to imperfections/non-linearities in the implementation. It is though considered acceptable since not much speech information is present at 7-8 kHz so a slight distortion at those frequencies will not do much harm. However, since the implementation of the gammatone filter bank is written by DeLiang Wang and hereby presumably the implementation used in [Healy et al., 2013] the filter bank will be used as is.

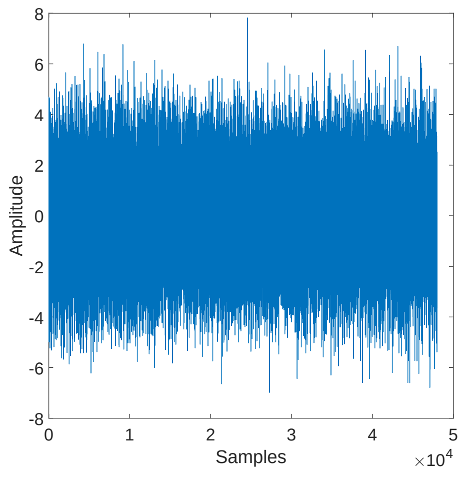
From the plots shown by Fig 6.9 and from supporting listening tests of the processed speech signal it is assumed that the speech analysis and synthesis functionality is working as intended and is correctly implemented.

To investigate if the construction of the IBM is implemented correctly the IBM is constructed from a test signal consisting of different sine waves mixed with Gaussian white noise at 0 dB SNR. The LC is set to 0 dB. The verification of the IBM is then based on visual inspection of the IBM as well as by synthesizing the noisy test signal with the IBM being applied to see if the IBM can remove the Gaussian noise. From visual inspection it is expected that the spectrogram of the noisy test signal and the IBM are similar, in the sense that T-F units in the IBM corresponding to sine wave dominated T-F units in the noisy test signal are set to one and all the noise dominated T-F units being set to zero. When the IBM is applied to the T-F representation of the noisy test signal, each frame in each frequency channel is being either multiplied with zeros or ones depending on the corresponding value of the T-F unit in the IBM, effectively removing the noise dominated T-F units.

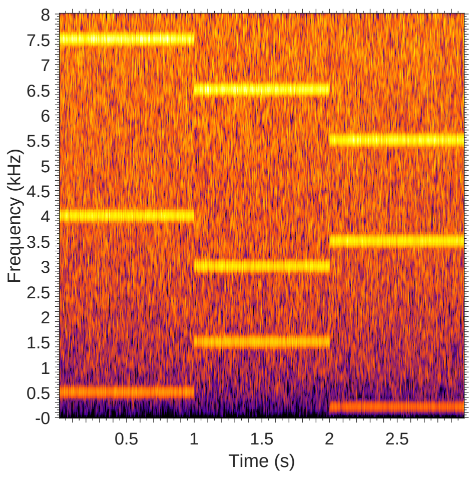
The results from the test just described are presented in Fig. 6.10. A gammatone filter bank with 64 filters have been used. The time domain representation as well as the spectrogram of the noisy test signal are presented in Fig. 6.10a and Fig. 6.10b respectively. The IBM, which is constructed based on the signal in Fig. 6.10a is shown in Fig. 6.10c and the spectrogram of the processed test signal is shown by Fig. 6.10d

It is seen from Fig. 6.10b that 9 distinct signal components are visible in the spectrogram of the noisy test signal, which corresponds to the 9 different sine waves the test signal was made of. It is also seen that the ones, i.e., the white pixels, in the IBM in Fig. 6.10c corresponds very well to the locations of the sine waves in the spectrogram. It is also seen from Fig. 6.10d that the spectrogram of the synthesized test signal contains the 9 sine waves and the majority of the noisy regions have been removed.

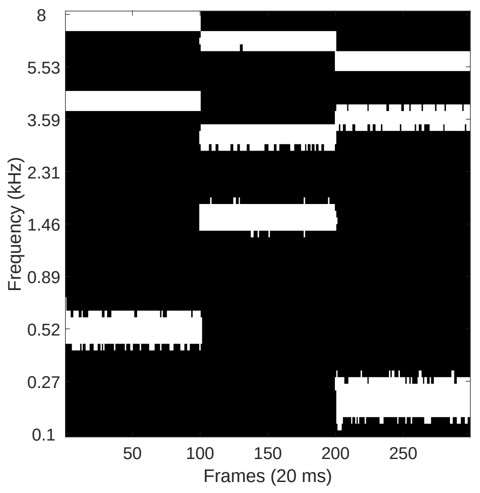
Another thing to notice is the non-linear frequency mapping of the gammatone filter bank, which is seen directly from the frequency axis of the IBM in Fig. 6.10c. It appears as a mismatch between the frequency axis of the spectrogram and the frequency axis of the IBM, but this is not the case and is due to the non-linear frequency mapping of the gammatone filter bank as seen from Fig. 2.4. This means that there are more filters covering the lower part of the spectrum compared to the



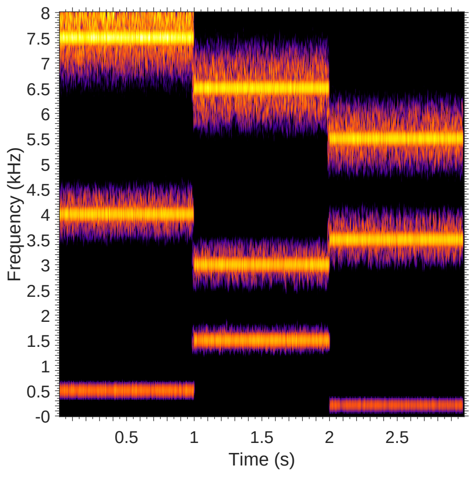
**(a)** Noisy test signal



**(b)** Spectrogram of noisy test signal



**(c)** Ideal Binary Mask



**(d)** Spectrogram of processed test signal

**Figure 6.10:** This figure presents the results of a test used to evaluate the implementation of the construction of the IBM. A synthetic test signal was created based on 9 sine waves with different frequency. A noisy version of the test signal was constructed by adding white Gaussian noise at 0 dB SNR. An IBM was hereby constructed from the clean and the noisy test signal. Figure 6.10a shows the noisy test signal and Fig. 6.10b shows the same signal represented as a spectrogram, where the 9 sine waves a easily seen. Fig. 6.10c shows the constructed IBM and it is seen that the T-F units which are preserved, i.e., white pixels corresponds very well to the locations of the sine waves in Fig. 6.10b. Fig. 6.10d shows the noisy test signal when it has been processed by the IBM and it is seen that the sine waves are more or less preserved. The amount of noise which is preserved as well is due to the non-linearities of the gammatone filter bank. See Fig. 2.4

higher part of the spectrum where only a relatively small number of filters are used. This is also seen on the spectrogram of the synthesized test signal where the sine signals in the low frequencies are very well preserved with almost no visible noise. At higher frequencies, the sine wave are also preserved but likewise is a lot of the noise. This is due to the width of the upper filters of the gammatone filter bank which are larger than the filters in the lower frequencies. A way to circumvent this phenomena is to use more frequency bands, i.e., filters, or to use a linear frequency mapping with the same number of filters. This however leads to either lower spectral resolution at the lower regions of the spectrum, if the same number of filters are used with a linear frequency scale, or higher computational requirements due to the use of more filters having a non-linear frequency mapping. Since most of the energy in speech is at low frequencies, which are also where the ear is most sensitive, a good trade off between spectral resolution and computational complexity is to use the gammatone filter bank. From the above argumentation, the IBM and spectrograms in Fig. 6.10 it is assumed that the IBM construction as well as the synthesis with the IBM being applied is implemented correctly.

### 6.3.2 AMS Feature Evaluation

This section will justify that the implementation of the AMS features are working as intended. As described in section 2.5.3 the implementation of the AMS features extraction is based on the method described in [Hang and Wang, 2012], [Kim et al., 2009] and [Healy et al., 2013].

To verify that the implementation of the AMS feature extraction works as intended, a segment of voiced speech and babble noise was captured. These signal are shown in Fig. 6.11. The pitch period of the voiced segment (top panel) is seen to span over 115 samples. At the sampling frequency of 16 kHz this yields a pitch period of 7.2 ms or a fundamental frequency of 139 Hz. Therefore, it is expected to see activity at the modulation frequencies around the fundamental frequency and its harmonics. From the top panel of Fig. 6.12, activity at the modulation frequency around the fundamental frequency and its first harmonic, is visible. It can also be observed that the activity on the modulation axis is relative broad. This is due to the short frame of 20 ms, which yields a limited modulations frequency resolution. The middle panel of Fig. 6.12 is the AMS pattern generated from a segment of babble noise. In this case no particular structures across the modulation frequency axis are observed.

To further test the AMS implementation, a synthetic signal is generated. This signal is given by

$$s(n) = \sin\left(f_0 \frac{2\pi}{f_s} n\right) + \sin\left(f_1 \frac{2\pi}{f_s} n\right) \tag{6.2}$$

where $f_0 = 500\,\text{Hz}$, $f_1 = 600\,\text{Hz}$ and $f_s = 16\,\text{kHz}$. This means that the signal consist of two equal amplitudes sinusoids at 500 Hz and 600 Hz. The envelope of this signal

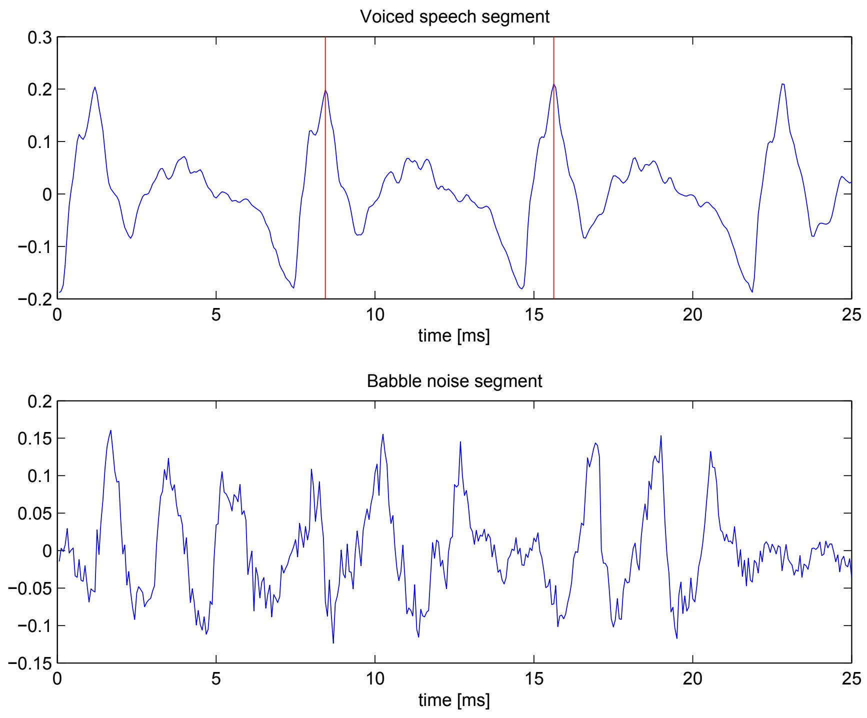


**Figure 6.11:** The top figure shows a voiced speech segment with a fundamental frequency of approximate 139 Hz. The bottom figure shows a segment of babble noise.

can be found by use of the trigonometric identity

$$\sin\alpha + \sin\beta = 2\sin\left(\frac{\alpha+\beta}{2}\right)\cos\left(\frac{\alpha-\beta}{2}\right). \tag{6.3}$$

Here $\alpha = 500\,\text{Hz}$ and $\beta = 600\,\text{Hz}$. Putting these values into Eq. (6.3) yields

$$\sin 500\,\text{Hz} + \sin 600\,\text{Hz} = 2\sin(550\,\text{Hz})\cos(50\,\text{Hz}) \tag{6.4}$$

where the cosine term at 50 Hz is the envelope. The AMS feature extraction find the envelope in each frequency band by taking the absolute value of the samples. Since the test signal consist of sinusoids with no offset, taking the absolute value of such signal should yield an envelope twice as big as the one calculated in Eq. (6.4). This will be referred to as the upper envelope. Therefore, it is expected to see activity at the modulation filter around 100 Hz. From the bottom panel of Fig. 6.12 it is seen that the signal as expected lightens up the modulation filters around 100 Hz. Likewise, quite intense activity is seen at the modulation filters around DC. This is due to the fact that the upper envelope have a mean value different from zero. Also, harmonics occur at 200 Hz, 300 Hz and 400 Hz, which are caused by taking the absolute value of the envelope found in Eq. (6.4). According to these tests it is assumed that the implementation of the AMS feature extraction is correct.

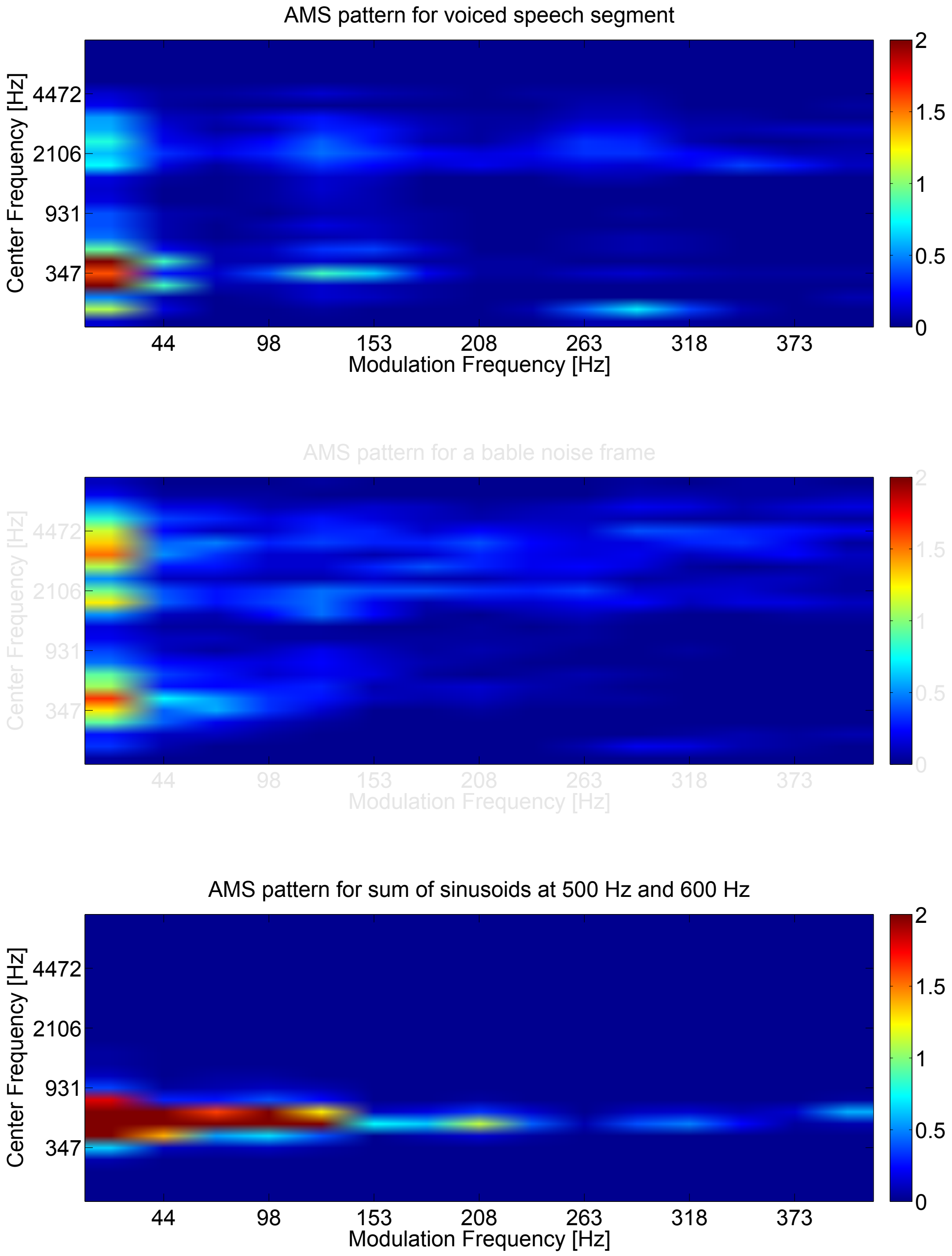


**Figure 6.12:** The two topmost figures shows the AMS pattern for a the voiced speech segment and the babble noise segment shown in Fig. 6.11. The bottom figure shows the AMS pattern of the test signal from Eq. (6.4).

### 6.3.3 DNN Training & Pre-training Evaluation

This section describes how the implementation of the DNN pre-training and fine-tuning is performing. The implementation of the DNN is based on the public available code provided by Geoffrey Hinton [Salakhutdinov and Hinton, 2006]. This code implements a three layer DBN consisting of Bernoulli-Bernoulli RBMs and a four layer DNN consisting of three layers of sigmoid hidden units and one softmax output layer. The DBN is trained using contrastive divergence with $k = 1$ and the DNN is trained using the non-linear conjugate gradient algorithm where the cross-entropy objective function is used and the method of backpropagation is used to compute the corresponding gradients.

Since the system described in [Healy et al., 2013] uses a two layer DBN and a three layer DNN as well as both GBRBMs and BBRBMs, the code provided by Geoffrey Hinton has been altered to fit the described system. The alternations consists of changing the BBRBM in the first layer of the DBN to a GBRBM as well as reducing the number of layers to two in the DBN and three in the DNN. Changing the type of RBM corresponds to sampling from a Gaussian distribution as the one given by Eq. (5.11) instead of sampling from a Bernoulli distribution as the one given by Eq. (5.6). However, instead of drawing a true sample from the Gaussian distribution it is reported in [Wang and Wang, 2013], that a sample $v_j$ is approximated by the mean of the Gaussian distribution, which is given by

$$v_j \approx b_j + \sum_{i=1}^{n} w_{ij} h_i. \tag{6.5}$$

where $b_j$ is a bias, $h_i$ is the state of the hidden unit and $w_{ij}$ is the corresponding weight. To investigate if the DBN and DNN training work as expected a couple of tests have been performed. The first test will investigate if the backpropagation training is working and the next couple of tests will investigate if the DBN training is working, i.e., the DNN pre-training.

#### Achieving Zero Training Error

One way to investigate if the backpropagation training is working is to train a DNN for so long time that the training error approaches zero. If the number of units in the DNN is sufficiently large, compared to the training set, such that the DNN can capture all the variations of the training data, the training error is likely to be zero after a given number of epochs [Montavon et al., 2012, p. 464].

A system used for testing, without pre-training and incorporating only one DNN, is illustrated in Fig. 6.1. The DNN chosen is the one covering the frequency channel around 1 kHz, which is assumed to be a frequency channel with large variations, when speech data is used. The training set comprises 10 randomly chosen HINT sentences mixed with babble noise at -5 dB SNR, which gives 2189 training samples. The LC, used when creating the IBM, is set to -8 dB. The test set comprises 10 randomly chosen HINT sentences, chosen from an isolated test set such that there is no overlap between training and test data. The test set consists of 2084 samples and is mixed with babble noise in the same way as the training set. The DNN consists of two hidden layers each having 50 hidden units and the DNN is trained using full-batch training for 500 epochs. The training error is defined as the percent of misclassified training cases, and can be calculated as

$$Training\ error = \left(\frac{\#Misclassified}{\#Training\ Samples}\right) \times 100 \tag{6.6}$$

and the test error is computed in a similar fashion. The training error as well as the test error as a function of epochs is illustrated in Fig. 6.13.

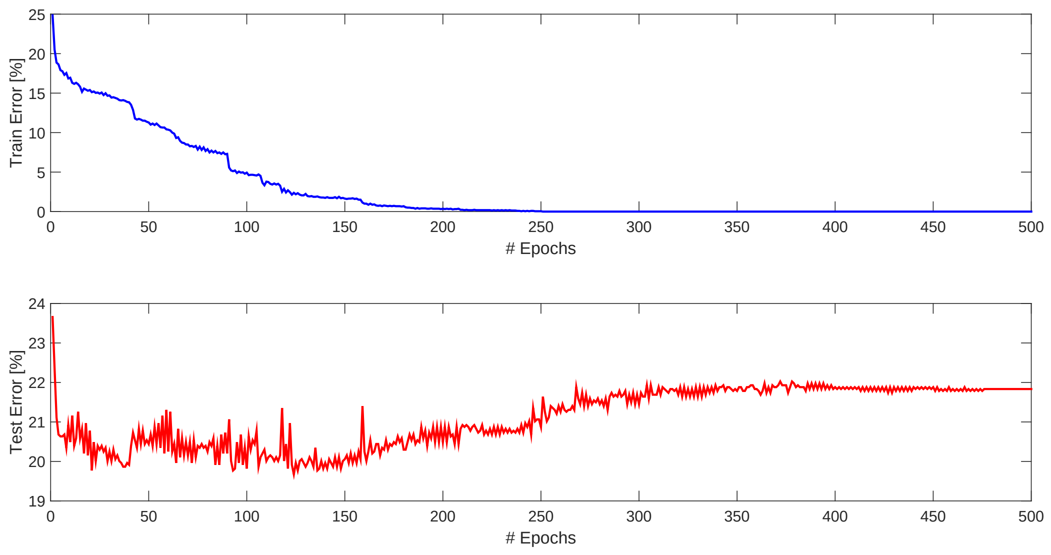


**Figure 6.13:** This figure shows how a DNN can be trained to produce a training error of 0 using the backpropagation algorithm. It is also seen that the test error gets worse after approximately 150 epochs which indicates that the DNN start to overfit the training data. These plots indicate that the backpropagation training is implemented correct.

From Fig. 6.13 it is seen that the training error is approaching zero as the number of epochs increases and after approximately 200 epochs the training error is zero and stays at zero for the remaining 300 epochs. From Fig. 6.13 it is also seen that the test error starts decreasing and then starts increasing again before it stagnates at an error rate of approximately 22%, which is assumed to be caused by the model being overfitted, which is the phenomena that occurs when a model is trained to much.

From the results of the test, which shows that a DNN can be trained to achieve a training error of zero it is assumed that the backpropagation training is implemented correctly and is working as intended.

### Pre-training Evaluation Part 1

Evaluating the implementation of the DBN training is not a trivial task. Investigating whether or not a DBN and more specific a RBM captures the right statistics of the training data can be difficult, especially due to the intractable computation of the log-likelihood function as given by Eq. (5.20). Ideally, if the log-likelihood function could be evaluated after each epoch, it could indicate if the DBN learns the statistics of the data and potentially also if the DBN starts to overfit the data if the log-likelihood function was evaluated on validation data [Hinton, 2012].

A metric which is natural to use when monitoring the learning process is the so called reconstruction error. The reconstruction error of a given training sample is the difference between the training sample and the sample which has been generated by the RBM after $k$ Gibbs steps when using k steps contrastive divergence. If the RBM has captured the statistics well, the sample from the RBM will ,with high probability, be similar to the training sample and the difference between the two will be small. However, the sample from the RBM is highly dependent on the mixing rate of the Gibbs chain. This means that if the mixing rate is low and CD-1 is used the sample from the RBM and the training sample are very similar. This will give a small reconstruction error, even though the equilibrium distribution of the Gibbs chain is very different from the distribution of the training data. Nevertheless, the reconstruction error is used but as Geoffrey Hinton says "Use it but don't trust it" [Hinton, 2012].

Some literature also propose to look at the histogram of the weights of the RBM and look at the non-gaussianity. Since the RBM is often initialized with Gaussian random variables the histograms of the weights should appear Gaussian in the beginning of training and then gradually become less Gaussian and more sparse as training progress [Dieleman and Schrauwen, 2012].

The most straightforward way to investigate if DBN training is working is to train a DNN and compare the classification performance with a DNN, which has been initialized with parameters from a trained DBN [Dieleman and Schrauwen, 2012]. Several of such tests have been performed in order to investigate how well the pre-training is working and under what conditions it is working. The tests are divided into two parts. The first parts investigates the influence of the size of the training set as well as the number of hidden units as function of epochs of backpropagation training. The second part investigates how the learning rate of the GBRBM and BBRBM as well as the number of pre-training epochs, influence the classification performance as function of epochs of backpropagation. The systems used for the two parts are similar to the system used in the former section for testing the backpropagation learning. The first part consists of 18 Pre-training Evaluation Tests (PET) as shown by Tab. 6.1. The 18 PETs consists of DNNs trained with and without pre-training for three different sizes of training set and three different number of hidden units. The training set is based on the HINT sentences and the three training sets are based on 10, 50 and 100 sentences, i.e., approximately 2200, 11000 and 22000 training samples per frequency channel. All sentences are mixed

| Test ID | # Training Sentences | # Hidden Units | Pre-training |
|---|---|---|---|
| PET1 | 10 | 50 | No |
| PET2 | 10 | 200 | No |
| PET3 | 10 | 500 | No |
| PET4 | 10 | 50 | Yes |
| PET5 | 10 | 200 | Yes |
| PET6 | 10 | 500 | Yes |
| PET7 | 50 | 50 | No |
| PET8 | 50 | 200 | No |
| PET9 | 50 | 500 | No |
| PET10 | 50 | 50 | Yes |
| PET11 | 50 | 200 | Yes |
| PET12 | 50 | 500 | Yes |
| PET13 | 100 | 50 | No |
| PET14 | 100 | 200 | No |
| PET15 | 100 | 500 | No |
| PET16 | 100 | 50 | Yes |
| PET17 | 100 | 200 | Yes |
| PET18 | 100 | 500 | Yes |

**Table 6.1:** This table presents 18 pre-training evaluation tests and the purpose of these tests are to investigate if the implementation of the DNN pre-training is working and under what circumstances pre-training has a positive effect on the DNN training. The results can be seen in Fig. 6.14

with babble noise at -5 dB SNR and the LC for the creation of the IBM is set to -8 dB. The test set for all tests in the first part consists of 50 HINT sentences, chosen from an isolated test such that there is no overlap between training and test data. The test set is mixed with babble noise in the same way as the training set.

The T-F representation of the training and test data is based on a 24 channel gammatone filter bank and to reduce the computational burden of the test, only six out of the 24 frequency channels are used for training and classification. The overall classification performance is then given as the average across the six DNNs which are trained on the six frequency channels. The channels selected for the test are channel 1, 5, 10, 15, 20 and 24 in an attempt to cover the whole 8 kHz spectrum.

The six DNNs are trained with and without pre-training and with a size of either 50, 200 or 500 hidden units in each of the two layers and each DNN is trained for 250 epochs of backpropagation. Mini-batch training is used with mini-batches of 2048 samples. The results from the tests, in the first part of the pre-training evaluation, are shown by Fig. 6.14

For clarity, the results from the tests of the first part is divided into three plots where each plot presents results from six tests with DNNs, with and without pre-training, having 10, 200 and 500 hidden units in each hidden layer. The topmost plot of Fig. 6.14 shows how the classification accuracy in number of misclassified test

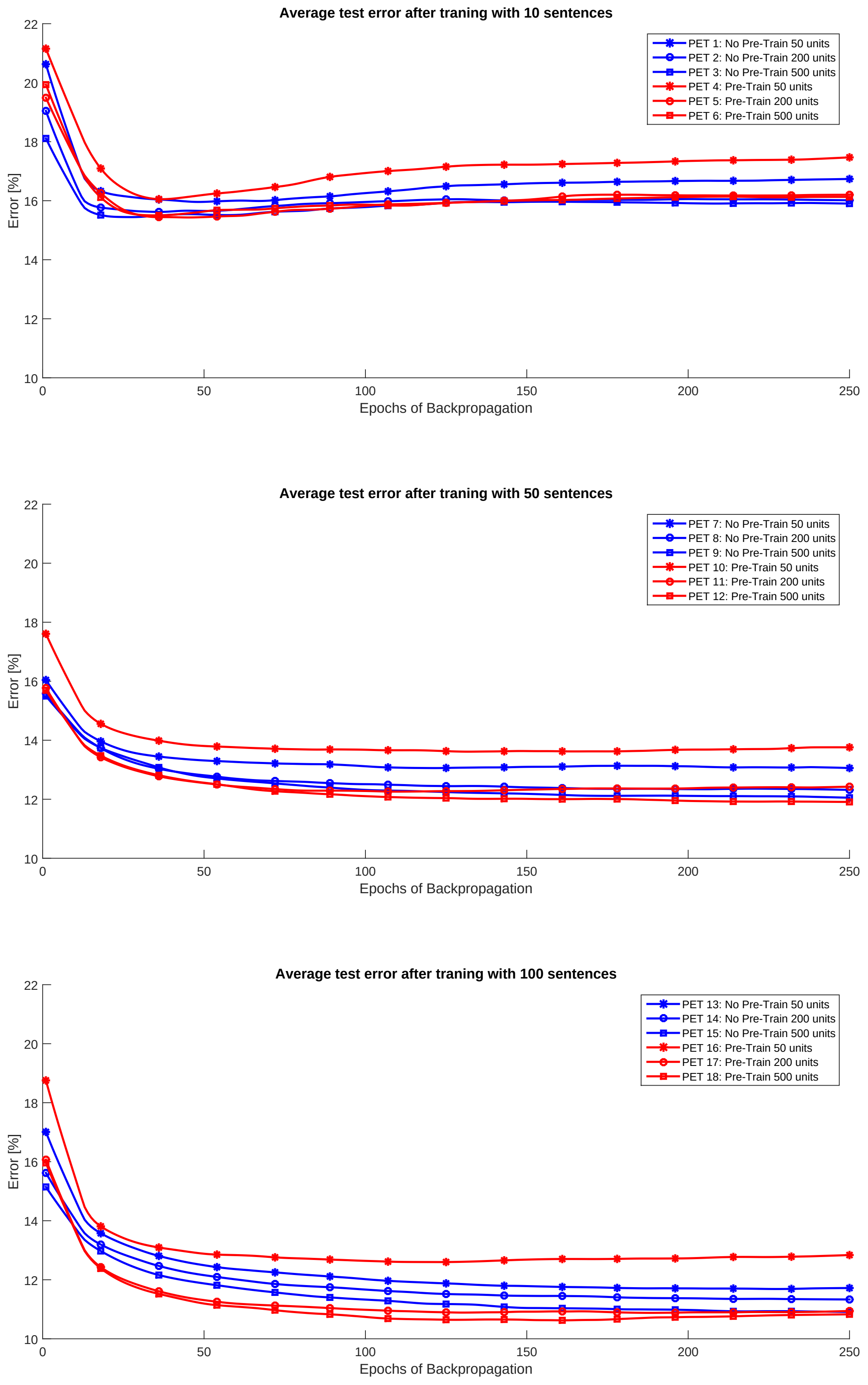


**Figure 6.14:** This figure presents 18 pre-training evaluation tests and the purpose of these tests are to investigate if the implementation of the DNN pre-training is working and under what circumstances pre-training has a positive effect on the DNN training. It is seen that pre-trained DNNs only perform better than non pre-trained DNNs for large training sets and large models.

samples varies as a function of epochs of backpropagation training using a training set comprising 10 HINT sentences. The plot in the middle presents results where the training set has increased to 50 sentences and the plot in the bottom of Fig. 6.14 show the result with a training set comprising 100 HINT sentences.

The results from Fig. 6.14 indicate that pre-training does have a positive effect on the classification accuracy but only with large models, i.e., a large number of hidden units, and with a large amount of training data. As it is seen from the topmost plot of Fig. 6.14 a DNN trained with backpropagation from random initialized weights is better than using the weights from the DBN for all three sizes of DNN.

From the plots in the middle of Fig. 6.14, where the training set is comprised of 50 HINT sentences it is seen that pre-training outperforms random initialization, but only with the DNN having 500 hidden units. Likewise, the bottom plot of Fig. 6.14 shows that the largest DNN with pre-training performs the best but also the second largest DNN with pre-training, having 200 hidden units performs better than the counterpart with random initialization for most epochs.

The overall result from the tests presented in Fig. 6.14 indicate that the implementation of the pre-training is working, however pre-training is only beneficial for large datasets and large models.

Two possible arguments explaining the seen behaviour could be the following. The pre-training can be seen as a method to prevent backpropagation from being stuck in a bad minimum and hereby should pre-training initialize backpropagation training such that it starts from a more favorable location in parameter space. However if the DNN only have a small amount of hidden units the parameter space is not as complicated as if there were a large number of hidden units. With a small parameter space backpropagation might have an easier job finding a good minima, compared to the large models where the parameter space is large and where pre-training seems to benefit.

An argument explaining the requirement of a large number of training samples could be based on the gradient approximations done using contrastive divergence. Two rough approximations were made in the derivation of contrastive divergence. First, an expected value was approximated by a sample and second, that sample was approximated by a single Gibbs step, so maybe these two approximations require a fair amount of training data before the approximations become good enough to train a good model.

It should be mentioned that the reconstruction error during pre-training for all tests was observed to be decreasing as expected, which just supports a proper implementation of the DNN pre-training functionality.

### Pre-training Evaluation Part 2

The second part of the pre-training evaluation is related to 5 PETs where the learning rate of the GBRBM and the BBRBM as well as the number of pre-training epochs are monitored. The configuration of the 5 PETs are listed in Tab. 6.1. In [Hinton, 2012] Geoffrey Hinton proposes that the learning rate for a GBRBM should be an

| **Test ID** | **Pre-training Epochs** | **BBRBM Learning rate** | **GBRBM Learning rate** |
|---|---|---|---|
| PET19 | No Pre-training | - | - |
| PET20 | 100 | $\mu = 0.01$ | $\mu = 0.001$ |
| PET21 | 50 | $\mu = 0.01$ | $\mu = 0.001$ |
| PET22 | 100 | $\mu = 0.1$ | $\mu = 0.01$ |
| PET23 | 50 | $\mu = 0.1$ | $\mu = 0.01$ |

**Table 6.2:** This table presents 5 pre-training evaluation tests and the purpose of these tests is to investigate if the implementation of the DNN pre-training is working and under what circumstances pre-training has a positive effect on the DNN training. The results can be seen in Fig. 6.15

order of magnitude smaller than the learning rate for a BBRBM. In [Healy et al., 2013] learning rates of 0.01 and 0.1 were used for the GBRBM and the BBRBM respectively and in [Wang and Wang, 2013] learning rates of 0.001 and 0.01 were used for the GBRBM and the BBRBM respectively. To investigate how the learning rate influences the classification performance both of these learning rates are tested in the second part of the tests. Also the number of pre-training epochs, i.e., epochs of contrastive divergence is investigated. Based on the results of the tests in the first part, a large training set is used, which consists of 100 HINT sentences each mixed with babble noise six times which increases the training set to 600 sentences. This approach is similar to what is presumably done in [Healy et al., 2013]. Using 600 sentences for training each frequency channel is represented by approximately 132000 samples. The test set is based on 160 HINT sentences, mixed with babble noise, which is equivalent of approximately 35000 test samples per frequency channel. Like the former tests the noisy sentences are mixed at -5 dB SNR with a local criterion of -8 dB. Also like the former tests, only a subset of the frequency channels were used for training and the overall result is constructed by an average over these frequency channels. In the current tests, frequency channel number 5, 10, 15 and 20 was used out of a total of 24 and each DNN had 200 hidden units in each layer and was trained for 50 epochs of backpropagation. The result for the second part of the pre-training evaluation is presented in Fig. 6.15.

Figure 6.15 shows the results from five PETs. One test without pre-training (PET 19 blue line), which acts as a reference, and four tests with pre-training, each with its own setting of number of pre-training epochs and learning rates (PET 20-23). It is seen that within the duration of the 50 epochs all four DNNs with pre-training outperforms the DNN without pre-training, which support the results from the tests from the first part.

It is also seen that the two DNNs with the larger learning rates, i.e, 0.1 for the BBRBM and 0.01 for the GBRBM (PET 22 and PET 23), yields the best and very similar performance. The difference between the two is the number of pre-training epochs which are either 50 or 100. Third best is the DNN with learning rates of 0.01 for the BBRBM and 0.001 for the GBRBM (PET 20), which has been trained for

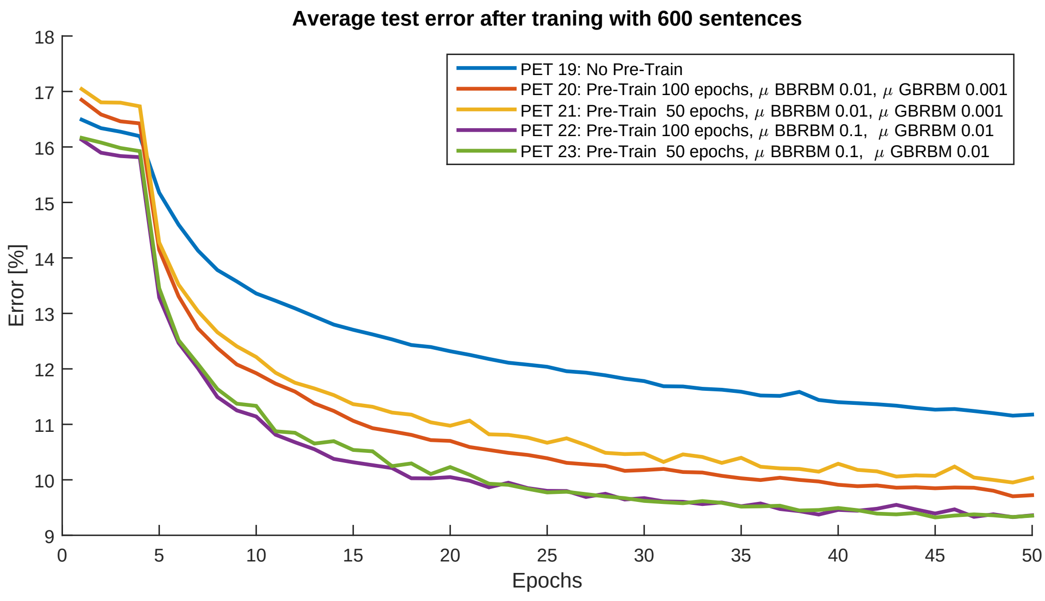


**Figure 6.15:** This figure presents 5 pre-training evaluation tests and the purpose of these tests is to investigate if the implementation of the DNN pre-training is working and under what circumstances pre-training has a positive effect on the DNN training. It is seen that learning rates of 0.1 and 0.01 for the BBRBM and GBRBM respectively achieves the best results and no noticeable performance improvement is achieved by training for 100 epochs instead of 50.

100 epochs and fourth best is the same DNN with only 50 epochs of pre-training (PET 21).

From these tests it is clear that on the current training set, learning rates of 0.1 for the BBRBM and 0.01 for the GBRBM yields the best performance and nothing significant is gained by pre-training the DNNs for more than 50 epochs.

The overall conclusion for the pre-training evaluation is that it is assumed that the implementation of the algorithms are correct and from tests it is justified that pre-training under some circumstances yields improved performance.

## 6.4 Early Stopping

During initial training sessions, it was identified that the classification accuracy, on the test set, for different subband DNNs peaked at different epochs. This means that if all DNNs are trained for the same amount of time the capability of the system will not be completely utilized hence, some DNNs are overfitted and some have not reached their maximum classification performance yet. Another thing identified, was that the DNNs trained with pre-training started overfitting relatively quickly compared to DNNs trained with backpropagation alone. This means that in order to utilize the power of pre-training, as well as potentially reducing the training time, the length of the backpropagation training must be carefully chosen for each subband

DNN. This is done using an early stopping technique as presented in the following. A plot showing the observed behaviour is depicted in Fig. 6.16.

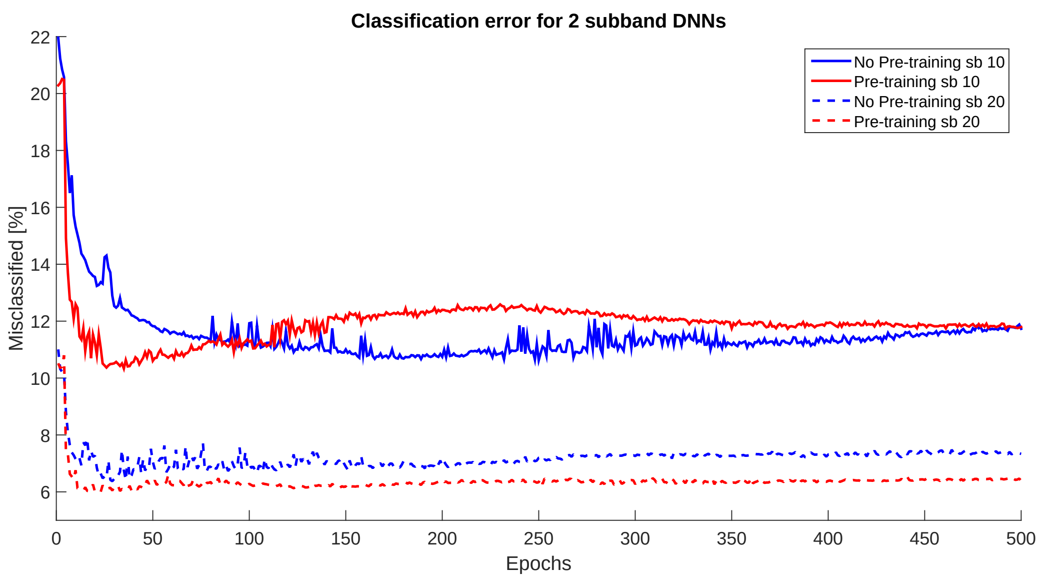
The top figure in Fig. 6.16 shows the classification test error for subband DNN 10 and 20 and the bottom figure of Fig. 6.16 shows the average classification test error for 24 subband DNNs. The DNNs are trained similar to the tests performed in Section 6.3.3 excepts that the size of the test set is reduced to 80 instead of 160 sentences and the batch size is set to 512 instead of 2048. The blue lines show the test error when only backpropagation is used and the red lines show the test error when 100 epochs of pre-training is used. From the top figure it is seen that the DNNs achieving the smallest classification error are the DNNs with pre-training, but the minimum is achieved at different epochs. The subband DNN for subband 10 achieves the minimum error around epoch 35 whereas the subband DNN for subband 20 achieves the minimum error around epoch 20. It is also seen that if subband 10 is trained for 500 epochs pre-training have no effect but if only 35 epochs of backpropagation training is used the pre-trained DNN will perform the best.

In the bottom figure the average classification error for all 24 subband DNNs, pre-trained and non pre-trained are computed. From this figure it is seen that the pre-trained DNNs on average achieves the lowest classification test error after approximately 35 epochs. It is also seen that if all DNNs are trained for 150 epochs the DNNs trained with backpropagation alone achieves the same classification error as the pre-trained DNNs. From Fig. 6.16 it can be concluded that it is vital to detect the minimum classification test error in order to fully utilize the power of pre-training. Identifying such minima and stopping the training is known as early stopping.

Early stopping refers to the mechanism that stops the training before the maximum number of epochs have elapsed and is a mechanism, which aims at reducing the amount of overfitting, hence improving the DNNs ability to generalize [Montavon et al., 2012, p. 53].

To implement early stopping functionality a validation set is used to evaluate the progress of training. The purpose of the validation set is to monitor the classification accuracy on a data set which is not part of the training set. The validation set is constructed such that the error on the validation set is assumed to be similar to the error achieved if the test set was used. To fulfill this assumption the validation set and the test set must be of similar size and variety of data. During training the error on the validation set is repeatedly evaluated and when the performance has peaked the training is stopped and the system can be evaluated on the test set.

Since the validation error is not a monotonically decreasing function of epochs, early stopping is implemented in terms of a *lookahead* variable. This implies that if the validation error has not decreased for a certain number of epochs, defined by the *lookahead* variable, the training is stopped. The size of the *lookahead* variable enables the early stopping functionality to be resilient against local minima within the number of *lookahead* epochs. During each epoch the validation error is evaluated and the DNN weights and biases are stored if the validation error is lower than it was at the preceding epoch. In this way the parameters corresponding to the minimum



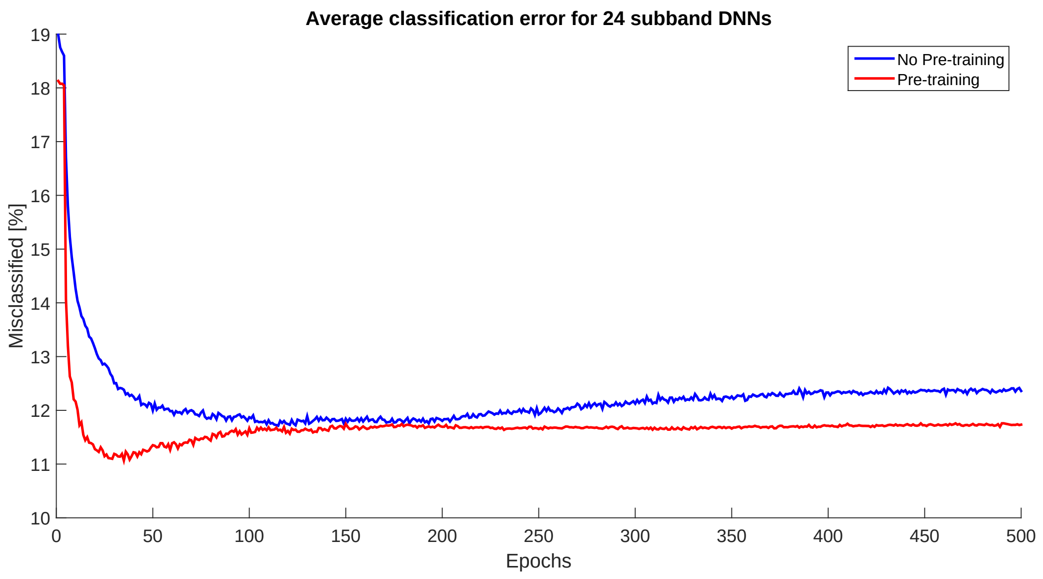


**Figure 6.16:** These plots show how the test error is distributed as function of epochs. In the top figure two DNNs have been trained both with and without pre-training and it is clear that the distribution of the test error between the two DNNs are very different. However, the pre-trained DNNs achieve the lowest classification error. On the bottom figure an average is made across 24 DNNs trained with both pre-training and without pre-training and it is seen that the pre-trained DNNs achieves the lowest average classification error after 35 epochs. Achieving such a performance in practice require the training to be stopped prematurely, and therefore early stopping has been implemented.

validation error is always the one being stored.

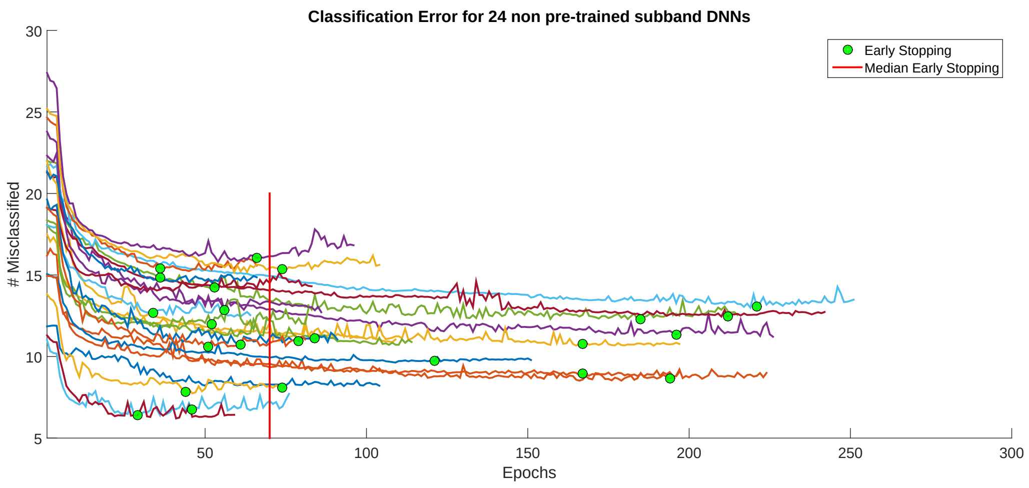
In Fig. 6.17 the same simulation as the one presented in Fig. 6.16 has been run, but this time, with early stopping implemented. The *lookahead* variable was set to 30 epochs, which means that the training was terminated for a particular DNN if the validation error has not decreased for 30 epochs. Figure 6.17 shows the classification test error for all subband DNNs without pre-training in the top figure, all subband DNNs with pre-training in the figure in the middle and an average across subband DNNs in the bottom figure. The blue line in the bottom figure is the average classification test error for 24 DNNs trained with backpropagation alone and the red line is the average classification test error for 24 pre-trained DNNs. The green circles on the different plots indicate the epoch for which the validation error was the smallest. This also mean that training of a given DNN is stopped 30 epochs after a green circle. Since the different subband DNNs are trained for different number of epochs the blue and red lines in Fig. 6.17 become more and more "noisy". During the first 40 epochs the average is made across approximately 24 subband DNNs but after 40 epochs the majority of the DNNs have been terminated due to early stopping. This implies that the average is made using a small number of subband DNNs, hence producing the noisy plots.

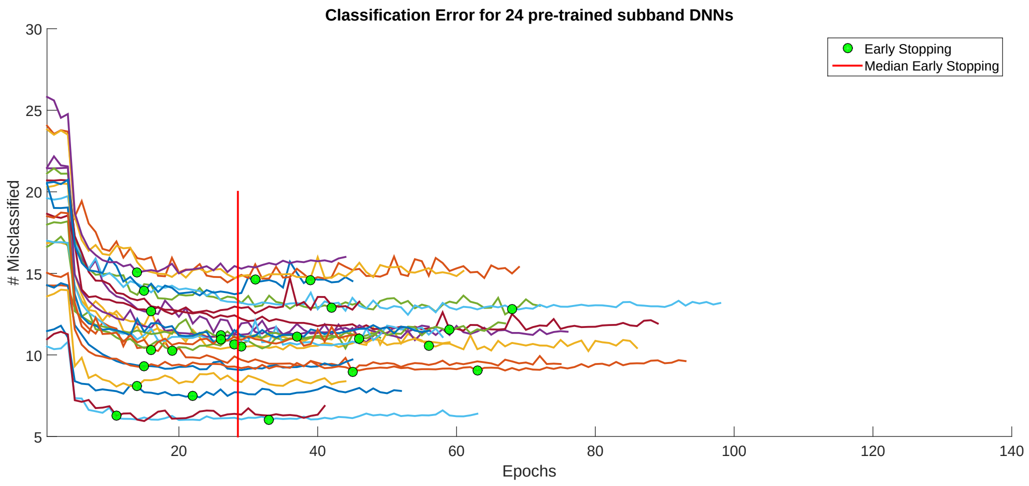
From Fig. 6.17 it is seen that the DNNs trained with pre-training have a lower average classification test error as well as the shortest training time since the majority of the DNNs are terminated before 50 epochs have elapsed. Some of the subband DNNs trained purely with backpropagation have not even reached its minimum after 100 epochs. The average classification performance as well as the training time for the four different simulation settings used in Fig. 6.16 and Fig. 6.17 are presented in Tab. 6.3

| **Settings** | | **Results** | |
|---|---|---|---|
| Pre-training | Early Stopping (*lookahead*) | Classification Accuracy | Training Time |
| No | No | 87.64 % | 31.8 Hours |
| Yes | No | 88.27 % | 34.7 Hours |
| No | Yes (30 Epochs) | 88.62 % | 13.7 Hours |
| Yes | Yes (30 Epochs) | 89.17 % | 6.7 Hours |
| Yes | Yes (50 Epochs) | 89.18 % | 9.4 Hours |

**Table 6.3:** Results from five different simulations where early stopping was investigated. It is seen that early stopping improves the classification performance of the speech enhancement system for both pre-trained and non pre-trained DNNs. It is also seen that the training time is reduced considerably using early stopping but it is a trade off between training time and classification performance

From Tab. 6.3 it is seen that early stopping improves both classification accuracy and training time regardless of the use of pre-training. However, the largest overall performance gain is achieved when both pre-training and early stopping is applied where a classification accuracy increase of 0.9 % is achieved together with a decrease of training time of 28 hours or approximately 80 %. An additional simulation was made where the *lookahead* variable was changed from 30 to 50 in order to investi-





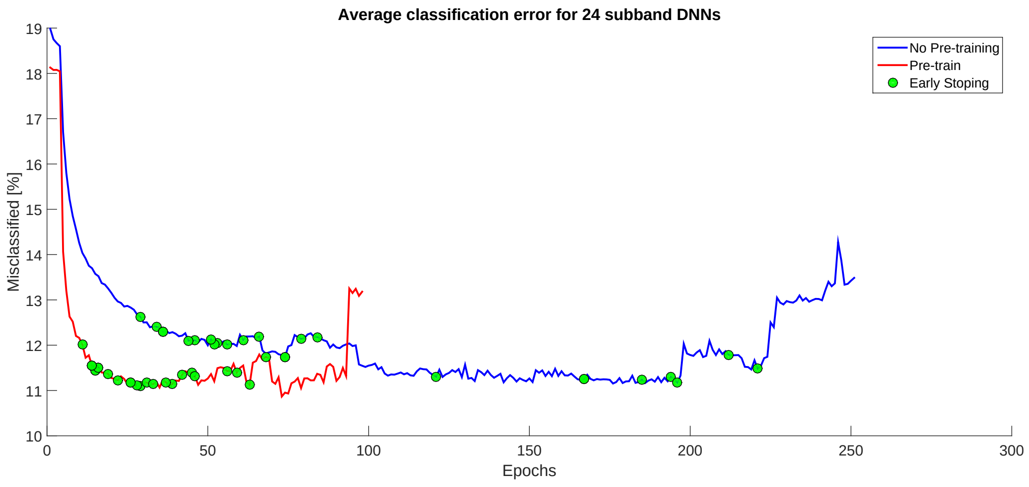


**Figure 6.17:** Three plots showing the effect of early stopping on non pre-trained and pre-trained DNNs. The top figure shows 24 non pre-trained DNNs and the figure in the middle shows the same DNNs but with pre-training applied. It is seen that the median of the epochs at which the DNN training is terminated is several epochs lower for the pre-trained DNNs compared to the non pre-trained DNNs. This can also be seen from the bottom figure which shows the average classification performance for both pre-trained and non pre-trained DNNs. From the figures it is clear that both the classification error and training time is reduced using early stopping.

gate if a performance gain could be achieved. It can be seen that the classification accuracy is increased by 0.01 % and the training time is increased to 9.4 hours or approximately 40 %. From this it can be concluded that setting the size of the *lookahead* variable is a trade off between classification accuracy and training time. Setting the *lookahead* variable to a high number can potentially lead to a better performance, but it definitely leads to a larger training time.

From Figs. 6.16, 6.17 and Tab. 6.3 it is clear that early stopping both improves the classification accuracy as well as reducing the training time. It is also seen that by applying early stopping the power of pre-training can be effectively utilized.

In [Healy et al., 2013] nothing was mentioned about how the DNNs were trained and from the above results and argumentation it is hard to believe that the DNNs were trained equally, i.e., all subband DNNs were trained for the same number of epochs. However, since no information about the training is available it has been decided to perform early stopping on all simulations reported in the subsequent chapter.

## 6.5 Algorithmic Analysis

This section describes some of the aspects of the speech enhancement algorithm, which are relevant if the system is to be implemented on dedicated hardware. Some of the major blocks that need to be implemented if the speech enhancement algorithm shall function in a real-life scenario are illustrated by the speech enhancement chain in Fig. 6.18.

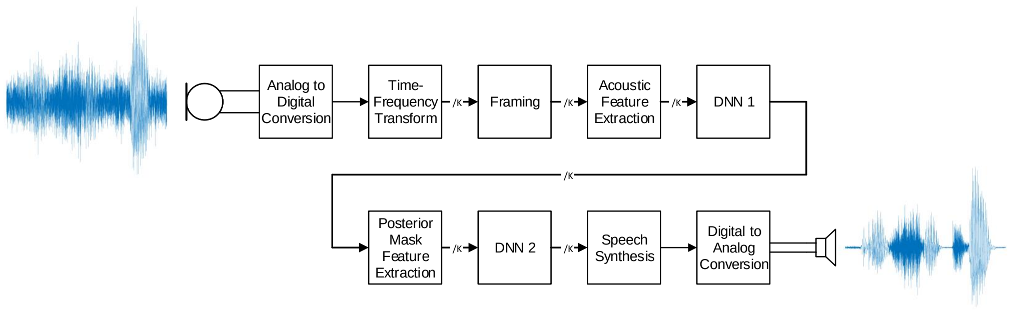

Analog to Digital Conversion
Time-Frequency Transform
/K
Framing
/K
Acoustic Feature Extraction
/K
DNN 1
/K
Posterior Mask Feature Extraction
/K
DNN 2
/K
Speech Synthesis
Digital to Analog Conversion

**Figure 6.18:** This figure presents a speech enhancement chain which is a block diagram covering the major blocks needed if the speech enhancement system presented in Fig. 6.1 is to be implemented on dedicated hardware.

From Fig. 6.18 it is seen that the first thing carried out is an analog to digital conversion of the signal. Then a T-F transform is performed followed by a framing procedure. When enough samples are recorded to form a frame, i.e., 20 ms, it can be transformed into features and the frame can be classified as being either speech or noise dominated by a DNN as illustrated by the `DNN 1` block. When the classification is performed by the first $K$ subband DNNs, new features are computed based on probabilities from previous classified frames, hence forming a posterior mask feature vector. The posterior mask feature vector is then used to compute a more accurate speech domination probability using a second set of $K$ subband DNNs, illustrated by the `DNN 2` block. This probability is used to synthesize the current frame before the synthesized and enhanced speech signal is converted back into the analog domain by a digital to analog converter.

All the operations just described introduces latency and use memory. The following sections will describe some of the aspects to be aware of regarding algorithmic delay, computational and memory complexity as well as inherent parallelism. Information that is needed if a hardware architecture is to be designed for running the speech enhancement algorithm in real-time.

### 6.5.1 Algorithmic Delay

This section describes the algorithmic delay which is meant as the time delay that is intrinsic to the algorithm independent on the processing time. This means that algorithmic delay is hardware independent and is the minimum delay that can be achieved without changing the algorithm. In the speech enhancement chain three stages contribute to the algorithmic delay: The T-F transform, the framing and the posterior mask feature extraction. The T-F transform is based on a gammatone filter bank, which consists of bandpass FIR filters having non-linear phase, hence a non-constant group delay [Wang and Brown, 2006, p. 16]. Due to the group delay the input signal will be delayed and due to the non-linear group delay, phase distortion will occur. By analyzing the impulse responses of the different filters the largest group delay was found to be approximately 600 samples, which is equivalent to 3.75 ms at a 16 kHz sampling frequency. This means that the T-F transform itself introduces up to 3.75 ms of delay. A phase-compensated gammatone filter bank exists, which compensate for the difference in group delay between the different filters [Wang and Brown, 2006, p. 17]. This filter bank does not remove the group delay but it reduces the phase distortion across frequencies, something which has been shown to be audible already at a few milliseconds [Blauert and Laws, 1978].

The framing of the T-F transformed input signal is another source of intrinsic delay. Since a frame consists of 320 samples, 320 samples are needed before it can be processed by the acoustic feature extraction block as shown by Fig. 6.18. Due to the overlap of 10 ms, the effective delay will be 10 ms between consecutive frames. This means that no matter how fast the hardware can sample and process the signal a 10 ms delay is introduced due to the framing process. The only way to reduce this delay is to reduce the size of the frames and/or the overlap.

The last source of intrinsic delay is the posterior mask feature extraction. The posterior mask features are based on a window of probabilities using the probabilities of frames two frames into the future. This means that in order to construct the posterior mask feature vector a delay of two frames, i.e., 30 ms is introduced. Combined with the delay of the 10 ms of the framing and the few ms of group delay the total intrinsic delay of the speech enhancement chain is between 40 and 43.75 ms. Studies has shown that delays longer than 10 to 15 ms. in hearing aids are perceived disturbing so an intrinsic delay of approximately 40 ms. in the presented speech enhancement algorithm might be too long to be applicable in a hearing aid [Kates and Arehart, 2005].

### 6.5.2 Computational Complexity

This section describes some aspects of the computational complexity of the speech enhancement chain. The computational complexity is highly dependent on the implementation of the individual sub algorithms which are present in the speech enhancement chain. Operations such as convolution, DFTs and DCT can, if implemented using the FFT and the Fast Cosine Transform (FCT) algorithms, be implemented with a $\mathcal{O}(n \log n)$ complexity opposed to the original $\mathcal{O}(n^2)$. As mentioned, the T-F transform in Fig. 6.18 consists of a gammatone filter bank, which is based on bandpass FIR filters. Such filters can be implemented using multiplication in the frequency domain instead of convolution in the time domain and potentially decrease the computational complexity of the filter bank.

The DFT and DCT are both used in the acoustic feature extraction step and they should both be implemented using the fast methods in order to decrease the computational complexity.

The computational complexity of the DNN is linear, i.e., $\mathcal{O}(n)$ in the number of hidden units and hidden layers and with respect to the dimension of the input vector. The computational complexity of the synthesis stage is similarly linear. From the above argumentation the computational complexity of the speech enhancement chain is $\mathcal{O}(n \log n)$ regarding the number of samples per frame and $\mathcal{O}(n)$ regarding the number of frequency bands.

Another way to quantify the computational complexity of the speech enhancement chain would be using the measure of Floating-point Operations Per Second (FLOPS). The FLOPS measure represents the number of mathematical operations, using a floating point number representation, per second required by a given algorithm. The actual number of FLOPS of an algorithm is directly related to the implementation and is, among other things, influenced by the order of operations such as performing a sum operation before multiplication instead of multiplying before summing. Since the FLOPS measure is so dependent on the implementation and since the implementation of the speech enhancement algorithm shown in Fig. 6.1 is designed for off-line computing a FLOPS count will not be computed since a realistic FLOPS count will require a redesign of the implementation for real-time use, which is beyond the scope of this project.

To give an indication on how computational heavy the speech enhancement algorithm is, a 2.5 s noisy speech sequence has been processed by the speech enhancement algorithm. The current MATLAB implementation of the algorithm uses 64 frequency channels, i.e., 64 subband DNNs and the implementation is purely sequential. The noisy speech sequence consists of 40032 samples and it takes approximately 6.5 seconds to process the speech sequence on a standard laptop featuring a 2.7 GHz Intel core i7 4800MQ Central Processing Unit (CPU). The feature extraction itself took 4.85 s or equivalently 76 ms per subband DNN, which is approximately 75% of the total execution time. These numbers might reflect the $\mathcal{O}(n \log n)$ complexity of the T-F transformation and the feature extraction stage opposed to the $\mathcal{O}(n)$ complexity of the DNN classification stages. These execution times are also comparable to the once reported in [Healy et al., 2013] where a execution time of 123 ms per subband DNN is reported whereas 107 ms is spent on the feature extraction. These results were based on a 3 s speech sequence using a 2.8 GHz Xeon CPU.

### 6.5.3 Inherent Parallelism

Even though the computational complexity of the speech enhancement chain, as illustrated in Fig. 6.18, is not precisely described the execution time can be reduced if multi core, Single Instruction Multiple Data (SIMD) or Multiple Instruction Multiple Data (MIMD) architectures are utilized opposed to single core and Single Instruction Single Data (SISD) architectures. The speech enhancement chain possess some inherent parallelism and the most straight forward way to utilize this is by running all $K$ subband DNNs in parallel on different processor cores. Running the subband DNNs in parallel a speedup of almost a factor of $K$ can be achieved.

Within each subband DNN, data are propagated forward following the structure of Eq. (4.4). Equation (4.4) can be implemented in parallel using a SIMD architecture since all inputs are multiplied independently with a weight for each hidden unit. This potentially gives a speedup of a factor of the number of input units times the size of the first hidden layer. When the output of the first hidden layer is computed, the second hidden layer and subsequent layers can be computed in a similar and parallel fashion leading to even larger speedups. This can of course be achieved for both `DNN 1` and `DNN 2` in Fig. 6.18.

In the current implementation, `DNN 1` has 85 input units with 200 hidden units. This means that each input is multiplied with a weight for each of the 200 units and all these computations can be done in parallel leading to a speedup in the first layer of $85 \times 200 = 17,000$ times compared to a purely sequential implementation. Achieving this speedup in an actual implementation is not possible since a severe memory overhead must be expected when this many parallel processing units use the same data. However, the speedups described in this section indicate how important the hardware architecture is regarding an efficient real-time implementation of the speech enhancement algorithm.

Especially in small embedded systems, such as hearing aids, where low power consumption and real time performance are critical, new strategies might be used and it might be beneficial to leave the traditional Von Neumann architecture and look into the emerging field of brain inspired silicon chips, where neural networks are hardwired in silicon [IBM Research, 2014b] [Cognimem, 2014]. These chips have the benefit of being highly applicable to DNN applications utilizing a high degree of parallelism while having a power consumption orders of magnitude lower than traditional Von Neumann inspired chips [IBM Research, 2014a].

### 6.5.4 Memory Complexity

The memory complexity of the speech enhancement chain is also highly dependent on a given implementation since the amount of memory which can be shared among the processing units depends on how the scheduling is designed, which again is highly dependent on the hardware architecture. The memory analysis presented in this section is based on the current implementation and is related to the number representation, the sampling frequency, the number of frequency bands, the length of the gammatone filters and the size of the DNNs. The analysis is not complete and the memory estimates presented cover only the parts of the algorithm which are considered to be the most memory demanding. Furthermore, it is assumed that the different subband DNNs are implemented in parallel, which limit the amount of memory that can be shared.

The number representation used for all simulations presented in this thesis is the standard "double" in MATLAB, which is based on the IEEE 754 B64 floating point representation. A single variable represented using this number representation takes up 8 bytes of memory.

The individual gammatone filters used in the current implementation are 2048 samples long and if 64 frequency channels are used for the T-F transform, approximately one MB ($64 \times 2048 \times 8 = 1,048,576$) is needed solely to store the impulse responses required to perform the T-F transformation. To this, memory to store the samples to be transformed should be added. Depending on the level of parallelism utilized this could require up to an additional megabyte of memory.

The framing operation works on 20 ms frames which is 320 samples when a sampling frequency of 16 kHz is used. For the 64 frequency channels this require $64 \times 320 = 163,840$ bytes in order to store the 64 frames produced by the T-F transformation.

The acoustic feature extraction block computes the MFCC, AMS and RASTA-PLP features based on a single frame. Since the computation of each of the three features require different arithmetic operations, temporary variables may be needed to store temporary results. This could potentially require an additional $64 \times 2 \times 3 \times 320 \times 8 = 983,040$ bytes if each of the three feature computations require data memory and temporary memory.

The DNNs are by far the most memory consuming part of the speech enhancement chain. In the configuration used in Chapter 7, 64 subband DNNs are used having three layers each. In the first classification stage (DNN 1 in Fig. 6.18) the input vector has dimension 85. In the second classification stage, (DNN 2 in Fig. 6.18) the dimension of the input vector varies from 45 to 85 due to the border conditions of the posterior mask as described in Section 6.1.4. Each of the two hidden layers have 200 units and the output layer has 2 units. In the first classification stage, approximately 30 MB ($64 \times (86 \times 200 + 201 \times 200 + 201 \times 2) \times 8 = 29,594,624$ bytes) is needed to store the weights and biases needed to perform a forward pass in the

DNNs. These 30 MB is excluding the memory needed to store the temporary results from each hidden unit. The memory required by the second classification stage is approximately 27 MB due to some of the input vectors having a dimension less than 85.

The posterior mask feature extraction generation requires memory to store the past and future mask probabilities, which adds up to approximately 35 kB.

The speech synthesis process only requires a copy of a frame from each frequency channel as well as the result from the subband DNNs which is approximately equal to 164 kB.

From the above very rough estimates on the memory consumption of the speech enhancement chain a total of approximately 60 MB is needed for processing a single frame of noisy speech. The majority of the memory is used for storing weights and biases for the subband DNNs. One way to reduce this is to consider a different number representation for the weights and biases. It could be so that the performance decrease due to lost numerical precision is negligible compared to the decrease in memory complexity by using for example a fixed point number representation [Gupta et al., 2015]. Another way to reduce the memory requirement is to share memory resources among computational units by performing more computations in sequence compared to parallel. Potentially, this increases the execution time so a trade off between execution time and memory complexity may be made.

# Chapter 7

# Test Results & Discussion

This chapter presents the evaluation tests of the speech enhancement system presented in Chapter 6 and shows the speech enhancement capability for different training and test conditions. The reasoning behind the different training and test conditions are based on the objectives described in Chapter 1.

Table 7.1 presents an overview of the tests, which has been conducted. In the following the system presented in Chapter 6 will be referred to as the Implementation Under Test (IUT) and a total of 16 IUTs have been tested based on the settings in Tab. 7.1. This means that the 16 IUTs differ in the way their subband DNNs have been trained. All results, i.e., figures, tables and wav files for the clean, noisy and processed signals can be found on the appendix DVDs and a detailed list of the parameters used for the system, can be found in appendix E.

As seen from Tab. 7.1 a total of 16 tests have been conducted. The tests have been designed such that their results can be used to conclude on the objectives defined in Chapter 1. The 16 tests can be divided into two groups. A group of 8 tests (S1-1/2 - S4-1/2) where looped noise is used and a group of 8 tests (S5-1/2 - S8-1/2) where non-looped noise is used. Each of these two groups can similar be divided into two. A group where generative pre-training have been utilized and a group where only discriminative training have been used. All test IDs ending with a -1 indicate that pre-training has not been utilized and all test IDs ending with a -2 indicate that pre-training has been utilized.

The noise types, SNRs and LC have been chosen based on the ones used in [Healy et al., 2013] which are the following: Babble noise at 0 dB SNR and -5 dB SNR with a LC of -6 dB and -10 dB, respectively. SSN at -2 dB SNR and -5 dB SNR with a LC of -6 dB and -10 dB, respectively. The speech corpus has also been chosen based on [Healy et al., 2013], and is the HINT corpus available from [Wang, 2013]. The babble noise and SSN sequences used for the tests with looped noise are also the ones used in [Healy et al., 2013] which are available from [Wang, 2013].

In Section 7.2, results are presented which are related to objective 1, 2 and 3. In Section 7.3 results are presented which are related to objective 4 and 5. In Section 7.4 suggestions on how to improve the performance of the speech enhancement system will be presented.

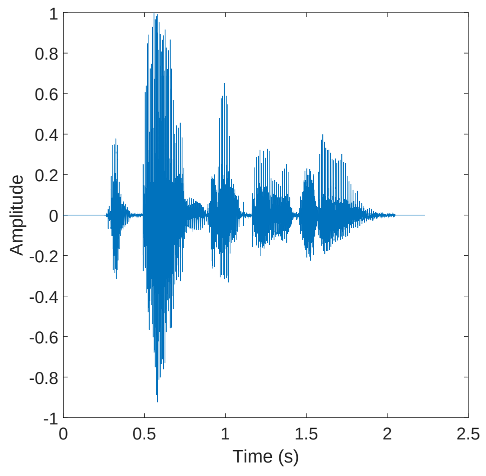
| Test ID | Pre-training | Corpus | Noise Type | SNR | LC | Training Set Size | Validation Set Size | Test Set Size | # Layers | # Units/ layer | Batch size |
|---|---|---|---|---|---|---|---|---|---|---|---|
| S1-1 | No | HINT | Babble (looped) | -5 dB | -10 dB | 100 Sentences x 6 | 80 Sentences | 80 Sentences | 2 | 200 | 512 |
| S1-2 | Yes | HINT | Babble (looped) | -5 dB | -10 dB | 100 Sentences x 6 | 80 Sentences | 80 Sentences | 2 | 200 | 512 |
| S2-1 | No | HINT | Babble (looped) | 0 dB | -6 dB | 100 Sentences x 6 | 80 Sentences | 80 Sentences | 2 | 200 | 512 |
| S2-2 | Yes | HINT | Babble (looped) | 0 dB | -6 dB | 100 Sentences x 6 | 80 Sentences | 80 Sentences | 2 | 200 | 512 |
| S3-1 | No | HINT | SSN (looped) | -5 dB | -10 dB | 100 Sentences x 6 | 80 Sentences | 80 Sentences | 2 | 200 | 512 |
| S3-2 | Yes | HINT | SSN (looped) | -5 dB | -10 dB | 100 Sentences x 6 | 80 Sentences | 80 Sentences | 2 | 200 | 512 |
| S4-1 | No | HINT | SSN (looped) | -2 dB | -6 dB | 100 Sentences x 6 | 80 Sentences | 80 Sentences | 2 | 200 | 512 |
| S4-2 | Yes | HINT | SSN (looped) | -2 dB | -6 dB | 100 Sentences x 6 | 80 Sentences | 80 Sentences | 2 | 200 | 512 |
| S5-1 | No | HINT | Babble (non-looped) | -5 dB | -10 dB | 100 Sentences x 6 | 80 Sentences | 80 Sentences | 2 | 200 | 512 |
| S5-2 | Yes | HINT | Babble (non-looped) | -5 dB | -10 dB | 100 Sentences x 6 | 80 Sentences | 80 Sentences | 2 | 200 | 512 |
| S6-1 | No | HINT | Babble (non-looped) | 0 dB | -10 dB | 100 Sentences x 6 | 80 Sentences | 80 Sentences | 2 | 200 | 512 |
| S6-2 | Yes | HINT | Babble (non-looped) | 0 dB | -10 dB | 100 Sentences x 6 | 80 Sentences | 80 Sentences | 2 | 200 | 512 |
| S7-1 | No | HINT | SSN (non-looped) | -5 dB | -10 dB | 100 Sentences x 6 | 80 Sentences | 80 Sentences | 2 | 200 | 512 |
| S7-2 | Yes | HINT | SSN (non-looped) | -5 dB | -10 dB | 100 Sentences x 6 | 80 Sentences | 80 Sentences | 2 | 200 | 512 |
| S8-1 | No | HINT | SSN (non-looped) | -2 dB | -6 dB | 100 Sentences x 6 | 80 Sentences | 80 Sentences | 2 | 200 | 512 |
| S8-2 | Yes | HINT | SSN (non-looped) | -2 dB | -6 dB | 100 Sentences x 6 | 80 Sentences | 80 Sentences | 2 | 200 | 512 |

**Table 7.1:** This table presents the 16 tests which have been conducted to evaluate the performance of the speech enhancement system presented in Chapter 6.

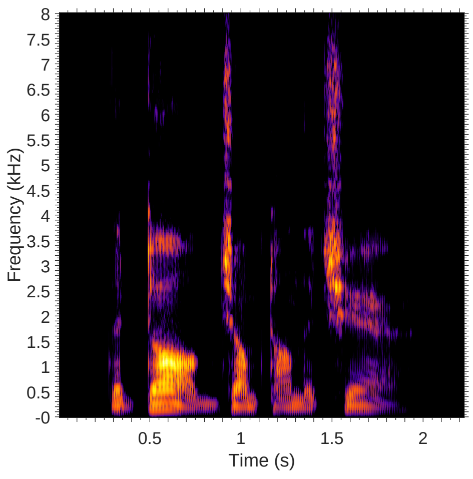
## 7.1 Visual Evaluation

Before the results are presented in terms of HIT-FA, STOI and PESQ some figures are presented to visualize noisy speech in terms of spectrograms before and after it has been processed by the IUT. The speech material used for this particular test is the fifth HINT sentence from list 16; *The dog jumped on the chair* [Nilsson et al., 1994]. The HINT sentence has been processed by the IUT using settings from test S1-2 and test S3-2 meaning that looped babble noise and looped SSN have been used with a SNR of -5 dB. The clean and noisy signals are presented in Fig. 7.1. The processed HINT sentences, with babble noise and SSN, is presented in Figs. 7.2 and 7.3, respectively.

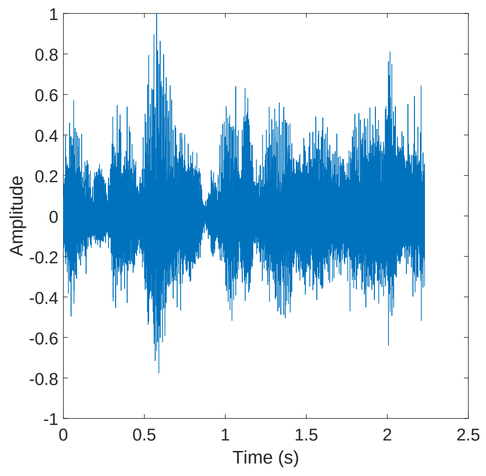
In Fig. 7.1a the time domain representation of the HINT sentence is presented and in Fig. 7.1b the spectrogram representation is presented. In Fig. 7.1c the time domain representation of the clean HINT sentence is presented with -5 dB SNR



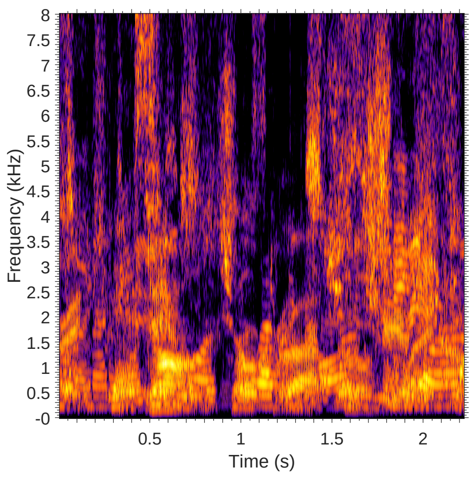
**(a)** Clean speech signal



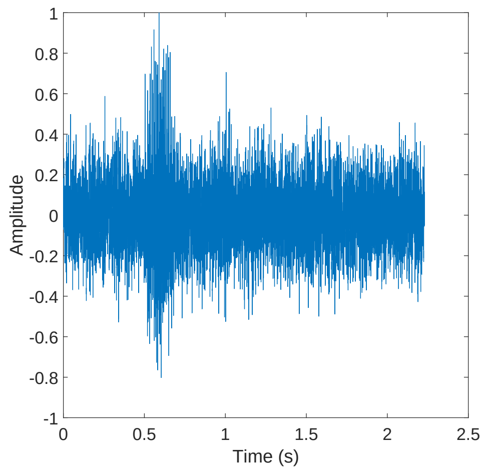
**(b)** Spectrogram of clean speech signal



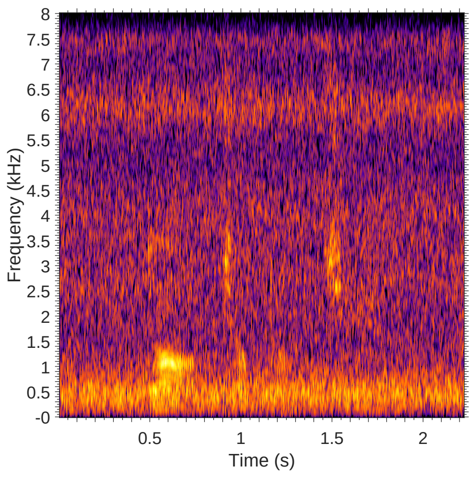
**(c)** Noisy speech signal (-5 dB SNR babble)



**(d)** Spectrogram of noisy speech signal (-5 dB SNR babble)

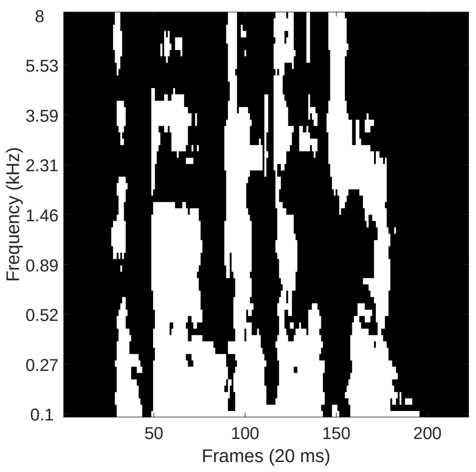


**(e)** Noisy speech signal (-5 dB SNR SSN)

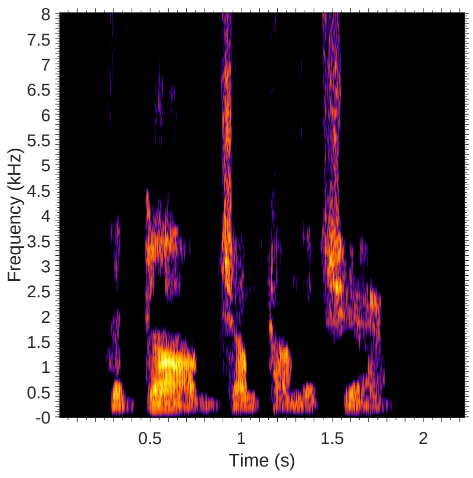


**(f)** Spectrogram of noisy speech signal (-5 dB SNR SSN)

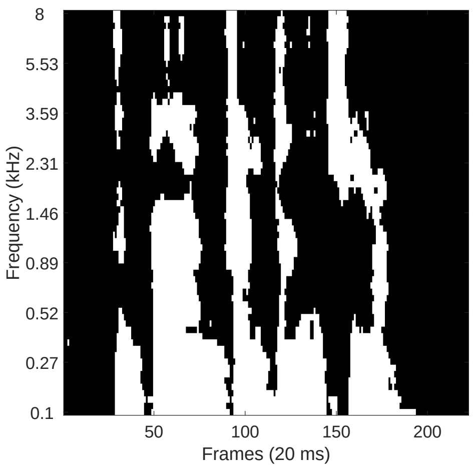
**Figure 7.1:** This figure presents a time domain representation and a spectrogram representation of a clean speech signal. Also two noisy versions of the signal is presented. One version with babble noise and one version with SSN. The speech material is the HINT sentence *The dog jumped on the chair.*



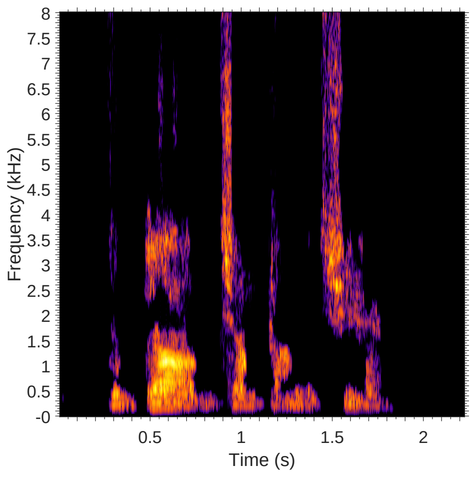
**(a)** Ideal binary mask



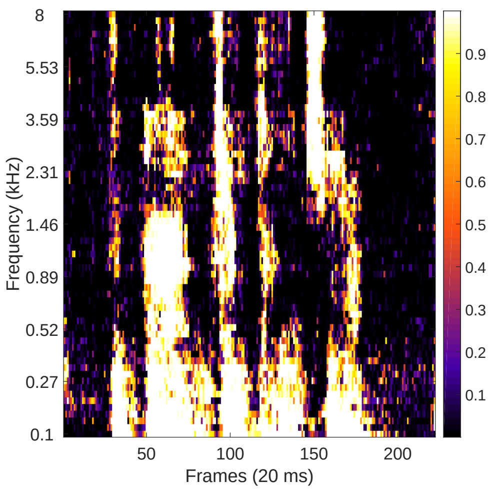
**(b)** Noisy speech signal from Fig. 7.1d processed by the IBM



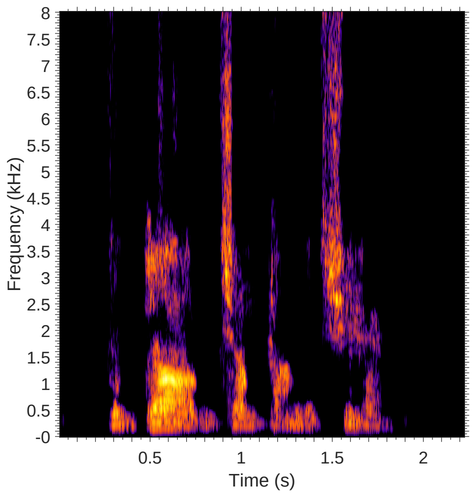
**(c)** Estimated ideal binary mask



**(d)** Noisy speech signal from Fig. 7.1d processed by the estimated IBM

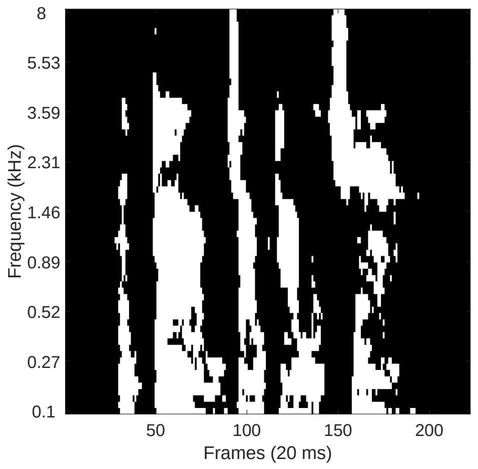


**(e)** Soft Mask

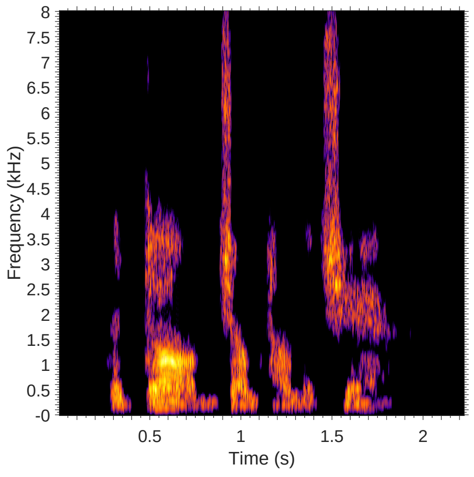


**(f)** Noisy speech signal from Fig. 7.1d processed by the soft mask

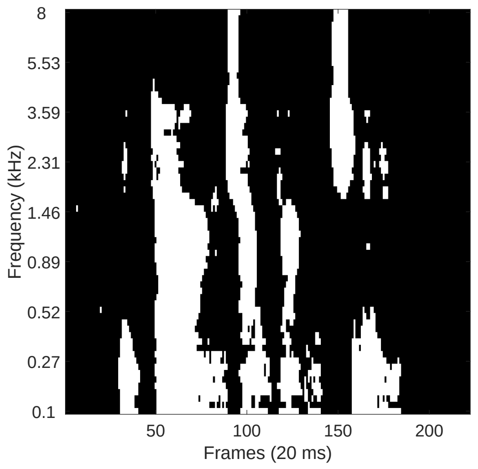
**Figure 7.2:** This figure presents the IBM, estimated IBM and the soft mask based on the HINT sentence *The dog jumped on the chair* corrupted with babble noise at -5 dB SNR (Fig. 7.1d). Also the corresponding spectrograms of the, IUT S1-2, processed version of the signal is presented.



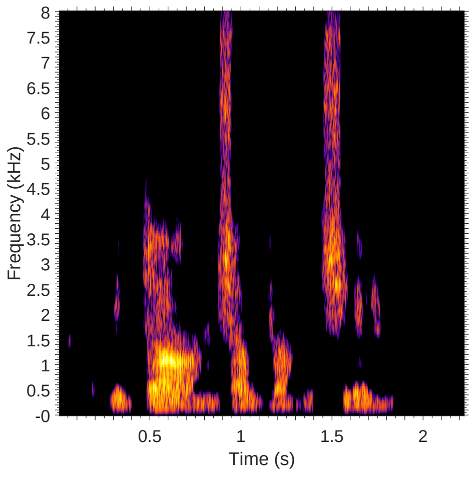
**(a)** Ideal binary mask



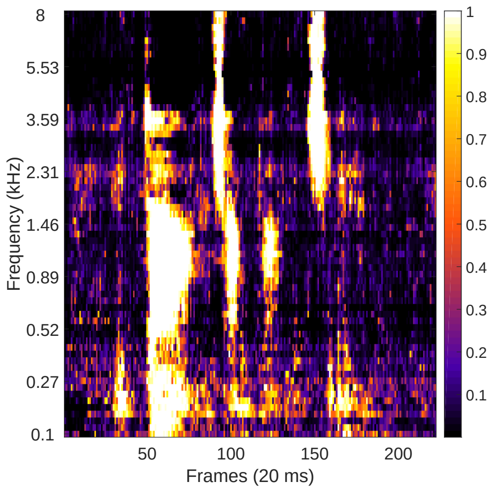
**(b)** Noisy speech signal from Fig. 7.1f processed by the IBM



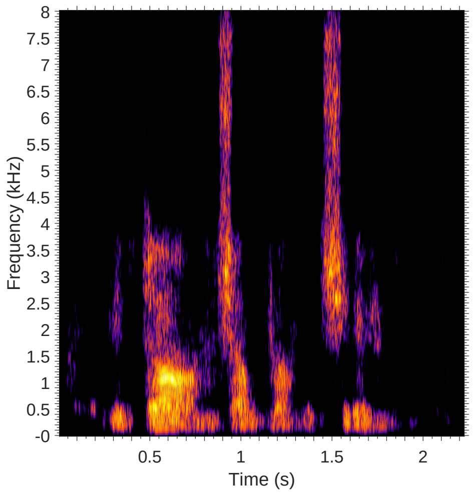
**(c)** Estimated ideal binary mask



**(d)** Noisy speech signal from Fig. 7.1f processed by the estimated IBM



**(e)** Soft Mask



**(f)** Noisy speech signal from Fig. 7.1f processed by the soft mask

**Figure 7.3:** This figure presents the IBM, estimated IBM and the soft mask based on the HINT sentence *The dog jumped on the chair* corrupted with SSN at -5 dB SNR (Fig. 7.1f). Also the corresponding spectrograms of the, IUT S3-2, processed version of the signal is presented.

babble noise added. In Fig. 7.1d the spectrogram representation of the noisy HINT sentence is presented. In Figs. 7.1e and 7.1f the same HINT sentence is presented but with SSN instead of babble noise.

By visual inspection of Figs. 7.1d and 7.1f it is clear to see the difference between babble noise which looks similar to speech and SSN which looks more similar to white noise.

In the next section the results from test S1-S8 are presented in Tab. 7.2 and Tab. 7.3 and results labeled *(NOISY)* refers to scores computed based on signals such as the ones presented in Figs. 7.1d and 7.1f. Results with the label *(Delta ...)* is referring to the improvement of a given score with reference to the corresponding *(NOISY)* score.

In Fig. 7.2 spectrograms of processed versions of the noisy signal from Fig. 7.1d are presented and similar in Fig. 7.3 spectrograms of processed versions of the noisy signal from Fig. 7.1f are presented. In Fig. 7.2a and Fig. 7.3a the IBM is presented for each of the two noisy speech signals. The IBM have been constructed using a LC of -10 dB.

In Fig. 7.2b and Fig. 7.3b a spectrogram of the IBM processed noisy speech signal is presented for the babble noise distorted signal and the SSN distorted signal respectively. It is seen that for both the babble noise distorted signal and SSN distorted signal, the noise have been removed and the majority of the speech structures from the clean speech signal in Fig. 7.1b is preserved.

The IBM, as the name implies, is the ideal scenario and is what the DNNs have been trained to estimate. This means that the upper bound on the classification performance of the DNNs is the IBM and the distortion introduced by the IBM cannot be resolved by the DNNs. In Tab. 7.2 and Tab. 7.3, in the next section, the label *(IBM)* indicates that signals have been processed by the IBM and is based on signals such as the ones presented in Fig. 7.2b and Fig. 7.3b.

In Fig. 7.2c and Fig. 7.3c the estimated IBM is presented and the corresponding processed signals, using this estimate, is presented in Fig. 7.2d and Fig. 7.3d respectively. It is seen that the IBMs and the estimated IBMs share a lot of similarities and so do the spectrograms of the IBM processed signals and the spectrogram of the estimated IBM processed signals. In Tab. 7.2 and Tab. 7.3, in the next section, labels with *(IBM ESTIMATE)* refer to results based on signals processed with the estimated IBM such as Fig. 7.2d and Fig. 7.3d. The labels *Correctly Classified*, *HIT*, *FA* and *HIT-FA* have been computed based on an IBM and an estimated IBM such as the ones presented in Fig. 7.2 and Fig. 7.3

In Fig. 7.2e and Fig. 7.3e the soft mask is presented, which is the unprocessed output of the DNNs. The soft mask is the data used to perform the binary classification used in the construction of the estimated IBM. The corresponding processed signals are presented in Fig. 7.2f and Fig. 7.3f respectively. From the soft masks it is obvious that not all parts of the estimated IBMs have been constructed with the same confidence, meaning that some T-F units have been preserved with only a small amount of evidence and similar some T-F units have been discarded.

The hypothesis behind using the soft mask is that since the DNNs are not equally confidence in all its decisions, better results might be achieved by utilizing this uncertainty using a soft mask instead of a binary mask. From Fig. 7.2f and Fig. 7.3f it is not obvious to see that there is a significant difference between the two processed signals. However, as presented in the next section, results labeled with *(SOFT MASK)* always achieves higher PESQ and STOI scores compared to the scores achieved by the estimated IBM.

From Figs. 7.2e and 7.3e it is also observed that the SSN seem to be harder to discriminate since the uncertainty is considerably larger in Fig. 7.3e compared to Fig. 7.2e where babble noise was use. This is not in line with intuition and might be explained by the fact that the babble noise sequence is 5s compared to the SSN sequence which is 10s. This means that the DNNs only have to learn 5s of noise in the case of babble noise, which seem to be an easier job compared to learning 10s of SSN, even though the SSN is an easier noise type. As seen from the results from Tab. 7.2 and Tab. 7.3, presented in the next section, SSN seem to be an easier noise type when tested in more comparable scenarios using non-looped noise.

## 7.2 Effects of Pre-training, Looped Noise & Soft Mask

This section presents the results from test S1-1/2 - S8-1/2 where the IUT has been tested using the same settings as it was trained with, i.e., the same noise type and SNRs. These results are related to objective 1, 2 and 3 from the introduction in Chapter 1.

To investigate how the IUT performs compared to the system presented in [Healy et al., 2013] the results from [Healy et al., 2013] will also be presented, which are available from [Wang, 2013].

The results are divided into two tables. All results related to tests where babble noise is used, is presented in Tab. 7.2 and all results related to tests where SSN is used, is presented in Tab. 7.3. The results are presented in terms of classifier accuracy, HIT-FA, STOI and PESQ. All scores are computed as an average over the 80 HINT sentences in the test set. Early stopping with a lookahead of 50 epochs has been used in these tests.

In Tab. 7.2 the results from test S1-1/2, S2-1/2, S5-1/2 and S6-1/2 are presented and in Tab. 7.3 the results from test S3-1/2, S4-1/2, S7-1/2 and S7-1/2 are presented. The test ID *DW* is short for DeLiang Wang and is representing the results from [Healy et al., 2013] [Wang, 2013].

It is seen from Tab. 7.2 that both S1-1/2 and S2-1/2 have better classification accuracy and lower FA than reported by [Healy et al., 2013] but a higher HIT and a slightly lower STOI (Delta Est. IBM). S1-1/2 and S2-1 achieves a HIT-FA slightly lower than reported by [Healy et al., 2013] and S2-2 achieves a HIT-FA slightly higher.

From Tab. 7.3 it is seen that S3-1/2 for all performance measures except FA, performs significantly worse than the system from [Healy et al., 2013]. S4-1/2 also

| Corpus - Noise Type: | HINT - Babble | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| Noise Realizations: | Looped Noise | | | | | | Non-looped Noise | | | |
| SNR: | -5 dB | | | 0 dB | | | -5 dB | | 0 dB | |
| Test ID: | S1-1 | S1-2 | DW | S2-1 | S2-2 | DW | S5-1 | S5-2 | S6-1 | S6-2 |
| Correctly Classified | 91.5 % | 91.8 % | 90.6 % | 92.0 % | 92.2 % | 90.4 % | 84.3 % | 84.3 % | 85.8 % | 85.9 % |
| HIT | 82.9 % | 83.5 % | 89.9 % | 85.1 % | 85.8 % | 89.6 % | 67.7 % | 67.3 % | 73.3 % | 73.7 % |
| FA | 4.83 % | 4.70 % | 9.13 % | 4.81 % | 4.90 % | 9.08 % | 7.88 % | 7.78 % | 7.82 % | 7.86 % |
| HIT-FA | 78.1 % | 78.8 % | 80.9 % | 80.3 % | 80.9 % | 80.4 % | 59.8 % | 59.6 % | 65.5 % | 65.8 % |
| STOI (IBM) | 0.834 | 0.834 | - | 0.884 | 0.884 | - | 0.849 | 0.849 | 0.895 | 0.895 |
| STOI (IBM ESTIMATE) | 0.738 | 0.748 | 0.756 | 0.818 | 0.828 | 0.827 | 0.591 | 0.592 | 0.714 | 0.722 |
| STOI (SOFT MASK) | 0.755 | 0.762 | - | 0.834 | 0.837 | - | 0.659 | 0.653 | 0.765 | 0.768 |
| STOI (NOISY) | 0.537 | 0.537 | 0.531 | 0.666 | 0.666 | 0.661 | 0.545 | 0.545 | 0.673 | 0.673 |
| STOI (Delta Est. IBM) | 0.201 | 0.211 | 0.224 | 0.153 | 0.162 | 0.167 | 0.046 | 0.047 | 0.042 | 0.049 |
| STOI (Delta SOFT MASK) | 0.218 | 0.225 | - | 0.168 | 0.171 | - | 0.115 | 0.108 | 0.092 | 0.096 |
| PESQ (IBM) | 1.140 | 1.140 | - | 1.222 | 1.222 | - | 1.168 | 1.168 | 1.266 | 1.266 |
| PESQ (IBM ESTIMATE) | 1.072 | 1.074 | - | 1.114 | 1.128 | - | 1.042 | 1.042 | 1.067 | 1.068 |
| PESQ (SOFT MASK) | 1.132 | 1.129 | - | 1.213 | 1.194 | - | 1.109 | 1.104 | 1.186 | 1.196 |
| PESQ (NOISY) | 1.056 | 1.056 | - | 1.077 | 1.077 | - | 1.061 | 1.061 | 1.088 | 1.088 |
| PESQ (Delta Est. IBM) | 0.017 | 0.018 | - | 0.037 | 0.051 | - | -0.019 | -0.019 | -0.021 | -0.020 |
| PESQ (Delta SOFT MASK) | 0.076 | 0.073 | - | 0.136 | 0.118 | - | 0.048 | 0.043 | 0.098 | 0.108 |

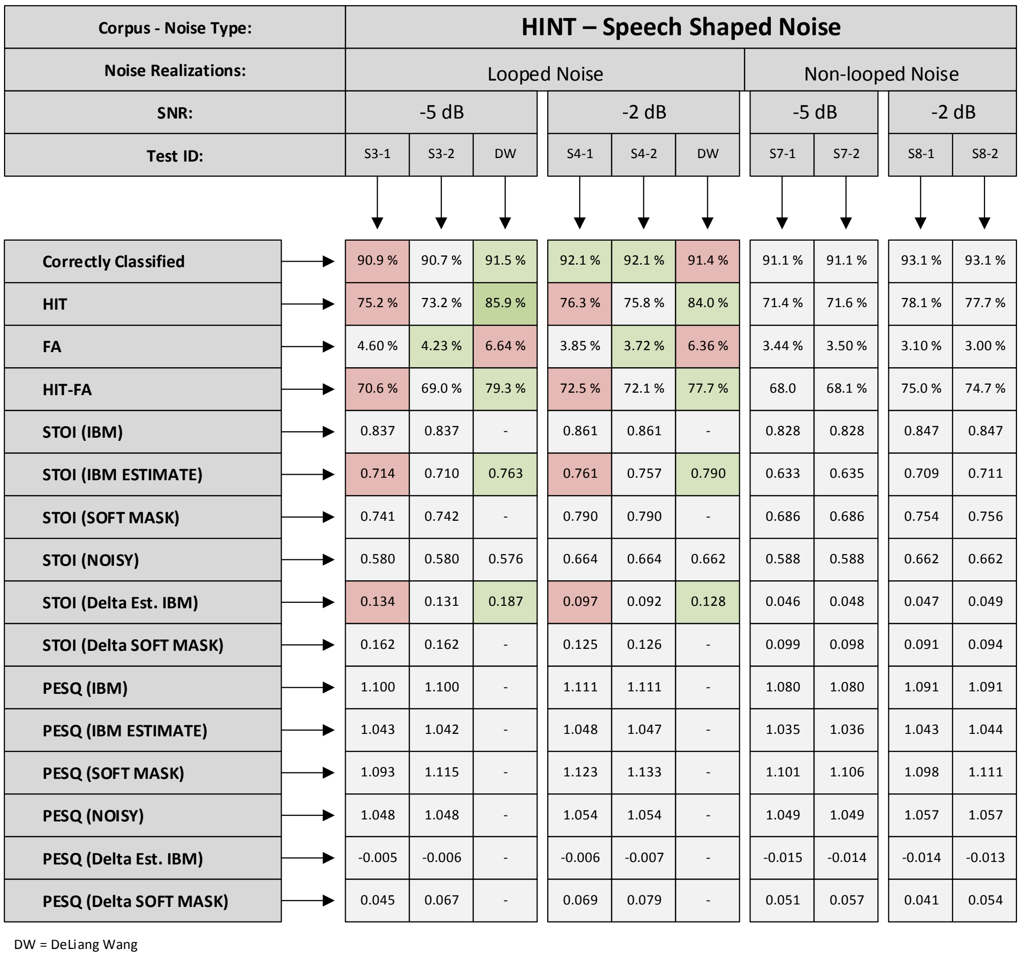
DW = DeLiang Wang

**Table 7.2:** System evaluation results for IUTs S1-1/2, S2-1/2, S5-1/2 and S6-1/2 where both looped and non-looped babble noise has been used

perform significantly worse than the system from [Healy et al., 2013] regarding HIT, HIT-FA and STOI even though it has a higher classification accuracy.

The IUT used to produce the results in Tabs. 7.2 and 7.3 are the same. It is a surprise that the performance of the IUTs are comparable to the results reported by [Healy et al., 2013] only when babble noise is used and significantly worse when SSN is used. Since SSN is a stationary noise type, it was considered as an easier noise type compared to babble noise, since babble noise is non-stationary. However, such reasoning might not be so trivial based on the results in Tab. 7.2 and Tab. 7.3.

The performance difference between the IUT and the system reported in [Healy et al., 2013] is due to differences between the two systems, indicating that either there is an error in the IUT or the system described in [Healy et al., 2013] is different



| Corpus - Noise Type: | HINT – Speech Shaped Noise | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| Noise Realizations: | Looped Noise | | | | | | Non-looped Noise | | | |
| SNR: | -5 dB | | | -2 dB | | | -5 dB | | -2 dB | |
| Test ID: | S3-1 | S3-2 | DW | S4-1 | S4-2 | DW | S7-1 | S7-2 | S8-1 | S8-2 |
| Correctly Classified | 90.9 % | 90.7 % | 91.5 % | 92.1 % | 92.1 % | 91.4 % | 91.1 % | 91.1 % | 93.1 % | 93.1 % |
| HIT | 75.2 % | 73.2 % | 85.9 % | 76.3 % | 75.8 % | 84.0 % | 71.4 % | 71.6 % | 78.1 % | 77.7 % |
| FA | 4.60 % | 4.23 % | 6.64 % | 3.85 % | 3.72 % | 6.36 % | 3.44 % | 3.50 % | 3.10 % | 3.00 % |
| HIT-FA | 70.6 % | 69.0 % | 79.3 % | 72.5 % | 72.1 % | 77.7 % | 68.0 | 68.1 % | 75.0 % | 74.7 % |
| STOI (IBM) | 0.837 | 0.837 | - | 0.861 | 0.861 | - | 0.828 | 0.828 | 0.847 | 0.847 |
| STOI (IBM ESTIMATE) | 0.714 | 0.710 | 0.763 | 0.761 | 0.757 | 0.790 | 0.633 | 0.635 | 0.709 | 0.711 |
| STOI (SOFT MASK) | 0.741 | 0.742 | - | 0.790 | 0.790 | - | 0.686 | 0.686 | 0.754 | 0.756 |
| STOI (NOISY) | 0.580 | 0.580 | 0.576 | 0.664 | 0.664 | 0.662 | 0.588 | 0.588 | 0.662 | 0.662 |
| STOI (Delta Est. IBM) | 0.134 | 0.131 | 0.187 | 0.097 | 0.092 | 0.128 | 0.046 | 0.048 | 0.047 | 0.049 |
| STOI (Delta SOFT MASK) | 0.162 | 0.162 | - | 0.125 | 0.126 | - | 0.099 | 0.098 | 0.091 | 0.094 |
| PESQ (IBM) | 1.100 | 1.100 | - | 1.111 | 1.111 | - | 1.080 | 1.080 | 1.091 | 1.091 |
| PESQ (IBM ESTIMATE) | 1.043 | 1.042 | - | 1.048 | 1.047 | - | 1.035 | 1.036 | 1.043 | 1.044 |
| PESQ (SOFT MASK) | 1.093 | 1.115 | - | 1.123 | 1.133 | - | 1.101 | 1.106 | 1.098 | 1.111 |
| PESQ (NOISY) | 1.048 | 1.048 | - | 1.054 | 1.054 | - | 1.049 | 1.049 | 1.057 | 1.057 |
| PESQ (Delta Est. IBM) | -0.005 | -0.006 | - | -0.006 | -0.007 | - | -0.015 | -0.014 | -0.014 | -0.013 |
| PESQ (Delta SOFT MASK) | 0.045 | 0.067 | - | 0.069 | 0.079 | - | 0.051 | 0.057 | 0.041 | 0.054 |

DW = DeLiang Wang

**Table 7.3:** System evaluation results for IUTs S3-1/2, S4-1/2, S7-1/2 and S8-1/2 where both looped and non-looped SSN noise has been used

from the system that actually produced the results from [Wang, 2013].

Since errors never are intentional, the focus on explaining the performance difference between the IUT and the system from [Healy et al., 2013], will be on actual known differences between the two systems.

### 7.2.1 Plausible Causes for Deviating Performance

This section presents some plausible causes which might explain the deviating performance of IUTs S3-1/2 and S4-1/2 and the results reported in [Healy et al., 2013] [Wang, 2013].

Several vital details needed to reconstruct the system in [Healy et al., 2013] are unknown. These are, among other things, the number of epochs for pre-training, the number of epochs for backpropagation training as well as the learning parameters. It is also unknown what kind of output units the output layer of the DNN consists of and what variance has been used for the GBRBM. Furthermore it is unknown if the features have been normalized to have unit variance and zero mean. All these unknowns have a direct impact on the performance of the system and the settings used for the IUT can be found in Appendix E.

For example in [Le et al., 2011a] it was found that different optimization methods leads to different results. In the IUT non-linear Conjugate Gradient is used and in [Healy et al., 2013] it is reported that gradient descent is used. However, in [Healy et al., 2013] they refer to [Wang and Wang, 2013] regarding the training algorithm in which Limited-memory Broyden–Fletcher–Goldfarb–Shanno (L-BFGS) is used. Since the optimization method used in [Healy et al., 2013] differs from the one used by the IUT, it is difficult to directly compare the two systems, however, this contribution to the deviating performance is considered as a minor one.

Another explanation of the deviating performance of the two systems might be due to the way the noise is generated. In [Healy et al., 2013] it is reported that the noise sequence is constructed randomly by selecting a starting point of the noise sequence. A noise sequence as long as the corresponding HINT sentence is then extracted starting from the randomly chosen starting point. It is not reported what happens if the extracted noise sequence is shorter than the HINT sentence but since the word "looped noise" is used, it is assumed by the authors, that the remaining noise sequence is taken from the beginning of the noise sequence. However, in [Wang, 2013] it is reported that the noise sequence is split into 10 subsequences and a noise sequence is then generated by randomly permuting these 10 subsequences. This approach is different from the one reported in [Healy et al., 2013], even though they in [Wang, 2013] directly refer to [Healy et al., 2013].

A comparison of the two approaches has not been investigated due to time limitations, but it is disconcerting that the author of [Healy et al., 2013] report two different methods without being explicit about what method has been used when.

As mentioned in Section 6.4, it was found hard to believe that the system in [Healy et al., 2013] trained each subband DNN for the same amount of epochs, since it has been found by the authors, that different subband DNNs peaked at different epochs.

In [Narayanan and Wang, 2013] which is a paper about ASR using DNNs they report that cross validation and early stopping has been used. They even refer to [Wang and Wang, 2013] regarding the DNN based IBM estimation procedure, which is similar to what they do in [Healy et al., 2013]. In [Narayanan and Wang, 2013], similar to [Healy et al., 2013], information such as learning parameters and algorithm is missing, information which are crucial if the results are to be reproduced. Since [Narayanan and Wang, 2013], [Wang and Wang, 2013] and [Healy et al., 2013] share the same senior author as well as referring to each other, it is natural to think that the DNN used in the three papers share similarity and combined with the inadequate amount of details in the papers, there might be some details which the authors of this thesis are unaware of hence, preventing the system described in Chapter 6 to perform as well as the system described in [Healy et al., 2013].

Another unknown parameter is the threshold for which the binary classification has been performed. This parameter, defining at what level a T-F unit is classified as either speech or noise dominated, is known to have a great impact on the performance of a binary classifier [Fawcett, 2006]. The IUT uses 0.5 as the threshold. A way to evaluate the performance of a binary classifier is to identify the Receiver Operating Characteristics (ROC), which is a way to illustrate how the HIT rate of a binary classifier relates to the FA rate, as function of the threshold for which a true positive, i.e., a HIT is classified.

### Receiver Operating Characteristic (ROC) Analysis

Since the results from Tabs. 7.2 and 7.3 show that the IUT generally achieves lower HIT rates but also lower FA rates, compared to the results from [Healy et al., 2013] it might be due to the classifiers from the two systems operate in different regions of the ROC space [Fawcett, 2006]. To investigate this hypothesis the ROC curves related to test S1-1 - S8-1 have been computed, based on the test set, and is depicted in Fig. 7.4. With a similar approach, curves that represent HIT-FA as function of threshold as well as STOI as function of threshold are depicted in Fig. 7.4.

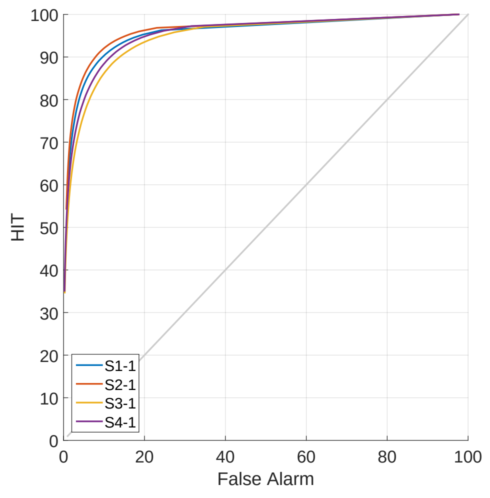
Figures 7.4a, 7.4c and 7.4e are related to tests S1-1 to S4-1 and Figs 7.4a, 7.4d and 7.4f are related to tests S5-1 to S8-1 and will be commented upon in Section 7.2.2.

As seen from Fig. 7.4a all four ROC curves related to S1-1 to S4-1 are located primarily in the northwest quadrant of the ROC space, having relatively large HIT rates compared to FA rates. The curves are also way above the diagonal line (gray), which is the performance of a randomized classifier. It is also seen that S1-1 and S2-2 performs the best, having the largest Area Under ROC curve (AUC), which also correlate well to the results in Tabs. 7.2 and 7.3 [Fawcett, 2006].

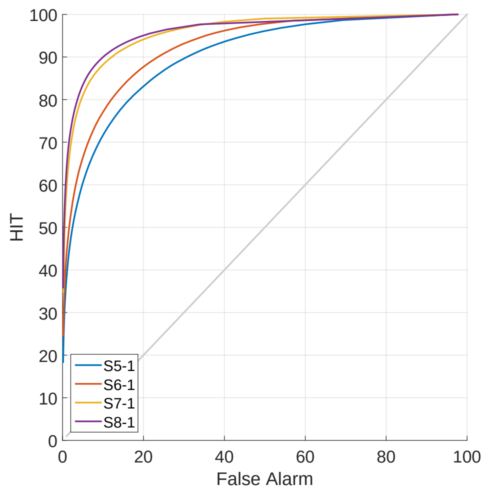
From Fig. 7.4 it is seen that the IUT performance and especially HIT-FA, but also to some extent STOI, are relatively sensitive to the value of the threshold and a threshold of 0.5 is not optimal. Identifying the optimal threshold using the validation set and then use different thresholds for different IUTs, might hereby improve overall performance. The horizontal dashed lines on Fig. 7.4e and Fig. 7.4f are the STOI scores when the soft mask is used, which is independent of the threshold since no classification is performed when the soft mask is applied. It is hereby seen that the soft mask outperforms the binary mask, regardless of the value of the threshold!

It is also seen that if the threshold is set to maximize HIT-FA, the HIT-FA rates for S1-1, S2-1 and S4-1 is even higher than the HIT-FA rates reported in [Healy et al., 2013]. Also an increase in STOI can be achieved by modifying the threshold.

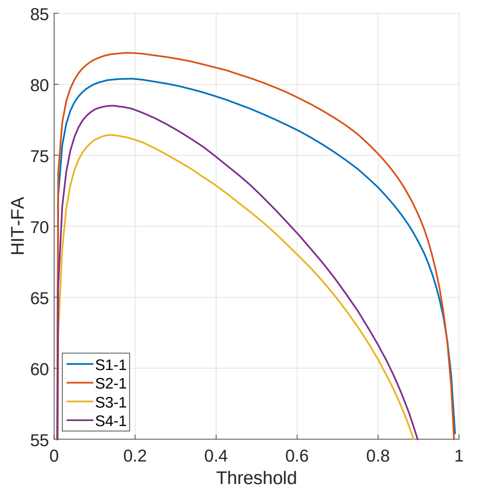
Based on the results in Figs. 7.4a, 7.4c and 7.4e it can be concluded that the performance of the IUT, when a binary mask is used, can be improved with respect to HIT-FA and STOI by modifying the threshold. However, the optimal threshold might depend on the global SNR hence, the global SNR needs to be known *a priori* in order to choose such optimal threshold.



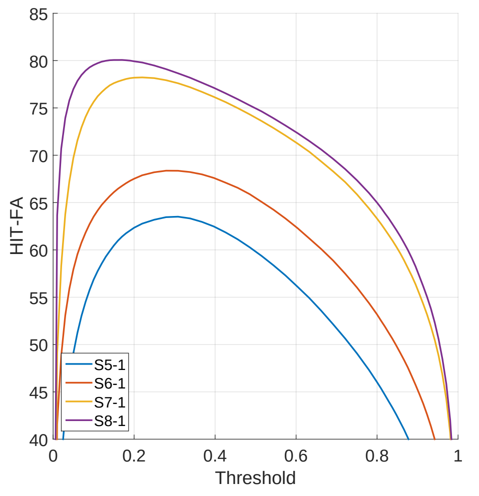
**(a)** ROC Curve for test S1-1 - S4-1



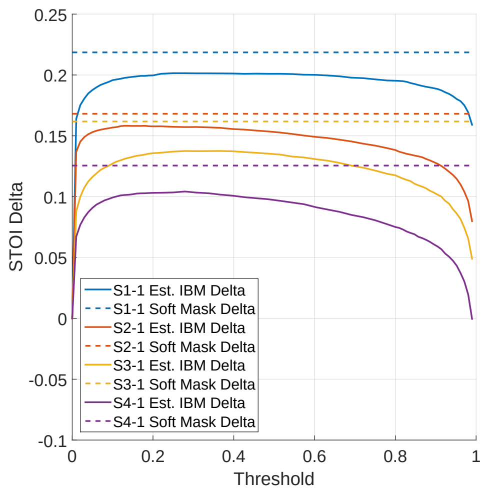
**(b)** ROC Curve for test S5-1 - S8-1



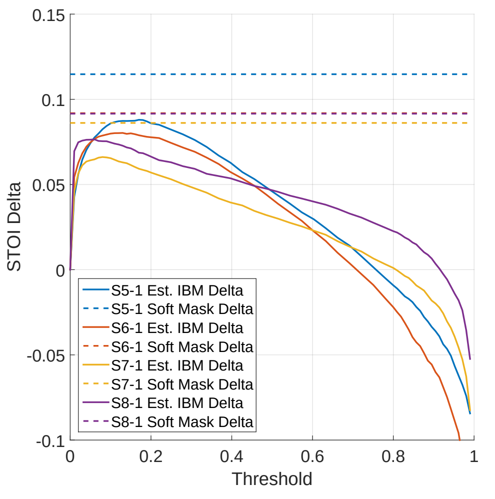
**(c)** HIT-FA vs. threshold for test S1-1 - S4-1



**(d)** HIT-FA vs. threshold for test S5-1 - S8-1



**(e)** STOI vs. threshold for test S1-1 - S4-1



**(f)** STOI vs. threshold for test S5-1 - S8-1

**Figure 7.4:** ROC Curves for IUTs S1-1 to S8-1. It is seen that the performance of the IUTs, especially relating to HIT-FA is highly dependent on the threshold used in the binary classification. It is also seen that setting the threshold is more crucial when non-looped noise is used but regardless of the noisy type and the threshold the soft mask achieves the best performance.

### 7.2.2 Discussion of Objective 1, 2 & 3

This section will, based on Tab. 7.2 and Tab. 7.3, discuss the results and relate them to the objectives defined in Chapter 1.

#### Discussion of Objective 1

Regarding objective 1 it is seen that all tests with a -2 ID in Tab. 7.2 generally have a slightly higher value of correctly classified compared with the tests having a -1 ID. The tests with a -2 ID have been pre-trained whereas the tests with a -1 ID have only been trained with backpropagation. From these results it could indicate that pre-training has a beneficial effect when babble noise is used.

When looking at Tab. 7.3 a different pattern is seen, where pre-training in general have a worse or equal performance compared to the tests where only backpropagation training have been applied. This argues against that pre-training has an effect. In [Maas et al., 2012], which is a paper from Andrew Ng who is a leading Deep Learning scientist, it was reported that pre-training was not found beneficial for the current application. Also, in [Deng et al., 2013], which is a paper from Geoffrey Hinton who is also leading Deep Learning scientist, it was argued that the benefit of pre-training was highly dependent on the amount of labeled training data and in the same time required a lot of experience to implement in practice.

Based on these publications, the results presented in Tab. 7.2 and Tab. 7.3 might not be a consequence of an erroneous IUT but merely a learning algorithm not properly tuned in combination with training data for which pre-training is not beneficial.

To support this argumentation a small test was conducted where the settings of S7 was used. The difference were that the amount of training data used for backpropagation training was reduced to 150 HINT sentences but still keeping the 600 HINT sentences for pre-training. The test was run with and without pre-training and it was found that the test where pre-training was utilized 77% correctly classified were achieved, whereas the test without pre-training only achieved 51%, which is a significant difference. This test supports the argument in [Deng et al., 2013] claiming that if enough labeled data is available, pre-training is not beneficial.

Another effect that may explain the difference in pre-training performance is the amount of overfitting that may occur during pre-training, which may be data specific. All DNNs were pre-trained for 100 epochs and it is likely to believe that different DBNs starting to overfit at different epochs. However, measuring overfitting in DBNs is not as simple as for DNNs but it may improve performance of the IUT if an early stopping algorithm was implemented for the DBN training as well [Hinton, 2012].

It should also be mentioned that the DNNs used in the IUT is trained discriminative using backpropagation based on the cross entropy error function. This means that it is the classification error that is minimized during DNN training and not STOI, HIT-FA or PESQ. It can hereby not be expected that DNN training can have a positive effect on other measures than the classification accuracy. How the DNN training relates to STOI, HIT-FA or PESQ is definitely interesting but is outside the scope of this thesis.

From the results in Tabs. 7.2 and 7.3 it can not be concluded that pre-training is beneficial in for the current design and training setup. However, from the small test just described as well as the observations reported in the literature, pre-training can be beneficial and can improve the performance of DNNs, but utilizing the power of pre-training is not a trivial task and requires fine-tuning of the training algorithm.

### Discussion of Objective 2

Regarding objective 2, which was related to the impact of using looped noise, the following can be observed from Tab. 7.2 and Tab. 7.3. When non-looped babble noise is used, i.e. test ID S5-1/2 and S6-1/2, it is observed that the performance generally decreases compared to the tests where looped noise have been used, which are S1-1/2 and S2-1/2. This applies for both the number of correctly classified T-F units as well as for HIT-FA, STOI and PESQ. This is also in line with the findings in [May and Dau, 2014], even though a direct comparison cannot be made since a different classifier, noise type and speech corpus were used. What is important to notice from S5-1/2 and S6-1/2 is that according to the STOI scores, improvement in intelligibility is still significant when non-looped noise is used.

Quite different is it when non-looped SSN is used as seen from tests S7-1/2 and S8-1/2 in Tab. 7.3. Tests S7-1/2 achieve a higher classification accuracy than S3-1/2 but a slightly lower HIT-FA. Tests S7-1/2 also achieve a higher classification accuracy but also a higher HIT-FA than S4-1/2, which is counterintuitive and not in line with the results reported in [May and Dau, 2014]. However, according to the STOI improvement both S7-1/2 and S8-1/2 perform worse than S3-1/2 and S4-1/2

When visualizing the performance of the classifiers using the ROC curves, similar patterns are observed. By computing the AUCs for S3-1, S4-1, S7-1 and S8-1 it is verified that the AUC for S3-1 and S4-1 are approximately 1% lower than the AUCs for S7-1 and S8-1. This is quite different from the AUCs for S1-1 and S2-1 relative to the AUCs for S5-1 and S6-1, which are approximately 5% higher. However, by observing Fig. 7.4e and Fig. 7.4f it is seen that the IUTs trained with non-looped noise have a STOI score which is generally lower than the IUTs trained with looped noise, which is more in line with what was expected.

To investigate the counterintuitive behaviour of S3-1, S4-1, S7-1 and S8-1 two additional test have been performed. The motivation behind the two tests is to exclude suspected errors in the code. The tests, S9 and S10, are based on the settings of S3-1 and S7-1 but with reduced duration of noise sequences. Test S9 uses the noise sequence from S3 but the duration is reduced to the length of a HINT sentence such that the exact same noise sequence is used for all training and test cases. S10 uses the same noise sequence as S7 and similarly to S9, the duration is reduced to the length of a HINT sentence such that the exact same noise sequence is used for all training and test cases. The goal of the tests is to verify that by using a very short noise sequence, the performance should increase, regardless of the noise type.

| | **S3-1** | **S9** | **S7-1** | **S10** |
|---|---|---|---|---|
| **Correctly Classified:** | 90.9% | 91.7% | 91.1% | 93.0% |
| **HIT-FA:** | 70.6% | 73.6% | 68.0% | 76.6% |
| **STOI (Delta SOFT MASK):** | 0.162 | 0.192 | 0.100 | 0.167 |

**Table 7.4:** This table presents the system evaluation results where a reduced noise duration is used. The settings and the noise type of S3-1 and S7-1 are used for simulation S9 and S10 respectively but the duration of the noise sequence is reduced such that the same realization is used for all training and test cases. This is done to exclude errors in the implementation.

The results from the two tests are presented in Tab. 7.4 and it is seen that both S9 and S10 have significant higher scores than S3-1 and S7-1 for all three reported performance measures. These results follow intuition and may argue that S3 and S7 are not comparable since two different noise sources are used. Even though they are both of the type SSN, they are not from the same realization, which could explain the counterintuitive behaviour of S3-1, S4-1, S7-1 and S8-1. Maybe the noise source used for S7 and S8 is easier to discriminate compared to the noise source used for S3 and S4, even though their duration differs significantly.

Based on Tabs. 7.2, 7.3 and 7.4 as well as the ROC curves, it is concluded that using looped noise yields classifier performance which is significantly better than if the classifier is trained and tested using non-looped noise. However, it is found that comparing IUTs trained with different noise realizations, even though they are of the same type, yield no useful information and should be avoided. Another thing that can be concluded is that, even though the performance of the IUT decreases when non-looped noise is used, the IUT still improves speech intelligibility, which is vital if the IUT is to be used in a real life scenario. The last thing to conclude is that setting the threshold at which T-F units are classified as speech dominated is crucial and is found to affect the performance of the IUT, when the binary mask is used, considerably.

### Discussion of Objective 3

Regarding objective 3 it was assumed that using a soft mask for speech synthesis, a better speech quality would be achieved and presumably also a better speech intelligibility. From Tabs. 7.2 and 7.3 it is seen that all 16 tests achieve the highest PESQ increase when the soft mask is used for speech synthesis. When the estimated IBM is used an average delta PESQ of 0 is achieved meaning that the estimated IBM cannot increase speech quality. However, when the soft mask is used the average increase is approximately 0.1 which in many cases is even better than the PESQ score when the IBM is used. These results were expected and in line with the literature [Loizou, 2013].

Regarding intelligibility it is seen from Tabs. 7.2 and 7.3 that the largest STOI scores are achieved using the soft mask for all 16 simulations. The STOI score using the soft mask is on average 0.04 larger than the binary mask. These results indicate that using a soft mask may increase speech intelligibility and with the current IUT the soft mask even outperformed the estimated IBM with considerable improvements in quality as well.

As already mentioned, it is seen from Figs. 7.4e and 7.4f that the STOI score for the soft mask (horizontal dashed lines) is the largest for all values of threshold, regardless of the use of looped noise. This supports that using the soft mask for speech synthesis yields better performance than using a binary mask, at least for the IUT. It is also seen from Fig. 7.4f that the STOI score, in the case where non-looped noise is used, is highly dependent on the threshold value, which again is a good argument for choosing the soft mask approach.

From listening tests performed by the authors it is also believed that the soft mask is better at preserving consonants such as *s*, *f* and *n* sounds, for which the binary mask in some cases completely remove these sounds.

A benefit of using a soft mask is that the classification stage is avoided and the estimation of the optimal threshold is hereby unneeded. So using the soft mask simplifies the IUT, while still achieving better performance than using a binary mask.

It must be mentioned that the soft mask becomes approximately binary if the classification problem is "easy". For example in test S9, where a short noise sequence was used, only a negligible difference between the soft mask and the binary mask was observed but still, there is no good argument for making a binary classification instead of using the output probabilities of the DNNs.

## 7.3 Evaluation of Robustness & Comparison to Classical Methods

This section presents the results which are related to objective 4 and 5 as presented in the introduction in Chapter 1. Objective 4 is related to the robustness of the proposed speech enhancement system and objective 5 is related to the performance of the proposed system compared to more classical speech enhancement algorithms.

As mentioned in Chapter 1, robustness can be evaluated in an numerous number of ways and in Chapter 1 it was decided to evaluate the robustness of the IUT regarding variations in noise type and SNR. To delimitate the number of tests to run, only the IUTs that have been trained without pre-training and with non-looped noise, will be tested. Furthermore, the IUTs will only be tested with one unseen noise type. The IUTs subject to these tests are therefore S5-1 to S8-1.

Regarding the comparison of the proposed system to more classical speech enhancement methods, it was decided to evaluate the performance of the Wiener filtering method described in Section 3.1.2 and the MMSE method described in Section 3.1.3. These methods will be subject to the same test settings as IUT S5-1 to S8-1. The IUTs and the classical methods will both be tested with SNRs from -10 dB to +30 dB, for both babble noise and SSN. Both STOI and PESQ will be used to evaluate their performance.

The results from the tests mentioned above are presented in Figs 7.5 and 7.6. Fig 7.5 presents the performance of the IUTs and the classical methods, when they are tested with babble noise and similarly, Fig 7.6 presents the results where SSN is used. All results are based on an average over the a test set consisting of 80 HINT sentences.

Figs 7.5a, 7.5b, 7.6a and 7.6b illustrate the performance difference between a IUT and the classical methods, evaluated using STOI and PESQ. The black line is produced from the noisy speech signal and is hereby a reference indicating when a given algorithm improves or degrades the intelligibility or quality.

Figs 7.5c, 7.5d, 7.6c and 7.6d illustrate the performance difference, evaluated using STOI and PESQ, between a IUT using the soft mask and the same IUT using a binary mask.

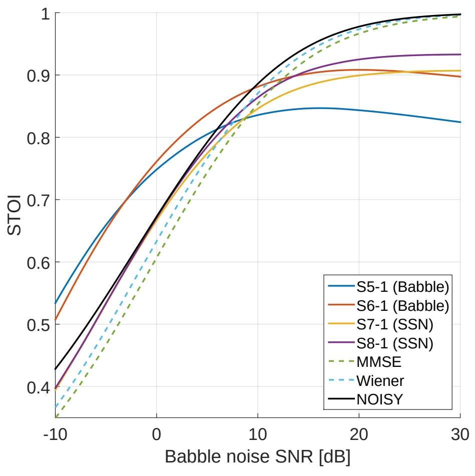
It is seen from Figs 7.5a and 7.5b that IUT S5-1 and S6-1 outperform the Wiener filtering method and the MMSE method for SNRs below 10 dB. Similarly, IUT S5-1 and S6-1 also outperform IUT S7-1 and S8-1. However, this were expected since IUT S5-1 and S6-1 are trained with babble noise, whereas IUT S7-1 and S8-1 are trained with SSN. It is also seen that IUT S7-1 and S8-1, even though they are trained with SSN, achieve a higher STOI score than the Wiener filtering method and the MMSE method for SNRs below 5 dB. In general, it can be observed that the IUT S6-1, on average, performs better than IUT S5-1, which is natural since IUT S6-1 is trained with 0 dB SNR whereas IUT S5-1 is trained at -5 dB SNR.

What is interesting to notice is that the classical method consistently follows the noisy curve (black line) and achieves no improvement in intelligibility and only minor improvement in quality. These results are also in agreement with Philipos C. Louizou findings saying that the methods could improve the quality but not the intelligibility [Loizou, 2007, p.609]. However, the IUTs introduce some distortion at high SNRs, even though a relatively large improvement in both STOI and PESQ are achieved at low SNRs. This is also supported by listening tests, performed by the authors, where the general opinion is that IUTs introduce distortion at high SNRs, which the classical methods do not.

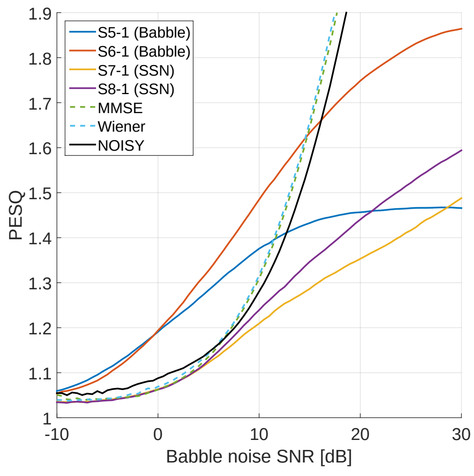
However, it should be noticed that the IUTs using the soft mask always outperform the IUTs using the binary mask as seen from Figs 7.5c and 7.5d. This is also supported by listening tests, performed by the authors, where the binary mask in some cases "cuts-off" minor parts of the signal. Parts, which are better preserved with the soft mask. Especially, at high SNRs the binary mask significantly degrade the intelligibility whereas the soft mask in general preserves it.

From Figs 7.6a and 7.6b, which are the tests where SSN is used, a similar pattern is observed. It is seen that IUT S7-1 and S8-1 outperforms the Wiener filtering method and the MMSE method for SNRs below 5 dB. It is also seen that IUT S8-1, which is the one trained with the highest SNR, has the best overall performance and similar with the former results the soft mask achieves the best overall performance compared to the binary mask, as seen from Figs 7.6c and 7.6d.

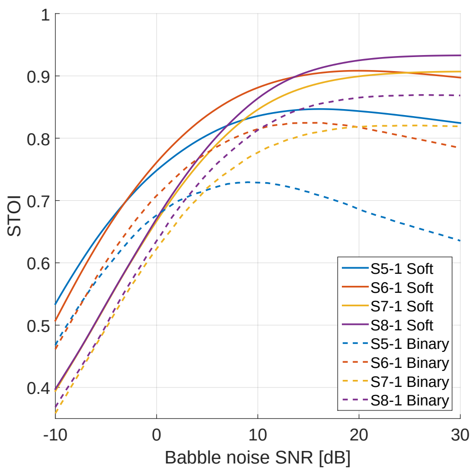
An interesting thing to notice from Figs 7.5a, 7.5b, 7.6a and 7.6b is that the clas-



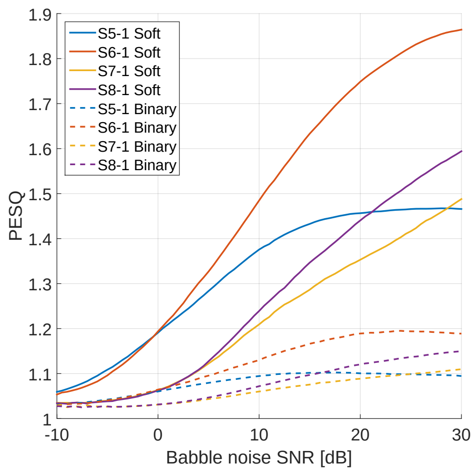
**(a)** A comparison of STOI performance between IUTs and the classical methods.



**(b)** A comparison of PESQ performance between IUTs and the classical methods.

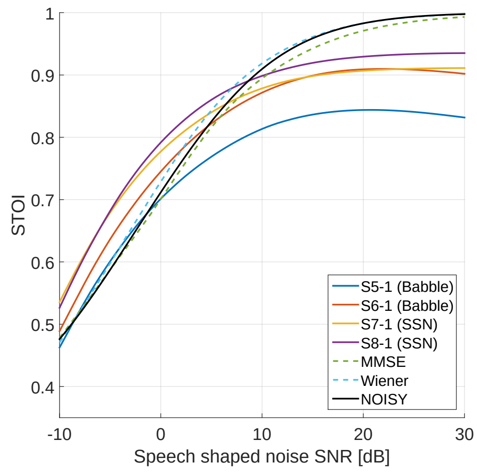


**(c)** A comparison of STOI performance between the soft mask and the binary mask.

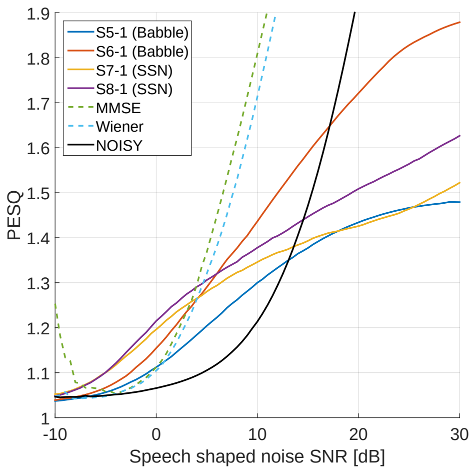


**(d)** A comparison of PESQ performance between the soft mask and the binary mask.

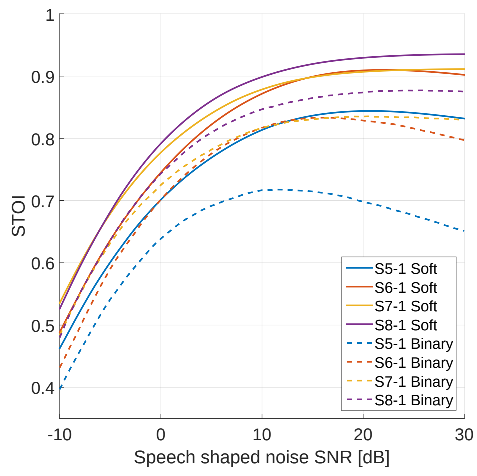
**Figure 7.5:** This figure presents the performance of IUT S5-1 to S8-1 as well as the Wiener filter and MMSE estimator, when tested with babble noise from -10 dB SNR to 30 dB SNR. The performance is quantified by both the STOI and PESQ measure and both the performance of the soft mask as well as the binary mask is presented. It is seen that IUTs S5-1 and S6-1 achieves the highest scores regarding both STOI and PESQ when the SNR is below 10 dB. When the SNR is above 10 dB the classical methods perform the best. It is also seen that the soft mask performs significantly better than the binary mask.



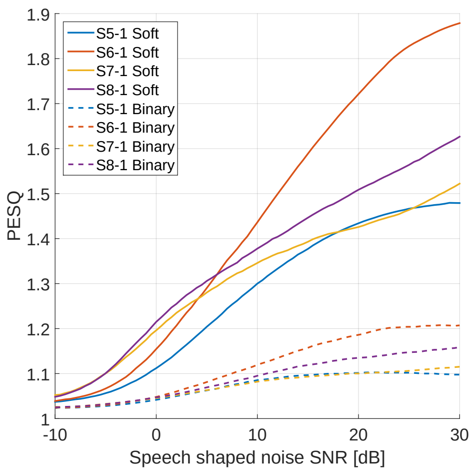
**(a)** A comparison of STOI performance between IUTs and the classical methods.



**(b)** A comparison of PESQ performance between IUTs and the classical methods.



**(c)** A comparison of STOI performance between the soft mask and the binary mask.



**(d)** A comparison of PESQ performance between the soft mask and the binary mask.

**Figure 7.6:** This figure presents the performance of IUT S5-1 to S8-1 as well as the Wiener filter and MMSE estimator, when tested with SSN noise from -10 dB SNR to 30 dB SNR. The performance is quantified by both the STOI and PESQ measure and both the performance of the soft mask as well as the binary mask is presented. It is seen that IUTs S7-1 and S8-1 achieves the highest scores regarding both STOI and PESQ when the SNR is below 5 dB. When the SNR is above 5 dB the classical methods perform the best. It is also seen that the soft mask performs significantly better than the binary mask.

sical methods, especially regarding quality, perform significantly better when they are exposed to SSN as opposed to babble noise. This is also in line with intuition since these classical methods are based on estimates of the noise spectrum. Good estimates are assumed easier to acquire when the noise is stationary (e.g. SSN) opposed to non-stationary (e.g. babble noise). More specifically, the estimation of the noise spectrum in the implementations of the wiener filter and the MMSE, involves a Voice Activity Detection (VAD) algorithm which in general do not work well in non-stationary environment such as with babble noise [Loizou, 2007, p.609].

It is also interesting to see that S6-1, when tested with SSN, improves both intelligibility and quality, as well as outperforming the classical methods at SNRs below 0 dB. This indicates that, even though S6-1 has been trained with babble noise at 0 dB SNR, it might have the potential to generalize to the unseen SSN.

The conclusion based on the tests presented in Figs 7.5 and 7.6 is that the IUTs are relatively robust against variations in SNR and outperforms the Wiener filtering method and the MMSE method for SNRs approximately below 10 dB for babble noise, and 5 dB for SSN noise. When the IUTs are tested with a noise type they have not been trained with, they do not, in general, improve either speech intelligibility or speech quality.

It can hereby be concluded that if the IUTs are tested with the same noise type, they have been trained with, they have the potential to outperform the classical methods at low SNRs. However, since the IUTs only have been trained with low SNRs, nothing argues against that the IUTs can perform even better if they are trained with more SNRs levels. Also, the robustness against different noise types is assumed to be improved if the IUTs are trained with both SSN and babble noise.

## 7.4 System Improvements

This section will present some ideas that might improve the performance of the IUT. The IUT is based on the design from [Healy et al., 2013], which in terms of Deep Learning evolution, is rather young and many improvements have been made since the techniques used in [Healy et al., 2013] were invented. This means that it is likely that a system can be designed which outperforms the current design of the IUT as presented in Chapter 6.

In the current contex, improvements can be categorized into two groups, which are equally important in each their own way. The first group of improvements is related to actual performance improvements in terms of classification accuracy, HIT-FA, STOI, PESQ etc. The other group of performance improvements is related to decreasing the turnaround time, i.e. the time it takes to test and evaluate a current system.

### 7.4.1 Increasing Performance

Regarding the actual performance improvements several techniques can be applied, related to both pre-training and fine-tuning. The straightforward way to improve the performance is to increase the model size and increase the quantity of training data. Also the robustness of the network is expected to improve if the network is trained with different noise types and SNRs .

To set things in perspective, the best performing speech recognition system, in the sense that it has the best speech recognition capabilities in noisy environments, is designed by the Chinese Internet search provider Baidu. The system is based on Deep Learning and consists of a network with over 5 billion trainable parameters and the system is trained on over 100,000 hours of speech material. The training data was generated by mixing the same utterance with many different noise types found on the Internet [Hannun et al., 2014] [The Huffington Post, 2015] [Dr. Andrew Ng, 2015]. In contrast, the system presented in this thesis consists of approximately 7,400,000 trainable parameters and is trained on approximately 20 min. of speech material, so it is believed that scaling up the model and the amount of training data will increase the performance of the system significantly. However, doing so have some practical challenges and leading Deep Learning companies such as Baidu, Facebook and Google, use large High Performance Computing (HPC) clusters for training their models. A way to alleviate these challenges are further discussed in Section 7.4.2.

#### Improving Pre-training

Apart from the obvious ways to improve the system, by using more data, other methods exist for which a performance increase is expected, if they are properly used. Regarding the pre-training stage the very rough approximations done by CD-1 could be made more accurate by running the Gibbs chain in CD for longer time, hence achieving a sample from the RBM distribution, which is statistical more accurate compared to the sample achieved by CD-1 [Tieleman, 2008]. This is particular useful if the mixing rate of the Gibbs chain is very slow. Other more successful alternatives to CD also exist, which are Persistent Contrastive Divergence (PCD) and Parallel Tempering (PT) [Tieleman, 2008] [Desjardins et al., 2010] [Fischer and Igel, 2014].

PCD is different from CD in the way that the Gibbs chain used to generate samples is not initialized from a training sample. Instead, the states of the former Gibbs chains are used; the chains are persistent [Tieleman, 2008].

PT is an extension of PCD where multiple Gibbs chains are used, each with their own beneficial properties such that some have a high mixing rate but rather incorrect equilibrium distribution and some have a low mixing rate but a more accurate equilibrium distribution. Using a technique called cross temperature state swaps, the beneficial properties of these multiple Gibbs chains can be combined into a sample, which are statistical aligned with the RBM distribution of interest [Tieleman, 2008].

### Modifying the Network Architecture

Another approach to improve performance is to redesign the DNNs, which can be related to either the structural architecture or the neuron design. In [Hannun et al., 2014], which as mentioned, is the current state-of-the-art ASR system, the network architecture and neuron design is different from the system presented in Chapter 6. Hereby, it is natural to be inspired from that design, since the speech enhancement task attempted to be solved in this thesis, is very related to the problem attempted to be solved in [Hannun et al., 2014].

The goal in [Hannun et al., 2014] is to make the ASR system robust against noisy speech. Hereby, it is natural to think that if the system in [Hannun et al., 2014] improves state-of-the-art ASR, the techniques could be used to improve state-of-the-art speech enhancement systems, such as the one presented in [Healy et al., 2013] and [Huang et al., 2014b], since their problem domains are similar. That is, retrieving speech information from noisy recordings.

In [Hannun et al., 2014] a Deep Bidirectional Recurrent Neural Network (DBRNN) is used, which is a neural network architecture which incorporates temporal information by using feedback connections. The DBRNN is hereby suitable for sequential data such as speech [Graves et al., 2013]. In [Hannun et al., 2014], this DBRNN replaces both the Gaussian mixture models and the hidden Markov models from traditional state-of-the-art ASR systems. In [Hannun et al., 2014] another neuron design is also used, where the activation functions are based on Rectified Linear units (ReLus), which is a non-linear function being zero from minus infinity to zero and linear from zero to positive infinity. ReLus have been found to improve DNN performance but at the same time decreasing training time [Zeiler et al., 2013] [Maas et al., 2013] [Dahl et al., 2013].

In [Hannun et al., 2014], also a technique known as dropout is utilized, which is a way to force each neuron to be more independent by avoiding co-adaption between neurons and hereby improve generalization [Hinton et al., 2012b] [Dahl et al., 2013]. Dropout is a very simple technique where neurons are randomly "turned off", i.e. set to zero, with a certain probability and in the original paper [Hinton et al., 2012b] a probability of 0.5 was proposed but in [Hannun et al., 2014] a probability between 0.05 - 0.1 was used.

As mentioned in the introduction in Chapter 1, the Deep Learning paradigm is based on models learning their own features, which might result in a completely new belief of what is the best way of pre-processing acoustic signals [Deng et al., 2013]. In [Hannun et al., 2014] spectrograms are used as feature representation and no additional features are computed. This is supported in [Deng et al., 2013] where it is reported that spectrogram features used with DNNs outperform traditional features such as MFCC and PLP. It is hereby likely to believe that the IUT would also benefit from discarding the traditional features and only use a spectrogram representation. As mentioned in Section 6.5.2, 75% of the execution time was spent on computing features, so reducing the number of feature representations to a single spectrogram representation, might not only improve performance but also execution time.

### 7.4.2 Decreasing Turnaround Time

The other group of performance improvements are related to decreasing the turnaround time, which inevitably will increase as larger models and quantities of data is utilized. On a NVIDIA conference in 2015, one of the leading Deep Learning scientists Andrew Ng, mentioned two equally important aspect of making deep learning a successful tool. He referred to these, as week and strong scaling [Dr. Andrew Ng, 2015]. Weak scaling is making larger neural networks and strong scaling is decreasing the time it takes to train them. This may also explain why the majority of leading Deep Learning scientists have been headhunted by the leading computing companies in the world [Wired, 2013], [Baidu, 2014], [Facebook, 2013]. Baidu recently announced that they intent to build a new supercomputer for Deep Learning, which will rank top 10 over the most powerful computers in the world, so computing power is obvious essential in Deep Learning [The Wall Street Journal, 2015].

As mentioned in Section 6.5.3, DNNs are inherently parallel and hereby ideal for hardware architectures such as Graphics Processor Units (GPUs), which are optimized for simple parallel computations. The trend in Deep Learning is also pointing in the direction of using GPUs instead of CPUs for training deep architectures [NVIDIA, 2015] [The Platform, 2015] [Hannun et al., 2014]. NVIDIA, which is one of the world largest manufactures of GPUs, has recently announced that their next GPU card is designed with Deep Learning in mind and will be optimized towards fixed point precision [NVIDIA, 2015]. However, utilizing a large amount of processing units in a GPU, or a CPU, is not trivial and require special designed training algorithms in order to ensure that the hardware is fully utilized. In general, fully connected DNNs, like the ones used in this thesis, are bandwidth limited and non-fully connected networks such as CNNs are computation limited [NVIDIA, 2015]. This means that implementing the current IUT on a GPU might not generate a speed up since too large amounts of data are needed to be transfered between primary memory and GPU memory. The authors of this thesis have tried using the build-in functionality in MATLAB for GPU programming but without achieving any speed-up, which might be due to memory bandwidth limitations. Achieving a speed-up, if even possible, might require GPU programming on a low abstraction level.

One approach which might alleviate the problem of being bandwidth limited is to use a numerical representation with less precision. In the IUT 64 bit floating points is used for all computations and the effective memory bandwidth will be increased by a factor of two if 32 bit floating points were used instead of 64 bits. Google uses 32 bit floating point for their DNN implementations and when they communicate parameters between servers, they even cut off 16 bit of the mantissa to reduce the memory consumption and increase the effective memory bandwidth [Jeff Dean, 2015]. In [Gupta et al., 2015] it is even shown that using a 16 bit fixed point number representation opposed to 32 bit floating point, only a negligible error is introduced. Google has also developed a version of gradient descent which is optimized towards large scale distributed computing which enables DNNs to be trained on HPC clusters including thousands of computers [Dean et al., 2012].

# Chapter 8

# Conclusion

This master's thesis has investigated how speech enhancement techniques from the CASA field can be combined with the emerging machine learning field Deep Learning. More specifically the scope of the thesis was to investigate the speech enhancement capabilities of the method described in [Healy et al., 2013] with a special focus on five objectives, which are reposted below for convenience. Based on the results and discussions in Chapter 7, a concluding remark for each of the five objectives will be given in the following paragraphs.

## Objectives

1. Does unsupervised pre-training of DNNs, when used for IBM estimation, improve performance?
2. What is the effect of using looped noise, as opposed to non-looped noise, regarding DNN classification performance?
3. Does using a soft decision rather than a hard decision, in a DNN based speech enhancement system, improve speech intelligibility and/or quality?
4. How well does DNNs used for IBM estimation generalize to unseen noise types and SNRs?
5. How do DNN based speech enhancement algorithms perform relative to the classical STFT based Wiener filter and the spectral magnitude MMSE estimator?

## Concluding Remark Objective 1

Regarding the benefit of pre-training it was found that pre-training under some circumstances have a beneficial effect. When pre-training was beneficial it was observed that the DNNs achieved a higher classification accuracy than the DNNs trained purely with backpropagation. It was also found that in all cases, where pre-training was beneficial, the lowest validation error was found considerably faster for the pre-trained DNNs than for the non pre-trained DNNs (Tab. 6.3). Likewise, enabling pre-training to be beneficial required that the amount of unlabeled data was considerably larger than the amount of labeled data. The findings regarding pre-training are in line with the literature, where it is reported that achieving a beneficial effect of pre-training requires a significant amount of experience and specialized skills of the researcher [Deng et al., 2013].

## Concluding Remark Objective 2

In [Healy et al., 2013] the same noise realization was used for both training and testing of the DNN based speech enhancement method, which in [May and Dau, 2014] was claimed led to performance, which is non-realizable in practice. These claims led to objective 2. It has hereby been investigated how DNNs perform when they are trained and tested using the same noise realization (looped noise) as opposed to the case where the DNNs are trained and tested with unique and different noise realizations (non-looped noise). The results from the DNN based speech enhancement system presented in this thesis support the findings in [May and Dau, 2014] and the performance of the system increases significantly when looped noise is used. However, even though the performance decreases when non-looped noise is used, consistent improvements in STOI from unprocessed to processed speech is achieved.

## Concluding Remark Objective 3

In the speech recognition literature is has been found beneficial to use a soft decision (soft mask) instead of a hard decision (binary mask) as part of a speech enhancement stage. This has also been shown beneficial for classical speech enhancement methods [Jensen and Hendriks, 2012]. This led to the hypothesis that using a soft mask in a DNN based speech enhancement algorithm, aimed at increasing speech intelligibility, would achieve better performance than using a binary mask. To ascertain if this hypothesis is true, formal human listening tests must be conducted. Since this is a time consuming and expensive task, speech intelligibility (STOI) and speech quality (PESQ) predictors have been used [Taal et al., 2010] [Rix et al., 2001]. Based on the results presented in this thesis the following can hereby concluded:

- The soft mask outperforms the binary mask in terms of STOI and PESQ, indicating that larger improvements in speech intelligibility and quality can be achieved by using a soft mask compared to a binary mask.
- Based on informal listening tests, as well as STOI and PESQ, it is found that the soft mask, especially at SNR levels above 0 dB, introduces less distortion than the binary mask.

To the authors knowledge, these results have not earlier been reported in the literature!

### Concluding Remark Objective 4

If a speech enhancement system is to be used in a real life scenario it is exposed to a large variety of noise types and SNRs. It is therefore of interest to identify how robust the DNN based speech enhancement system presented in this thesis is to unseen noise types and SNRs. When the speech enhancement system is trained and tested with the same noise type but with a varying SNR, it is found that the system can improve both STOI and PESQ for a processed speech signal. This applies in situations where the unprocessed speech signal has a SNR up to approximately 10 dB larger than what the system was trained to. This holds for both babble noise and SSN. When the speech enhancement system is trained with babble noise at -5 dB SNR and tested with SSN, it is found that the system can improve PESQ at SNRs up to approximately 12 dB. This is increased to approximately 17 dB if the system is trained with babble noise at 0 dB SNR. It is also found that a system trained with babble noise at 0 dB SNR can improve STOI when it is tested with speech signals corrupted by SSN, in the SNR interval from approximately -10 dB to 0 dB.

### Concluding Remark Objective 5

To proper evaluate the DNN based speech enhancement system presented in this thesis a comparison, to two classical methods, have been performed. These methods are the Wiener filter and the spectral magnitude MMSE estimator. It has been found that the DNN based speech enhancement system, when trained and tested with the same noise type, outperforms both the Wiener filtering method and the spectral magnitude MMSE estimator, regarding both STOI and PESQ, for both babble noise and SSN from -5 dB to 5 dB SNR (Figs. 7.5 and 7.6). However, it is found that the DNN based speech enhancement system introduces a distortion at SNRs above 15 dB but this distortion is assumed to be reduced if the system is trained at these SNR levels.

### Final Remark

What motivated the work leading to this master's thesis was the question if the results reported in [Healy et al., 2013] are achievable in real life situations where noise type, SNR, etc. is not known accurately. The conclusion is that these results are *not* realistic and they are artificially high due to the use of the *same* noise sequence during training and testing, i.e. in looped noise conditions. However, it has been found that a speech enhancement algorithm similar to the one presented in [Healy et al., 2013] presumably can improve speech intelligibility and quality in realistic scenarios. Before these results can be ascertained, human listening tests must be conducted to identify if the proposed algorithm in fact can improve intelligibility and quality.

Currently the design of the DNN based speech enhancement system presented in this thesis is not mature and is not considered practical applicable due to the quality of the enhanced speech, which at SNR levels at -5 dB is still considered too low to be used in real life scenarios.

However, at SNR levels from 0 dB and up to approximately 15 dB, the quality is considered good and the noise is considerably attenuated. This is even though the systems have not been trained at these SNR levels and the performance is hereby assumed to be improved, if such SNR levels are included in the training.

One especially significant finding, presented in this thesis, is that a soft mask approach outperforms the binary mask approach as the one used in [Healy et al., 2013], [Kim et al., 2009] and [Hang and Wang, 2012]. To the authors knowledge, such findings have not earlier been reported.

As the final remark, the results achieved in this thesis indicate that DNN based speech enhancement systems have a great potential and it is believed by the authors that if a sufficient amount of training data and a sufficient amount of computing power is available, DNN based speech enhancement systems can be practical applicable and will be able to successfully assist people by enhancing noisy speech in real life applications.

# Bibliography

Bach, P. and Kercel, S. W. (2003). Sensory substitution and the human–machine interface. *TRENDS in Cognitive Sciences*, 7(12):541–546.

Baidu (2014). Baidu Opens Silicon Valley Lab, Appoints Andrew Ng as Head of Baidu Research. http://ir.baidu.com/phoenix.zhtml?c=188488&p=irol-newsArticle&ID=1931950.

Barker, J., Josifovski, L., Cooke, M., and Green, P. (2000). Soft decisions in missing data techniques for robust automatic speech recognition. In *Proc. ICSLP-2000*, pages 373–376.

BBC Newsnight (2014). Is a.i. the problem or the solution? https://www.youtube.com/watch?v=lge-dl2JUAM.

Bengio, Y. (2009). Learning deep architectures for ai. *Found. Trends Mach. Learn.*, 2(1):1–127.

Bengio, Y., Goodfellow, I. J., and Courville, A. (2015). Deep learning. Book in preparation for MIT Press.

Bengio, Y., Lamblin, P., Popovici, D., and Larochelle, H. (2007). Greedy layer-wise training of deep networks. In *In NIPS*. MIT Press.

Bengtsson, S. and Røgeskov, M. (2010). *Personer Med Høretab I Danmark*. Videnscenter for Hørehandicap i samarbejde med SFI Det Nationale Forskningscenter For Velfærd.

Bishop, C. M. (1995). *Neural Networks for Pattern Recognition*. CLARENDON PRESS OXFORD., 1. edition.

Bishop, C. M. (8$^{\text{th}}$ printing, 2009). *Pattern Recognition and Machine Learning*. Springer., 1. edition.

Blauert, J. and Laws, P. (1978). Group delay distortions in electroacoustical systems. *The Journal of the Acoustical Society of America*, 63(5).

Brügge, K., Fischer, A., and Igel, C. (2013). The flip-the-state transition operator for restricted boltzmann machines. *Machine Learning*, 93(1):53–69.

Chen, J., Benesty, J., Huang, Y. A., and Doclol, S. (2006). New insights into the noise reduction wiener filter. *IEEE Transactions on Audio, Speech and Language Processing*, 14(2):1218–1234.

Chen, X. and Lin, X. (2014). Big data deep learning: Challenges and perspectives. *Access, IEEE*, 2:514–525.

Cherry, E. C. (1953). Some experiments on the recognition of speech, with one and with two ears. *The Journal of the Acoustical Society of America*, 25(5).

Cho, K., Ilin, A., and Raiko, T. (2011). Improved learning of gaussian-bernoulli restricted boltzmann machines. In *Proceedings of the 21th International Conference on Artificial Neural Networks - Volume Part I*, ICANN'11, pages 10–17, Berlin, Heidelberg. Springer-Verlag.

Christensen, V. T. (2006). *UHØRT - Betydningen Af Nedsat Hørelse For Arbejdsmarkedstilknytning og Arbejdsliv.* SFI Det Nationale Forskningscenter For Velfærd.

CNN Money (2014). Elon musk warns against unleashing artificial intelligence 'demon'. `http://money.cnn.com/2014/10/26/technology/elon-musk-artificial-intelligence-demon/`.

Cognimem (2014). Cm1k chip. `http://www.cognimem.com/products/chips-and-modules/CM1K-Chip/index.html`.

Dahl, G. E., Sainath, T. N., and Hinton, G. E. (2013). Improving deep neural networks for LVCSR using rectified linear units and dropout. In *IEEE International Conference on Acoustics, Speech and Signal Processing, ICASSP 2013, Vancouver, BC, Canada, May 26-31, 2013*, pages 8609–8613.

Danmarks Statistik (2014). Folketal i danmark 1. januar 2015. `http://www.dst.dk/da/Statistik/emner/befolkning-og-befolkningsfremskrivning/folketal.aspx`.

Dean, J., Corrado, G., Monga, R., Chen, K., Devin, M., Le, Q. V., Mao, M. Z., Ranzato, M., Senior, A. W., Tucker, P. A., Yang, K., and Ng, A. Y. (2012). Large scale distributed deep networks. In Bartlett, P. L., Pereira, F. C. N., Burges, C. J. C., Bottou, L., and Weinberger, K. Q., editors, *NIPS*, pages 1232–1240.

Deller, J. R., Hansen, J. H., and Proakis, J. G. (1993). *Discrete-Time Processing of Speech Signals.* Wiley Interscience, 1. edition.

Deng, L., Hinton, G., and Kingsbury, B. (2013). New types of deep neural network learning for speech recognition and related applications: An overview. In *in Proc. Int. Conf. Acoust., Speech, Signal Process.*

Deng, L. and Yu, D. (2014). Deep learning: Methods and applications. Technical Report MSR-TR-2014-21.

Desjardins, G., Courville, A., Bengio, Y., Vincent, P., and Delalleau, O. (2010). Tempered Markov Chain Monte Carlo for training of restricted Boltzmann machines.

In Teh, Y. W. and Titterington, M., editors, *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics, May 13-15, 2010, Chia Laguna Resort, Sardinia, Italy*, pages 145–152.

Dieleman, S. and Schrauwen, B. (2012). Accelerating sparse restricted boltzmann machine training using non-gaussianity measures. In Bengio, Y., Bergstra, J., and Le, Q., editors, *Deep Learning and Unsupervised Feature Learning, Proceedings*, page 9.

Dr. Andrew Ng (2015). Gpu technology conference 2015. http://www.ustream.tv/recorded/60113824.

Duda, R. O., Hart, P. E., and Stork, D. G. (2001). *Pattern Classification.* John Wiley & Sons., 2. edition.

Ellis, D. P. W. (2005). PLP and RASTA (and MFCC, and inversion) in Matlab. http://www.ee.columbia.edu/~dpwe/resources/matlab/rastamat/.

Facebook (2013). Yann lecun director of research lab at facebook. https://www.facebook.com/yann.lecun/posts/10151728212367143.

Fawcett, T. (2006). Roc graphs with instance-varying costs. *Pattern Recogn. Lett.*, 27(8):882–891.

Fischer, A. and Igel, C. (2012). *An Introduction to Restricted Boltzmann Machines*, pages 14–36. Lecture Notes in Computer Science. Springer.

Fischer, A. and Igel, C. (2014). Training restricted boltzmann machines: an introduction. *Pattern Recognition*, 47(1):25–39.

Freeman, W. J. (1975). *Mass Action in the Nervous System.* Academic Press, 1. edition.

Future of Life Institute (2015). Elon musk donates $10m to keep ai beneficial. http://futureoflife.org/misc/AI.

Garofolo, J., Lamel, L., Fisher, W., Fiscus, J., Pallett, D., Dahlgren, N., and Zue, V. (1993). Timit acoustic-phonetic continuous speech corpus. philadelphia: Linguistic data consortium.

Golumbic, E. M. Z., Ding, N., Bickel, S., Lakatos, P., Schevon, C. A., McKhann, G. M., Goodman, R. R., Emerson, R., Mehta, A. D., Simon, J. Z., Poeppel, D., and Schroeder, C. E. (2013). Mechanisms underlying selective neuronal tracking of attended speech at a cocktail party. *Neuron*, 77(5):980 – 991.

Gougoux, F., Belin, P., Voss, P., Lepore, F., Lassonde, M., and Zatorre., R. J. (2009). Voice perception in blind persons: A functional magnetic resonance imaging study. *Neuropsychologia*, 47(13):2967–2974.

Graves, A., Mohamed, A., and Hinton, G. E. (2013). Speech recognition with deep recurrent neural networks. *CoRR*, abs/1303.5778.

Gupta, S., Agrawal, A., Gopalakrishnan, K., and Narayanan, P. (2015). Deep learning with limited numerical precision. *CoRR*, abs/1502.02551.

Guyton and Hall (2011). *Textbook of Medical Physiology.* SAUNDERS ELSEVIER, 12. edition.

Hang, K. and Wang, D. (2012). A classification based approach to speech segregation. *Acoustical Society of America*, 28:3475–3483.

Hannun, A. Y., Case, C., Casper, J., Catanzaro, B. C., Diamos, G., Elsen, E., Prenger, R., Satheesh, S., Sengupta, S., Coates, A., and Ng, A. Y. (2014). Deep speech: Scaling up end-to-end speech recognition. *CoRR*, abs/1412.5567.

Haykin, S. (1999). *Neural Networks - A Comprehensive Foundation.* Prentice Hall International Inc., 2. edition.

He, K., Zhang, X., Ren, S., and Sun, J. (2015). Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. *CoRR*, abs/1502.01852.

Healy, E. W., Yoho, S. E., Wang, Y., and Wang, D. (2013). An algorithm to improve speech recognition in noise for hearing-impaired listeners. *Acoustical Society of America*, 134(4):3029–3038.

Hendriks, R. C., Gerkmann, T., and Jensen, J. (2013). Dft-domain based single-microphone noise reduction for speech enhancement - a survey of the state-of-the-art. *Synthesis Lecture on Speech Processing and Audio Processing*, pages 1–62.

Hermansky, H. (1989). Perceptual linear predictive (plp) analysis of speech. *Acoustical Society of America 87(4) 1900*, pages 1738–1752.

Hermansky, H. (1994). Rasta processing of speech. *IEEE Transactions on Speech and Audio Processing, Volume:2, Issue: 4*, pages 578–589.

Hinton, G. E. (2002). Training products of experts by minimizing contrastive divergence. *Neural Comput.*, 14(8):1771–1800.

Hinton, G. E. (2012). A practical guide to training restricted boltzmann machines. In *Neural Networks: Tricks of the Trade - Second Edition*, pages 599–619.

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

Hinton, G. E., Osindero, S., and Teh, Y.-W. (2006). A fast learning algorithm for deep belief nets. *Neural Comput.*, 18(7):1527–1554.

Hinton, G. E. and Salakhutdinov, R. R. (2006). Reducing the dimensionality of data with neural networks. *Science*, 313(5786):504–507.

Hinton, G. E., Srivastava, N., Krizhevsky, A., Sutskever, I., and Salakhutdinov, R. (2012b). Improving neural networks by preventing co-adaptation of feature detectors. *CoRR*, abs/1207.0580.

Huang, P.-S., Kim, M., Hasegawa-Johnson, M., and Smaragdis, P. (2014a). Deep learning for monaural speech separation. In *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 1562–1566.

Huang, P.-S., Kim, M., Hasegawa-Johnson, M., and Smaragdis, P. (2014b). Deep learning for monaural speech separation. In *Acoustics, Speech and Signal Processing (ICASSP), 2014 IEEE International Conference on*, pages 1562–1566.

Huang, X., Acero, A., and Hon, H.-W. (2001). *Spoken Language Processing - A Guide to Theory, Algorithm and System Development.* Prentice Hall, 1. edition.

IBM Research (2014a). Brain power. `http://www.research.ibm.com/cognitive-computing/neurosynaptic-chips.shtml#fbid=CkowsRIXNT6`.

IBM Research (2014b). Introducing a brain-inspired computer. `http://www.research.ibm.com/articles/brain-chip.shtml`.

IBM Research (2015). How deep blue works. `https://www.research.ibm.com/deepblue/meet/html/d.3.2.html`.

IDSIA (2014). Deep learning successes obtained by idsia. `http://deeplearning.net/2013/10/08/deep-learning-successes-obtained-by-idsia/`.

IEEE Spectrum (2015). Facebook ai director yann lecun on his quest to unleash deep learning and make machines smarter. `http://spectrum.ieee.org/automaton/robotics/artificial-intelligence/facebook-ai-director-yann-lecun-on-deep-learning#qaTopicEight`.

Jeff Dean (2015). Gpu technology conference 2015. `http://www.ustream.tv/channel/gpu-technology-conference-2015`.

Jensen, J. and Hendriks, R. C. (2012). Spectral magnitude minimum mean-square error estimation using binary and continuous gain functions. *IEEE Trans. Audio, Speech, Language Process, Volume:20, No: 1*, pages 92–102.

Jensen, J. and Tan, Z.-H. (2015). Minimum mean-square error estimation of mel-frequency cepstral features - a theoretically consistent approach. *IEEE/AMC Transactions on Audio, Speech and Language Processing, Volume:23, No: 1*, pages 186–197.

Jin, Z. and Wang, D. (2007a). gammatone.m. `http://web.cse.ohio-state.edu/pnl/shareware/cochleagram`.

Jin, Z. and Wang, D. (2007b). synthesis.m. `http://web.cse.ohio-state.edu/pnl/shareware/cochleagram`.

Kaggle (2012). Merck molecular activity challenge. `https://www.kaggle.com/c/MerckActivity/details/winners`.

Kates, J. M. and Arehart, K. H. (2005). Multichannel dynamic-range compression using digital frequency warping. *EURASIP J. Adv. Sig. Proc.*, 2005(18):3003–3014.

Kay, S. M. (2006). *Intuitive Probability and Random Processes using MATLAB*. Springer.

Khayam, S. A. (2003). The discrete cosine transform - theory and application. `http://www.lokminglui.com/DCT_TR802.pdf`.

Kim, G., Lu, Y., Hu, Y., and Loizoua, P. C. (2009). An algorithm that improves speech intelligibility in noise for normal hearing listeners. *Acoustical Society of America.*, pages 1486–1494.

Kollmeier, B. and Koch, R. (1993). Speech enhancement based on physiological and psychoacoustica models of modulation perception and binaural interaction. *Acoustical Society of America 95, 1593 (1994)*, pages 1593–1602.

Krause, O., Fischer, A., Glasmachers, T., and Igel, C. (2013). Approximation properties of dbns with binary hidden units and real-valued visible units. *JMLR: Workshop and Conference Proceedings*, 28:419–426.

Krizhevsky, A. (2009). Learning multiple layers of features from tiny images. Technical report, Department of Computer Science, University of Toronto, Canada.

Kurzweil, R. (2006). *The Singularity Is Near: When Humans Transcend Biology*. Penguin (Non-Classics).

Larochelle, H., Bengio, Y., Louradour, J., and Lamblin, P. (2009). Exploring strategies for training deep neural networks. *J. Mach. Learn. Res.*, 10:1–40.

Le, Q., Ranzato, M., Monga, R., Devin, M., Chen, K., Corrado, G., Dean, J., and Ng, A. (2012). Building high-level features using large scale unsupervised learning. In *International Conference in Machine Learning*.

Le, Q. V., Ngiam, J., Coates, A., Lahiri, A., Prochnow, B., and Ng, A. Y. (2011a). On optimization methods for deep learning. In Getoor, L. and Scheffer, T., editors, *ICML*, pages 265–272. Omnipress.

Le, Q. V., Zou, W. Y., Yeung, S. Y., and Ng, A. Y. (2011b). Learning hierarchical invariant spatio-temporal features for action recognition with independent subspace analysis. In *Proceedings of the 2011 IEEE Conference on Computer Vision and Pattern Recognition*, CVPR '11, pages 3361–3368. IEEE Computer Society.

LISA lab, U. o. M. (2014). Deep learning tutorial. http://deeplearning.net/tutorial/deeplearning.pdf.

Loizou, P. C. (2007). *Speech Enhancement -Theory and Practice*. Taylor and Francis, 1. edition.

Loizou, P. C. (2013). *Speech Enhancement -Theory and Practice*. Taylor and Francis, 2. edition.

Maas, A., Le, Q., O'Neil, T., Vinyals, O., Nguyen, P., and Ng, A. (2012). Recurrent neural networks for noise reduction in robust asr. In *Proceedings of INTERSPEECH*.

Maas, A. L., Hannun, A. Y., and Ng, A. Y. (2013). Rectifier nonlinearities improve neural network acoustic models. International Conference on Machine Learning.

Makhoul, J. (1975). Linear prediction: A tutorial review. *Proceedings of the IEEE (Volume:63 Issue: 4 )*, pages 561–580.

May, T. and Dau, T. (2014). Requirements for the evaluation of computational speech segregation systems. *Acoustical Society of America 136(6)*, pages 398–404.

McDermott, J. H. (2009). The cocktail party problem. *Current Biology*, 19(22):R1024 – R1027.

Melchior, J. (2012). Learning natural image statistics with gaussian-binary restricted boltzmann machine. Technical report, Institute for neural computation, Ruhr University Bochum, Germany.

Mike Brookes, I. C. L. (1997). Voicebox. http://www.ee.ic.ac.uk/hp/staff/dmb/voicebox/voicebox.html.

Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., Graves, A., Riedmiller, M., Fidjeland, A. K., Ostrovski, G., Petersen, S., Beattie, C., Sadik, A., Antonoglou, I., King, H., Kumaran, D., Wierstra, D., Legg, S., and Hassabis, D. (2015). Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533.

Mohamed, A., Dahl, G., and Hinton, G. (2012). Acoustic modeling using deep belief networks. *Audio, Speech, and Language Processing, IEEE Transactions on*, 20(1):14 –22.

Montavon, G., Orr, G. B., and Müller, K.-R., editors (2012). *Neural Networks: Tricks of the Trade*. Springer Berlin Heidelberg.

Moos, T. and Møller, M. (2010). *Basal Neuro Anatomi*. FADL's Forlag, 3. edition.

Murphy, K. P. (2012). *Machine Learning: A Probabilistic Perspective*. The MIT Press.

Narayanan, A. and Wang, D. (2013). Ideal ratio mask estimation using deep neural networks for robust speech recognition. In *Acoustics, Speech and Signal Processing (ICASSP), 2013 IEEE International Conference on*, pages 7092–7096.

Neal, R. M. (1992). Connectionist learning of belief networks. *Artif. Intell.*, 56(1):71–113.

New York Times (2012). Scientists see promise in deep-learning programs. `http://www.nytimes.com/2012/11/24/science/scientists-see-advances-in-deep-learning-a-part-of-artificial-intelligence.html?_r=1`.

New Yorker (2013). Hyping artificial intelligence, yet again. `http://www.newyorker.com/tech/elements/hyping-artificial-intelligence-yet-again`.

Nilsson, M., Soli, S. D., and Sullivan, J. A. (1994). Development of the hearing in noise test for the measurement of speech reception thresholds in quiet and in noise. *The Journal of the Acoustical Society of America*, 95(2).

NVIDIA (2015). Gpu technology conference 2015 - leaps in visual computing. `http://www.ustream.tv/channel/gpu-technology-conference-2015`.

Pearce, D., günter Hirsch, H., and Gmbh, E. E. D. (2000). The aurora experimental framework for the performance evaluation of speech recognition systems under noisy conditions. In *in ISCA ITRW ASR2000*, pages 29–32.

Raj, B. and Stern, R. (2005). Missing-feature approaches in speech recognition. *Signal Processing Magazine, IEEE*, 22(5):101–116.

Rix, A., Beerends, J., Hollier, M., and Hekstra, A. (2001). Perceptual evaluation of speech quality (pesq)-a new method for speech quality assessment of telephone networks and codecs. In *Acoustics, Speech, and Signal Processing, 2001. Proceedings. (ICASSP '01). 2001 IEEE International Conference on*, volume 2, pages 749–752 vol.2.

Rothauser, E. H., Chapman, W. D., Guttman, N., Hecker, M. H. L., Nordby, K. S., Silbiger, H. R., Urbanek, G. E., and Weinstock, M. (1969). IEEE recommended practice for speech quality measurements. *IEEE Transactions on Audio and Electroacoustics*, 17:225–246.

Rumelhart, D. E., Hinton, G. E., and Williams, R. J. (1986). Learning representations by back-propagating errors. *Nature*, 323(9):533–536.

Salakhutdinov, R. and Hinton, G. (2006). Training a deep autoencoder or a classifier on mnist digits. `http://www.cs.toronto.edu/~hinton/MatlabForSciencePaper.html`.

Simard, P. Y., Steinkraus, D., and Platt, J. C. (2003). Best practices for convolutional neural networks applied to visual document analysis. In *International Conference on Document Analysis and Recognition*, volume 2, pages 958–962.

Simpson, A. J., Roma, G., and Plumbley, M. D. (2015). Deep karaoke: Extracting vocals from musical mixtures using a convolutional deep neural network. Book in preparation for MIT Press.

Simpson, A. J. R. (2015). Deep transform: Cocktail party source separation via probabilistic re-synthesis. *CoRR*, abs/1503.06046.

Singh, S., Tripathy, M., and Anand, R. (2014). A fuzzy mask based on wavelet packet for improving speech quality and intelligibility. In *Signal Processing and Integrated Networks (SPIN), 2014 International Conference on*, pages 1–4.

Stanford Encyclopedia (2014). Connectionism. http://plato.stanford.edu/entries/connectionism/.

Taal, C., Hendriks, R., Heusdens, R., and Jensen, J. (2010). A short-time objective intelligibility measure for time-frequency weighted noisy speech. In *Acoustics Speech and Signal Processing (ICASSP), 2010 IEEE International Conference on*, pages 4214–4217.

Taal, C. H. (2010). Stoi - short-time objective intelligibility measure. http://www.ceestaal.nl/matlab.html.

Taigman, Y., Yang, M., Ranzato, M., and Wolf, L. (2014). Deepface: Closing the gap to human-level performance in face verification.

Tchorz, J. and Kollmeier, B. (2002). Estimation of the signal-to-noise ratio with amplitude modulation spectrograms. *Speech Communication 38*, pages 1–17.

Tchorz, J. and Kollmeier, B. (2003). Snr estimation based on amplitude modulation analysis with applications to noise suppression. *IEEE Transactions on Speech and Audio Processing, Volume:11 , Issue: 3*, pages 184 – 192.

The Guardian (2014). Google buys uk artificial intelligence startup deepmind for $650m. http://www.theguardian.com/technology/2014/jan/27/google-acquires-uk-artificial-intelligence-startup-deepmind.

The Huffington Post (2015). Inside the mind that built google brain: On life, creativity, and failure. http://www.huffingtonpost.com/2015/05/13/andrew-ng_n_7267682.html.

The Platform (2015). Deep learning pioneer pushing gpu neural network limits. http://www.theplatform.net/2015/05/11/deep-learning-pioneer-pushing-gpu-neural-network-limits/.

The Register (2015). Ai guru ng: Fearing a rise of killer robots is like worrying about overpopulation on mars. http://www.theregister.co.uk/2015/03/19/andrew_ng_baidu_ai/.

The Wall Street Journal (2015). Baidu leads in artificial intelligence benchmark. http://blogs.wsj.com/digits/2015/05/12/baidu-leads-in-artificial-intelligence-benchmark/.

Tieleman, T. (2008). Training restricted boltzmann machines using approximations to the likelihood gradient. In *Proceedings of the 25th international conference on Machine learning*, pages 1064–1071.

Töscher, A., Jahrer, M., and Bell, R. M. (2009). The bigchaos solution to the netflix grand prize.

Wang, D. (2005). On ideal binary mask as the computational goal of auditory scene analysis. In Divenyi, P., editor, *Speech Separation by Humans and Machines*, pages 181–197. Springer US.

Wang, D. (2008). Time-frequency masking for speech separation and its potential for hearing aid design.

Wang, D. and Brown, G. J. (2006). *Computational Auditory Scene Analysis*. Wiley and Sons.

Wang, Y. (2013). Intelligibility test data from hi and nh listeners. http://web.cse.ohio-state.edu/~dwang/pnl/corpus/Healy-jasa13/YWang.html.

Wang, Y., Han, K., and Wang, D. (2012). Exploring monaural features for classification-based speech segregation.

Wang, Y. and Wang, D. (2013). Towards scaling up classification-based speech separation. *Audio, Speech, and Language Processing, IEEE Transactions on*, 21(7):1381–1390.

Wired (2013). Google hires brains that helped supercharge machine learning. http://www.wired.com/2013/03/google_hinton/.

Wired (2015). How ray kurzweil will help google make the ultimate ai brain. http://www.wired.com/2013/04/kurzweil-google-ai/.

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

Zeiler, M. D., Ranzato, M., Monga, R., Mao, M. Z., Yang, K., Le, Q. V., Nguyen, P., Senior, A. W., Vanhoucke, V., Dean, J., and Hinton, G. E. (2013). On rectified linear units for speech processing. In *ICASSP*, pages 3517–3521. IEEE.

# Appendix A

# Derivation of the Wiener Filter

This appendix describes the derivation of the Wiener filter from Section 3.1.2. The signals from the model in Fig. 3.2 is recalled as

- $\mathbf{h} = [h_0 \;\; h_1 \dots h_{M-1}]^T$ is the filter coefficients.
- $\mathbf{y}(n) = [y(n) \;\; y(n-1) \dots y(n-M-1)]^T$ is the WSS input signals.
- $\hat{d}(n) = \mathbf{y}(n)^T\mathbf{h}$ is the output signal.
- $d(n)$ is a zero mean WSS desired signal.
- $e(n) = d(n) - \hat{d}(n)$ is the error signal.

As mentioned in Section 3.1.2, the aim is to make the filter optimal in a mean-squared error sense and since it is desired to minimize the error signal $e(n)$, a cost function can be formulated as Eq. (A.1) and rewritten as Eq. (A.2).

$$J(\mathbf{w}) = \mathbb{E}[e(n)^2]. \tag{A.1}$$

$$\begin{aligned} J(\mathbf{h}) &= \mathbb{E}[e(n)^2] = \mathbb{E}[(d(n) - \mathbf{y}^T(n)\mathbf{h})^2] = \mathbb{E}[(d(n) - \mathbf{y}^T(n)\mathbf{h})^T(d(n) - \mathbf{y}^T(n)\mathbf{h})] \\ &= \mathbb{E}[d(n)^2] + \mathbb{E}[\mathbf{h}^T\mathbf{y}(n)\mathbf{u}^T(n)\mathbf{h}] - \mathbb{E}[d(n)\mathbf{y}^T(n)\mathbf{h}] - \mathbb{E}[\mathbf{h}^T\mathbf{y}(n)\mathbf{y}^T(n)d(n)] \\ &= \mathbb{E}[d(n)^2] + \mathbf{h}^T\mathbb{E}[\mathbf{y}(n)\mathbf{y}^T(n)]\mathbf{h} - 2\mathbf{h}^T\mathbb{E}[\mathbf{y}(n)d(n)] \\ &= \sigma_d^2 + \mathbf{h}^T\mathbf{R}_{yy}\mathbf{h} - 2\mathbf{h}^T\mathbf{r}_{yd} \end{aligned} \tag{A.2}$$

where $\mathbf{R}_{yy}$ is the auto-correlation matrix of $\mathbf{y}(n)$ and $\mathbf{r}_{yd}$ is the cross-correlation vector between $\mathbf{y}(n)$ and $d(n)$. The gradient of Eq. (A.2) is found as

$$\nabla J(\mathbf{h}) = 2\mathbf{R}_{yy}\mathbf{h} - 2\mathbf{r}_{yd}. \tag{A.3}$$

An optimum is desired such $\nabla J(\mathbf{h}) = 2\mathbf{R}_{yy}\mathbf{h} - 2\mathbf{r}_{yd} = 0$ and the so-called Wiener Hopf equations is hereby given as

$$\mathbf{R}_{yy}\mathbf{h} = \mathbf{r}_{yd}. \tag{A.4}$$

Assuming that $\mathbf{R}_{yy}$ is invertible the filter coefficient can be expressed as

$$\mathbf{h} = \mathbf{R}_{yy}^{-1}\mathbf{r}_{yd}. \tag{A.5}$$

If $\mathbf{R}_{yy}$ is positive definite then Eq. (A.5) will yield the optimal filter coefficients.

# Appendix B

# GMM, Latent Variables & EM

This appendix describes how a Gaussian mixture model, as described in Section 3.2.1, can be expressed in terms of a latent variable in order to use the Expectation-Maximization (EM) algorithm. First, the general Gaussian mixture model is recalled as

$$p(\mathbf{x}) = \sum_{k=1}^{K} \pi_k \mathcal{N}(\mathbf{x}|\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k) \tag{B.1}$$

where $\boldsymbol{\mu}_k$ is the mean vector, $\boldsymbol{\Sigma}_k$ is the covariance matrix for the $k^{th}$ Gaussian component and $\pi_k$ is the mixing coefficient. The $K \times 1$ latent random vector $\mathbf{z}$ is next introduced. The elements satisfy $z_k \in \{0, 1\}$ and $\sum_k z_k = 1$ meaning one element is 1 and $K - 1$ elements are 0. Furthermore, $z_k = 1$ is the assignment of a sample $\mathbf{x}$ to the $k^{th}$ Gaussian component. The marginal probability of $\mathbf{z}$ can be expressed such that $p(z_k = 1) = \pi_k$. The probability of $\mathbf{z}$ is given by

$$p(\mathbf{z}) = \prod_{k=1}^{K} \pi_k^{z_k} \tag{B.2}$$

The conditional distribution of $\mathbf{x}$ given $\mathbf{z}$ can be written as

$$p(\mathbf{x}|\mathbf{z}) = \prod_{k=1}^{K} \mathcal{N}(\mathbf{x}|\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)^{z_k}. \tag{B.3}$$

The joint distribution $p(\mathbf{x}, \mathbf{z}) = p(\mathbf{z})p(\mathbf{x}|\mathbf{z})$ and summed over all $\mathbf{z}$ we obtain the marginal distribution of $\mathbf{x}$. With the expression from Eq. (3.26) in mind, this allows that

$$p(\mathbf{x}) = \sum_{\mathbf{z}} p(\mathbf{z})p(\mathbf{x}|\mathbf{z}) = \sum_{k=1}^{K} \pi_k \mathcal{N}(\mathbf{x}|\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k). \tag{B.4}$$

An expression of a mixture of Gaussians involving the latent variable $\mathbf{z}$ is now available. By using Bayes rule one can express the conditional probability of $\mathbf{z}$ given $\mathbf{x}$. The probability $p(z_k = 1|\mathbf{x})$ is denoted $\gamma(z_k)$. This can be thought of as the *a posteriori* probability that some observed $\mathbf{x}$ comes from the $k^{th}$ Gaussian component. It is also called the responsibility and is defined as

$$\gamma(z_k) \triangleq p(z_k = 1|\mathbf{x}) = \frac{p(z_k = 1)p(\mathbf{x}|z_k = 1)}{p(\mathbf{x})}. \tag{B.5}$$

Since $p(z_k = 1) = \pi_k$, $p(\mathbf{x}|z_k = 1) = \mathcal{N}(\mathbf{x}|\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)$ and $p(\mathbf{x})$ is given by Eq. (3.26) the responsibility can be written as

$$\gamma(z_k) = \frac{\pi_k \mathcal{N}(\mathbf{x}|\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)}{\sum_{j=1}^{K} \pi_j \mathcal{N}(\mathbf{x}|\boldsymbol{\mu}_j, \boldsymbol{\Sigma}_j)}. \tag{B.6}$$

Note the change of the index from $k$ to $j$. The responsibility for the n$^{\text{th}}$ observation $\mathbf{x}_n$ is denoted $\gamma(z_{nk})$ and one can interpret this as how much the k$^{\text{th}}$ Gaussian component is responsible for the n$^{\text{th}}$ observation. For an observation $\mathbf{x}_n$ with dimension $D$ the data set of $N$ observations can be represented by the $N \times D$ matrix $\mathbf{X}$. The corresponding latent variables can be represented by the $N \times K$ matrix $\mathbf{Z}$. The log likelihood function is given by [Bishop, 2009, p. 433]

$$\ln p(\mathbf{X}|\boldsymbol{\pi}, \boldsymbol{\mu}, \boldsymbol{\Sigma}) = \sum_{n=1}^{N} \ln \sum_{k=1}^{K} \pi_k \mathcal{N}(\mathbf{x}|\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k). \tag{B.7}$$

To obtain maximum likelihood estimates of $\boldsymbol{\mu}_k$ the derivative wrt. $\boldsymbol{\mu}_k$ of Eq. (B.7) is set equal to 0 and solved for $\boldsymbol{\mu}_k$. This yields [Bishop, 2009, p. 435]

$$\boldsymbol{\mu}_k = \frac{1}{N_k} \sum_{n=1}^{N} \gamma(z_{nk}) \mathbf{x}_n \tag{B.8}$$

where

$$N_k = \sum_{n=1}^{N} \gamma(z_{nk}). \tag{B.9}$$

Likewise, the covariance matrix can be estimated as [Bishop, 2009, p. 436]

$$\Sigma_k = \frac{1}{N_k} \sum_{n=1}^{N} \gamma(z_{nk})(\mathbf{x}_n - \boldsymbol{\mu}_k)(\mathbf{x}_n - \boldsymbol{\mu}_k)^T \tag{B.10}$$

and finally the mixing coefficient can be estimated as

$$\pi_k = \frac{N_k}{N}. \tag{B.11}$$

The formulation of the Gaussian mixture model using a latent variable allows the use of the EM algorithm as described in algorithm 2.

**Algorithm 2:** Expectation-Maximization

**Step 1:** Initiate the parameters $\mu_k$, $\Sigma_k$ and $\pi_k$ for each Gaussian component.
**Step 2:** Assign each data point $\mathbf{x}_n$ an responsibility score given by Eq. (B.6). This is also called the E-step.
**Step 3:** Given the responsibilities, the parameters $\mu_k$, $\Sigma_k$ and $\pi_k$ is adjusted for each component. This is also called the M-step.
**Step 4:** Evaluate the log likelihood function. Furthermore, check convergence criterion for the parameters. If convergence is not met, go to step 2.

# Appendix C

# Derivation of the Backpropagation Algorithm

In this appendix the backpropagation algorithm and the learning rules, used to train a multilayer FNN, will be derived. In the current derivation, the backpropagation algorithm is based on the sum-of-squares error function and the learning rules are based on the gradient descent optimization method. The derivation is based on a 2-layer fully connected FNN similar to the one shown in Fig. 4.4 with generalized neurons as shown in Fig. 4.2. The input layer is referred to as layer $I$, the hidden layer is referred to as layer $J$ and the output layer is referred to as layer $K$. Furthermore, $w_{ji}$ is a weight in the hidden layer, on the arch between input unit $i$ and hidden unit $j$. Likewise, $w_{kj}$ is a weight in the output layer, on the arch between hidden unit $j$ and output unit $k$. The $k^{th}$ neuron output, using a sigmoid activation function, is given by

$$y_k = \varphi(z_k) = \frac{1}{1+e^{-z_k}} \tag{C.1}$$

where the corresponding accumulator output is given by

$$z_k = \sum_{j=0}^{J} w_{kj} x_j. \tag{C.2}$$

Let $\mathbf{w} = [w_{k0},\ w_{k1},\ \dots\ ,\ w_{kJ}]^T$ denote a weight vector consisting of $J$ weights to unit $k$. As mentioned earlier, the bias terms are included in the summation in Eq. (C.2) such that they are treated as weights connected to a fixed input having the value of positive one. In order to develop a learning rule using gradient descent, the gradient must be computed. The gradient of interest is the rate of change of the error function relative to the weights at a given layer. The gradient descent algorithm is given by

$$\mathbf{w}(n+1) = \mathbf{w}(n) + \nabla\mathbf{w}(n) \tag{C.3}$$

where $\nabla\mathbf{w}(n)$, is given by

$$\nabla\mathbf{w}(n) = -\mu \frac{\partial E(\mathbf{w}(n))}{\partial \mathbf{w}(n)} \tag{C.4}$$

and $\mu$ is the learning rate. Equation (C.4) can be expressed in component form as

$$\nabla w_{kj}(n) = -\mu \frac{\partial E(\mathbf{w}(n))}{\partial w_{kj}(n)}. \tag{C.5}$$

The error function $E(\mathbf{w}(n))$ is the sum-of-squares of the difference between the output $y_k$ and a training case $t_k$ and is given by

$$E(\mathbf{w}(n)) = \frac{1}{2} \sum_{k \in K} (y_k(n) - t_k(n))^2. \tag{C.6}$$

Equation (C.6) shows the error function for a single training case and the index $n$ refers to the different training cases. To keep the notation simple the index $n$ is omitted in the following derivation. In Eq. (C.3) it is assumed that the weights $\mathbf{w}$ are updated after every training case. This form of learning is referred to as on-line learning. If all training examples were evaluated before the weights were updated it is referred to as batch learning. Also a stochastic learning approach exists where a fixed number of training cases are selected by random. What learning method to choose depends on the application and influences the learning speed, memory consumption and generalization capabilities [Duda et al., 2001, p. 316].

The learning rule for a FNN can be divided into two phases. The feed-forward phase and the feed-backward phase. The feed-forward phase is where a training case is added to the input of the FNN and the data is propagated forward through the FNN in a manner like Eq. (4.4). The output $y_k$ is then compared to the test case $t_k$ and the error is computed according to Eq. (C.6). The second phase is where the error is propagated backwards into the network in order to update the weights. This forward and backward operation is repeated for all training cases several times until the error is below some defined threshold which indicates that the FNN has satisfactory performance on the training cases. When all training cases have been applied to the FNN once, an epoch has elapsed. Typically it is needed to run several epochs before the error is below the desired threshold.

The backpropagation phase can also be divided into two. One phase for updating the weights belonging to an output and one phase for updating weights belonging to a hidden unit. The first phase is handled by the delta rule and the second phase, which is the solution to the credit assignment problem, is handled by the backpropagation algorithm.

The first phase of the backpropagation algorithm is to develop a rule to update a weight in an output layer based on the current outputs and the test data. This is done by rearranging Eq. (C.5) such that the gradient is expressed in terms outputs and training data. For simplicity, the learning rate $-\mu$ will be omitted for now. The error derivative of interest is given by

$$\frac{\partial E(\mathbf{w})}{\partial w_{kj}} = \frac{\partial}{\partial w_{kj}} \frac{1}{2} \sum_{k \in K} (y_k - t_k)^2. \tag{C.7}$$

By applying the chain rule and observing that $w_{kj}$ only depends on one $k$, Eq. (C.7) can be rewritten as

$$\frac{\partial E(\mathbf{w})}{\partial w_{kj}} = \frac{\partial}{\partial w_{kj}} \frac{1}{2}(y_k - t_k)^2 = \frac{\partial y_k}{\partial w_{kj}}(y_k - t_k). \tag{C.8}$$

By observing that $y_k$ depends on $z_k$ through Eq. (C.1) and that $z_k$ depends on $w_{kj}$ through Eq. (C.2) and furthermore applying the chain rule to Eq. (C.8) the following expression can be obtained

$$\frac{\partial E(\mathbf{w})}{\partial w_{kj}} = \frac{\partial y_k}{\partial z_k}\frac{\partial z_k}{\partial w_{kj}}(y_k - t_k). \tag{C.9}$$

From Eq. (C.9) the two partial derivatives are solved separately. $y_k$ is given by Eq. (C.1) so $\frac{\partial y_k}{\partial z_k}$ is simply the derivative of the sigmoid function with respect to $z_k$ as given by

$$\frac{\partial y_k}{\partial z_k} = \frac{d}{dz}\left(\varphi(z_k)\right) = \frac{d}{dz}\left(\frac{1}{1+e^{-z_k}}\right) = \varphi(z_k)(1-\varphi(z_k)). \tag{C.10}$$

Likewise, $z_k$ is given by Eq. (C.2) so $\frac{\partial z_k}{\partial w_{kj}}$ is the derivative of $z_k$ with respect to $w_{kj}$. As seen from Eq. (C.11), $\frac{\partial z_k}{\partial w_{kj}} = x_k$, which is the input to the output layer but since the output of the hidden layer is input to the output layer $x_k = y_j$.

$$\frac{\partial z_k}{\partial w_{kj}} = \frac{\partial}{\partial z_k}\sum_{j=0}^{J} w_{kj}x_j = x_k = y_j \tag{C.11}$$

inserting (C.10) and (C.11) into (C.9), we find

$$\frac{\partial E(\mathbf{w})}{\partial w_{kj}} = (y_k - t_k)\varphi(z_k)(1-\varphi(z_k))y_j \tag{C.12}$$

By recognizing that $\varphi(z_k) = y_k$, Eq. (C.12) can be rewritten as

$$\frac{\partial E(\mathbf{w})}{\partial w_{kj}} = (y_k - t_k)y_k(1-y_k)y_j. \tag{C.13}$$

By defining $\delta_k = (y_k - t_k)y_k(1-y_k)$, Eq. (C.13) can be rewritten as

$$\frac{\partial E(\mathbf{w})}{\partial w_{kj}} = \delta_k y_j. \tag{C.14}$$

Combining Eq. (C.14) and (C.5) the delta learning rule is obtained [Duda et al., 2001, p. 291]

$$\nabla w_{kj} = -\mu\delta_k y_j. \tag{C.15}$$

Equation (C.15) describes how a given weight $w_{kj}$ in the output layer must be changed in terms of the outputs $y_j$, $y_k$ and the target value $t_k$.

The second phase of the backpropagation algorithm is to find a learning rule similar to Eq. (C.15) but with respect to the weights in the hidden layer $w_{ji}$. The error derivatives of interest are the partial derivative of the error function with respect to the weights of the hidden units $w_{ji}$, which is given by

$$\frac{\partial E(\mathbf{w})}{\partial w_{ji}} = \frac{\partial}{\partial w_{ji}} \frac{1}{2} \sum_{k \in K} (y_k - t_k)^2. \tag{C.16}$$

Since the FNN is assumed to be fully connected, meaning that a single weight in the hidden layer influences all the outputs in the output layer, the summation in (C.16) cannot be omitted, as in the derivation of the delta rule. The first step is then to move the differentiation inside the summation and apply the chain rule which gives

$$\frac{\partial E(\mathbf{w})}{\partial w_{ji}} = \sum_{k \in K} (y_k - t_k) \frac{\partial y_k}{\partial w_{ji}}. \tag{C.17}$$

It can be observed that $y_k$ depends directly on $z_k$ and $z_k$ depends indirectly on $w_{ji}$ so by applying the chain rule to Eq. (C.17), it follows that

$$\frac{\partial E(\mathbf{w})}{\partial w_{ji}} = \sum_{k \in K} (y_k - t_k) \frac{\partial y_k}{\partial z_k} \frac{\partial z_k}{\partial w_{ji}}. \tag{C.18}$$

As shown in Eq. (C.10) $\frac{\partial y_k}{\partial z_k} = \varphi(z_k)(1 - \varphi(z_k))$, and by recognizing that $z_k$ depends on $w_{ji}$ via $y_j$, the chain rule can once again be applied to obtain

$$\frac{\partial E(\mathbf{w})}{\partial w_{ji}} = \sum_{k \in K} (y_k - t_k) \varphi(z_k)(1 - \varphi(z_k)) \frac{\partial z_k}{\partial y_j} \frac{\partial y_j}{\partial w_{ji}}. \tag{C.19}$$

By applying the relation $\varphi(z_k) = y_k$ and recognizing that $\frac{\partial z_k}{\partial y_j} = w_{kj}$ the following is given

$$\frac{\partial E(\mathbf{w})}{\partial w_{ji}} = \sum_{k \in K} (y_k - t_k) y_k (1 - y_k) w_{kj} \frac{\partial y_j}{\partial w_{ji}} \tag{C.20}$$

Intuitively, it makes sense that $\frac{\partial z_k}{\partial y_j} = w_{jk}$ since the only change that happens, on the arc between output unit $y_j$ in the hidden layer and input unit $z_k$ in the output layer, is the weight on that arc $w_{kj}$.

The chain rule can be applied once again by recognizing that $y_j$ depends on $w_{ji}$ via $z_j$ hence

$$\frac{\partial E(\mathbf{w})}{\partial w_{ji}} = \sum_{k \in K} (y_k - t_k) y_k (1 - y_k) w_{kj} \frac{\partial y_j}{\partial z_j} \frac{\partial z_j}{\partial w_{ji}}. \tag{C.21}$$

Similar to the step from Eq. (C.18) to (C.19), it can be found that $\frac{\partial y_j}{\partial z_j} = \varphi(z_j)(1 - \varphi(z_j) = y_j(1-y_j)$. Moreover, $\frac{\partial z_j}{\partial w_{ji}}$ is just the derivative of Eq. (C.2) with indices $ji$ which yields $\frac{\partial z_j}{\partial w_{ji}} = x_j = y_i$. Applying these steps to Eq. (C.21) leads to

$$\frac{\partial E(\mathbf{w})}{\partial w_{ji}} = \sum_{k \in K} (y_k - t_k) y_k (1 - y_k) w_{kj} y_j (1 - y_j) y_i. \tag{C.22}$$

Rearranging Eq.(C.22) and recalling the definition of $\delta_k$ from Eq. (C.14), it follows that

$$\frac{\partial E(\mathbf{w})}{\partial w_{ji}} = y_i y_j (1 - y_j) \sum_{k \in K} \delta_k w_{kj}. \tag{C.23}$$

By defining $\delta_j = y_j(1-y_j) \sum_{k \in K} \delta_k w_{kj}$ we arrive at

$$\frac{\partial E(\mathbf{w})}{\partial w_{ji}} = \delta_j y_i. \tag{C.24}$$

Equation (C.24) shows how a weight in the hidden layer depends on all outputs in the subsequent layer. By combining this expression with Eq. (C.5) the learning rule for updating the weights of a hidden unit is given by

$$\nabla w_{ji} = -\mu \delta_j y_i. \tag{C.25}$$

The backpropagation algorithm has now been derived and is given by Eq. (C.15) and Eq. (C.25) [Duda et al., 2001, p. 292]. Equation (C.15) is used to change a weight in an output layer and Eq. (C.25) is used to change a weight in a hidden layer and can theoretically be used on any number of hidden layers. Equation (C.25) is also the solution to the credit assignment problem since it shows how a weight in a hidden layer can be expressed in terms of all the outputs in the subsequent layers which is the core concept of backpropagation. The algorithm used to update the weights does not need to be the gradient descent method as shown in Eq. (C.3). The gradient descent method is a first order method since it only uses the gradient and faster convergence could be achieved if second order methods such as the Conjugate-Gradient method or the Newtons method was used [Haykin, 1999, p. 236] [Duda et al., 2001, pp. 318-323].

# Appendix D

# Implementation Overview

This appendix describes some of the details about the implementation of the DNN based speech enhancement system which is illustrated in Fig. 6.1. The system is implemented in MATLAB and the source files, noise files as well as the HINT corpora, are located in the `matlab\DNN_Speech_Enhancement` folder on the appendix DVDs. This appendix is not a complete description about the implementation but merely a help if one is intending to run or modify the code.

## D.1 DNN System Environment Creation

To train and test the system, a DNN environment must be set up, which is done by running either `gen_test_env_windows.m` or `gen_test_env_linux.m` depending on the operating system. Before running one of these files a number of parameters must be set such as the name of the environment, the path to store the environment, the noise type to use, the number of training cases, the test cases and the validation cases, the SNR levels of the noisy signal, the local criterion, the number of frequency channels used etc. The default parameters are the once used for test S5-1/2 from Tab. 7.1. Also the number of layers can be set by changing the `dnn_env_source_dir` variable to point at the folder containing the source files for a 2, 3 or 4 layer system. When the file is executed a DNN environment is created and depending on the size of the training set it can take from a few minutes to several hours to create and take up tens of gigabytes of hard disk space. Using the default settings of S5-1/2 the DNN environment takes up approximately 27 GB. The DNN environment follows the structure shown by Fig. D.1. The `metadata.txt` file contains a list of all the settings as well as lists of all the speech files used for generating the training, validation and test data. The `train_dnn_sys.m` is used to train the system and `eval_dnn_sys.m` is used to evaluate the performance of the system. The different folders are described in Section D.3 and the training and evaluation procedure is described in D.2.

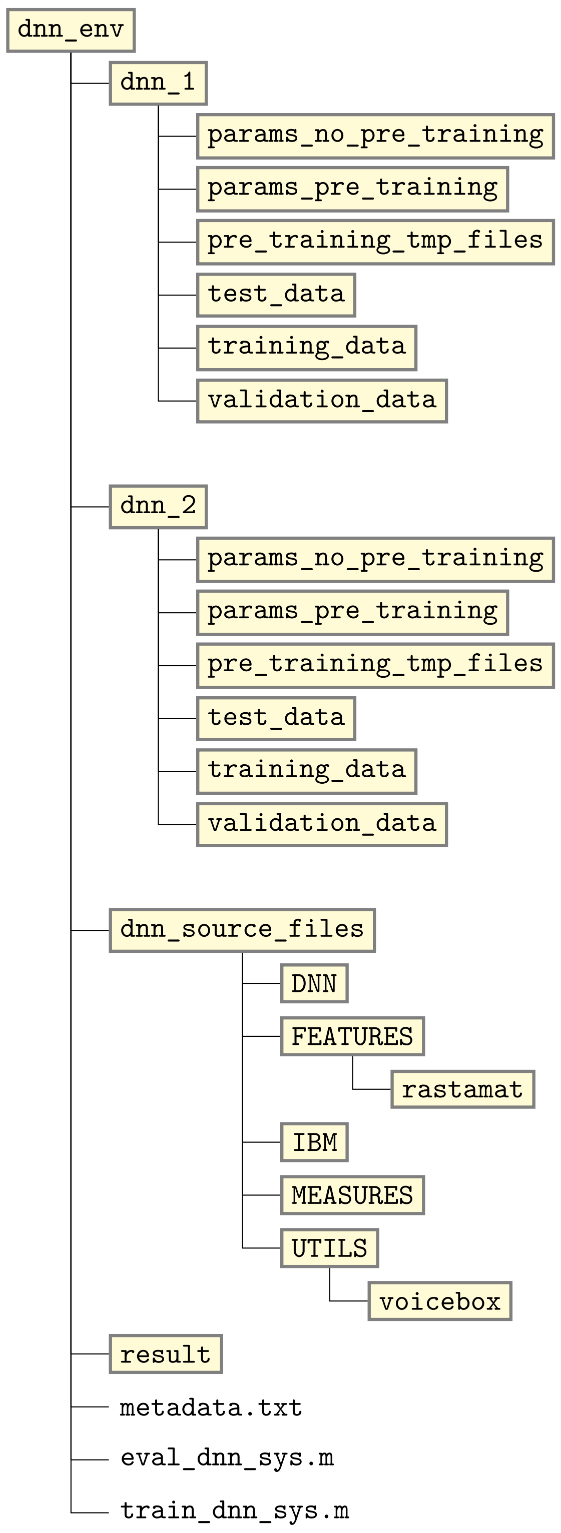


**Figure D.1:** This figure presents the folder structure of the implementation of the DNN based speech enhancement system presented in this thesis

## D.2 DNN System Training & Evaluation

To train the speech enhancement system, the file `train_dnn_sys.m` must be setup and run. The parameters to set are the following. The use of pre-training or not, the batch size, the number of hidden units for the hidden layers, the number of epochs for pre-training, the number of maximum epochs for fine-tuning for the first and second classification stage, the number of parallel workers used to speedup the training and the early stopping criteria. When these parameters are set the file can be run. MATLAB Parallel Computing Toolbox Version 6.5 (R2014b) is required if parallel computing is utilized. When the training is finished the generated parameters, i.e. the DNN weights and biases are stored in `dnn_1` and `dnn_2`. On an Intel Core i7 920 CPU running 2.67 GHz it takes approximately 67 hours to complete the training when 8 parallel workers are utilized. Likewise, 6 GB of memory is required to train the system with the S5-1/2 settings.

To evaluate the performance of the trained speech enhancement system the `eval_dnn_sys.m` file must be setup and run. The parameters to setup are the following. Whether or not to use weights computed from pre-trained DNNs or weights computed from DNNs trained with backpropagation alone. Whether or not the training, test or validation data should be used. What SNR to use, what noise type to use, the LC and the threshold for which a T-F unit is classified as speech dominated. Each test, validation or training sample is then processed by the speech enhancement system and HIT, FA, HIT-FA, STOI and PESQ are computed and stored in a text file. Also the clean, noisy, the IBM processed signal, the estimated IBM processed signal and the soft mask processed signal are stored as wav files. Likewise are the time domain representations and spectrograms of all the different signals. All these outputs are stored in individual folders in the result folder.

## D.3 File Structure & Description

This section describes the folders and some of the most used files in the DNN environment. The folder `dnn_env`, illustrated as the top yellow box in Fig. D.1 is named after the variable `env_name` in `gen_test_env_xxxxx.m` and contain all the files and folders required to train and test the speech enhancement system. The `dnn_1` folder contains the data structures which are related to the first DNN classifier illustrated by the *DNN 1* boxes on Fig. 6.1. These data structures contain the training, validation and testing data, which are based on the HINT speech corpus and the chosen noise. When the system is trained it can be chosen whether or not pre-training should be utilized. If pre-training is utilized the DNN parameters, i.e., the weights and biases, for all subband DNNs will be saved in `params_pre_training`. If only backpropagation is used the DNN parameters are saved in the `params_no_pre_training` folder. All the temporary files regarding the pre-training are stored in `pre_training_tmp_files`.

This is similar for the `dnn_2` but these are related to the second classification stage where the training, validation and test data is based on a posterior mask created from the output of the first classification stage. This is related to the boxes labeled with *DNN 2* on Fig. 6.1.

The `dnn_source_files` folder contains all source files, which make up the DNN system. This includes the files used for feature extraction and IBM creation as well as speech analysis and synthesis. As far as possible a single struct is used to store all variables regarding the DNN training except for the weights and biases. This struct is called DnnData and is the argument used for many of the functions. Some of the important files will briefly be described in the following.

The `DNN` folder contains all files related to the DNN training and the most important ones are the following:

- `trainDNN.m`
- `makebatches.m`
- `rbmvislinear.m`
- `rbm.m`
- `backpropclassify.m`
- `minimize.m`
- `CG_CLASSIFY.m`

The `trainDNN.m` file is a function which train a single DNN. This function is called several times from `train_dnn_sys.m` located in the root of the `dnn_env` directory, one function call per subband DNN. The `makebatches.m` file generates mini-batches by randomly permuting the training data based on the data structures in the `dnn_1/2` folders. The `rbmvislinear.m` implements a GBRBM trained with one step contrastive divergence and `rbm.m` implements in a similar fashion a BBRBM. In `rbmvislinear.m` and `rbm.m` the different learning rates are defined as well as the momentum and weight decay.

In `backpropclassify.m` a forward neural network is implemented initialized either by random or by the weights stored in the `pre_training_tmp_files` folder. The neural network is trained using backpropagation and the conjugate gradient algorithm is implemented in `minimize.m` and the backpropagation gradients are computed in `CG_CLASSIFY.m`.

The `Features` folder contains all files related to feature extraction. The folder also contains the rastamat toolbox [Ellis, 2005]. The following two files perform the top level feature extraction.

- `gen_AMS.m`
- `gen_features_and_labels.m`

The `gen_AMS.m` file computes the AMS features based on the DnnData struct containing the raw acoustic signal and the `gen_features_and_labels.m` file computes the IBM, MFCC and RASTA-PLP features using the rastamat toolbox and the labels based on the IBM, also using the DnnData struct.

The `IBM` folder contains all files related to speech synthesis. These files are primarily from [Jin and Wang, 2007b]. The most important files are the following two:

- `ibm_synthesis.m`
- `gammatone.m`

The `ibm_synthesis.m` file makes speech synthesis based on a binary mask or a mask based on rational numbers. The `gammatone.m` implements a gammatone filter bank, which is used to construct a T-F representation.

The `MEASURES` folder contains the implementation of the PESQ quality measure and the STOI intelligibility measure. These implementations are from [Loizou, 2013] and [Taal et al., 2010] respectively and is implemented in the following two files:

- `pesq.m`
- `stoi.m`

The `UTILS` folder contains different helper functions as well as the voicebox toolbox. Two helper functions which are worth mentioning are the following:

- `gen_post_mask.m`
- `gen_post_mask_feature_and_label.m`

The `gen_post_mask.m` makes a posterior mask based on the DNN parameters as well as the training, validation and test data in the `dnn_1` folder.
The `gen_post_mask_feature_and_label.m` file uses this posterior mask to construct new training, validation and test data for the second classification stage, which are stored in their respective folders in the `dnn_2` folder. The posterior mask features are constructed using a window of posterior probabilities similar to Fig. 6.5. If 24 bands are used the window span 5 time frames and 7 frequency bands and if 64 channels are used the window span 5 time frames and 17 channels.

# Appendix E

# DNN Parameters

This appendix lists the DNN parameters which are common for all simulations in Chapter 7.

## E.1 Feature Parameters

- 31 MFCC features are used computed using the *melfcc* RASTAMAT function. The parameters can be found on line 57 in `gen_features_and_labels.m` file.
- 15 AMS features are used which are computed based on the implementation described in Section 6.3.2. The implementation can be found in the `gen_AMS.m` file.
- 13 RASTA-PLP features are used computed by the *rastaplp* RASTAMAT function. The parameters can be found on line 61 in `gen_features_and_labels.m` file.
- 13 RASTA-PLP delta features and 13 RASTA-PLP delta-delta features are used which are computed using the *deltas* RASTAMAT function. The parameters can be found on line 62-63 in `gen_features_and_labels.m` file.
- A Posterior Mask window consisting of 5 time frames and 17 frequency frames is used. This is further described in Section 6.3.1.

## E.2 DNN System Parameters

- A learning rate for BBRBM of 0.1 is used.
- A learning rate for BBRBM of 0.01 is used.
- A weight decay of 0.0002 were used for RBM training.
- An initial momentum factor of 0.5 were used for the first 5 epochs of RBM training and 0.9 for the remaining epochs.
- Batch size: 512

- A DNN with two hidden layers is used.
- 200 hidden units are used for each hidden layer in the DNN
- A non-linear conjugate gradient algorithm is used for supervised training. The implementation is taken from [Salakhutdinov and Hinton, 2006]
- A maximum number of 500 epochs are used for all supervised training.
- The early stopping criterion is set to 50 epochs.
- 100 epochs of pre-training is used for all DNNs where pre-training is utilized.
- 64 frequency channels are used.
- The bandwidth is from 0.1 kHz - 8 kHz
- The frame width is 20 ms.
- The frame overlap is 10 ms.
- The sampling frequency is 16 kHz.
- Binary classification threshold for speech dominated T-F unit is 0.5.
- In the sampling step of the GBRBM in CD only the mean value is used and no actual sample is drawn [Wang and Wang, 2013].