# Probabilistic Speech Detection

Daniel J. Jacobsen

IMM-THESIS-2003-50

IMM

Printed by IMM, DTU

# Preface

This M.Sc. thesis is the final requirement to obtaining the degree of Master of Science in Engineering. The work has been carried out in the period from the 1st of February 2003 to the 1st of September 2003 at the Intelligent Signal Processing group at the Institute of Informatics and Mathematical Modelling, Technical University of Denmark. The work has been supervised by Associate Professor Jan Larsen and co-supervised by M.Sc., Ph.D. Søren Riis, Oticon A/S. I wish to thank Professor Lars Kai Hansen for additional guidance, and M.Sc. Peter Ahrendt for many useful and inspiring discussions as well as proof-reading assistance.

Kgs. Lyngby, September 1st, 2003

Daniel J. Jacobsen, s973341

# Resumé

Denne afhandling omhandler detektion af tale i signaler der indeholder meget forskellige typer støj. Dette problem kaldes 'VAD' (for 'Voice Activity Detection', dvs. 'detektion af taleaktivitet'). Signalerne består af segmenter af ren støj og segmenter med både tale og støj i en additiv blanding. To forskellige probabilistiske metoder implementeres for at løse VAD problemet. Den ene er en metode baseret på diskriminant funktioner, hvor et lineært netværk med ét logistisk output trænes til at give sandsynligheden for tilstedeværelsen af tale i et givet lydsignal. Den anden metode er baseret på modellering af klasse-betingede sandsynlighedstætheder, hvortil anvendes ICA (for 'Independent Component Analysis', dvs. 'Uafhængig Komponent Analyse'). Algoritmerne afprøves og sammenlignes. De sammenlignes også med en industri standard VAD algoritme, nemlig den tilhørende ITU-T G.729B anbefalingen, og en anden VAD algoritme. Resultaterne viser hvor afgørende vigtigt det er at tage typen af støj med i betragtning for at opnå robust tale detektion, og at det for visse støjtyper er muligt at opnå bedre resultater med de udviklede algoritmer.

***Nøgleord:*** *maskin læring, klassifisering, stemme aktivitet detektion, lineære netværk, uafhængig komponent analyse, modtager operations karakteristika*

## Abstract

This thesis deals with the detection of speech in signals that may contain very different noise types, referred to as the 'Voice Activity Detection' (VAD) problem. The signals consist of sections of noise only and sections of speech and noise in an additive mixture; convolutive mixtures are not addressed. Two different probabilistic methods are developed to solve the VAD problem. One is a discriminant-function based method in which a linear network with a single logistic output is trained to output the probability of speech presence from a given sound signal. The other is based on modelling of class-conditional probability densities, using Independent Component Analysis (ICA) methods. The algorithms are tested extensively and comparisons are made between them. They are also compared to an industry standard VAD algorithm, namely that of the the ITU-T G.729B recommendation and one other VAD. The results show the crucial importance of considering the type of noise present with the speech for obtaining robust speech detection and that for certain noise types, performance can be bettered with the developed VAD algorithms.

***Keywords:*** *machine learning, classification, voice activity detection, linear networks, independent component analysis, receiver operating characteristics*

# Contents

# Part I

# Background

# Chapter 1

# Introduction

Speech signals take a very special place amongst all other audio signals. To humans, they are not only special because they can be generated by themselves, but most of all because they carry information. Even in this modern age, much of the information that we receive comes in the form of speech. And most other audio signals that we perceive do not carry any information as such. Indeed, much would be classed as 'noise' in everyday situations.

Because of this special significance of speech signals, much work has been done in order to be able to automatically detect the presence of speech in noisy signals. This is for instance the case with cellular phone networks, where modern (e.g. GSM) phones actually stop transmitting if they detect the absence of speech, allowing on average around 3 times as much traffic to be sent using the same bandwidth[1].

The term 'speech detection' is often used interchangeably with the term 'Voice Activity Detection' or 'VAD', even though 'voice activity' may of course be a variety of things other than strictly speech. In any case, the majority of human voice activity could be called 'speech' and the two terms are also used interchangeably in this report. 'VAD' is also used to refer to any algorithm or system that is designed to detect speech, and then stands for 'Voice Activity Detector'.

A closely related problem is that of speech *enhancement*, where the object is to remove as much noise as possible thus 'cleaning' the speech and making it easier to understand. Depending on the approach, it may also be termed 'noise reduction'.

While Automated Speech Recognition (ASR) is often the motivation for denoising or separating out the speech signal (see e.g. [27]), in the hearing aid context it is also relevant purely for the purpose of producing cleaner speech for the benefit of the hearing aid user.

Although this problem is not treated specifically in this report, it deserves mention due to the close relation to the VAD problem, theoretically and algo-

[1]This goes by the name of 'Variable Transmission Rate', 'DTX' etc.

rithmically.

The approach taken in this work for the development of VAD systems is a probabilistic, machine-learning one. Such systems are able to give probabilities of the presence of speech, and 'learn' to do this correctly through 'training' on audio-signal examples.

The algorithms are implemented in `Matlab`[2] and are compared with two other VAD's, namely a VAD described in [4] and an industry standard VAD, namely that of the ITU-T G.729B recommendation (referred to as the ITU-T VAD ). The VAD described in [4] will henceforth be referred to as the OTI VAD and this is exclusively meant to designate the particular VAD described in [4].

## 1.1 Motivation

The VAD research field offers rich opportunities for applying machine-learning methods, which is a motivation in itself.

A different motivation comes from the hearing aid industry, for which both speech detection and speech enhancement are highly desirable goals. Persons with hearing disabilities require significant enhancement in order to be able to understand speech equally well as non-impaired persons ([25]).

In the context of hearing aids, a good VAD is useful for several purposes, for instance controlling the signal processing of the hearing aid so that it adapts to speech presence. The hearing aid can be put in 'comfort mode' (full noise reduction) when no speech is present and in 'speech mode' (no noise reduction) when speech is present. This principle is used in Oticon's Adapto hearing aid, where it is called 'VoiceFinder'.

The same principle can be extended regarding classification of audio signals into other classes, so that the hearing aid can adapt to a range of sound environments. For instance, if music is detected, the anti-feedback system present in modern hearing aid 's can be switched off so that the musical notes are not destroyed. The present work in speech detection is a first step towards this wider 'sound environment classification' problem.

Finally, inspiration comes from the human ability for audio processing. The ability of the human auditory system with respect to speech detection and -enhancement is truly awe-inspiring and represents the ultimate bench-mark for any VAD.

One might even hope to obtain some knowledge about audio processing that can say something about the way humans might solve the same problems. This can be very giving but is not an end in itself. The purpose of this project is not to model a biological system but to solve a specific engineering problem.

[2]Code is available on CD or from the author (dj@imm.dtu.dk)

## 1.2 Structure of the thesis

This thesis is organized into three parts. The first part provides the background for the project itself as well as the probabilistic classification framework. This covers the problem formulation, data material, and brief descriptions of probabilistic classification and 'feature extraction'.

The second part describes the different methods that have been studied and implemented to solve the speech detection problem. These can be grouped into *linear neural network* and *independent component analysis* methods. Some related work done by others is also mentioned.

The final part contains the experiments that have been carried out and discusses the results. This covers experimental setup, results for each method, comparison of the methods and an overall discussion and conclusion.

# Chapter 2

# Problem formulation

The objective of this work is to develop a speech detector that can classify audio input correctly into 2 classes: a speech class ($C_{VA}$) and a non-speech class ($C_{NVA}$, for 'no voice activity').

Speech detection is a well known classification problem which may sound simple but is difficult in practice. The difficulty is due to the great variety of "intrusion" signals (some of which might be naturally termed "noise") and the variety of speech signals (male/female, different rate, pitch etc. of different speakers).

There are several assumptions on the input signals that limit and focus the objective of this work in speech detection.

## 2.1 Signal model

First, the input signal $\mathbf{x}(n)$ is assumed to be an *additive* mixture of a speech signal and an intrusion signal:

$$\mathbf{x}(n) = \lambda_{\mathbf{s}}\mathbf{s}(n) + \lambda_{\mathbf{i}}\mathbf{i}(n) \tag{2.1}$$

This is the input signal model. Note that 'convolutive noise' in which the speech signal itself is distorted for instance due to reverberation is not addressed.

The signal symbols are in boldface, denoting vectors, as the input signal typically is multivariate. For instance, $\mathbf{x}(n)$ could be a frequency line from a spectrogram, or a time-domain section (a 'frame') of a speech signal.

The basic premise is that $\mathbf{x}(n)$ is known and available, while $\mathbf{s}(n)$ and $\mathbf{i}(n)$ are not. The scaling of each signal, $\lambda_{\mathbf{s}}$ and $\lambda_{\mathbf{i}}$, are also unknown, as is the absolute scaling of $\mathbf{x}(n)$. $\mathbf{x}(n)$ is the signal as it would be picked up by say a hearing aid microphone.

The speech signal, $\mathbf{s}(n)$, is assumed to be made up of an alternating sequence of active speech and pauses (between words or sentences or when no-one speaks).

A further assumption is that - as in most real-life situations - the noise source will almost never be completely absent at any given time, whereas the speech signal will often be.

This means that the $\mathbf{x}(n)$ is at all times *either* noise only *or* a mixture of speech and noise. In other words, $\lambda_{\mathbf{i}}$ is assumed to always exceed zero. This assumption is made to comply with real-life listening situations, where there is always at least *some* noise.

## 2.2 The Signal-to-Noise Ratio measure

The Signal-to-Noise ration or SNR is a traditional measure of the relative levels of 'signal' to 'noise' in a mixture of the two, as determined by $\lambda_{\mathbf{s}}$ and $\lambda_{\mathbf{i}}$.

The 'signal' in 'SNR' is any signal that is a target signal while the 'noise' is anything that is seen as an interference in a given application. In the VAD context, the target signal is speech and anything else is noise. The SNR is then defined as

$$SNR = 10 \ \log_{10} \frac{P_s}{P_n} \tag{2.2}$$

where $P_x$ is the power of signal $x$, defined as

$$P_x = \lim_{N\to\infty} \frac{1}{2N+1} \sum_{n=-N}^{N} |x(n)|^2$$

For the signals generated for this project, $N$ is sufficiently large to give a reliable SNR measure.

The SNR is thus measured on a logarithmic scale. It is simply one way of quantifying the relative levels of signal and noise. It also has the intuitively nice properties that it is equal to 0 only when the power of signal and noise are equal, is positive when the signal is stronger than the noise and negative for vice versa.

For the mixtures to be used for this project, only SNR's between of 0 and 10 are used. The reasons for this are as follows. SNR less than 0 is uninteresting, as even humans have trouble detecting speech at these SNR's anyway and the signal is so noisey that for most intents and purposes it would be useless to class it as speech. Therefore, anything less than SNR 0 can be classed as belonging to $C_{NVA}$ *a priori*. SNR's higher than 10 are uninteresting for another reason, namely that even extremely simple detection systems will perform well for these signals. Thus it is the range from 0 to 10 that poses the relevant challenge. The focus is on the edges of this area, namely SNR 0 and SNR 10.

Many of the experiments done in this project are very time consuming, so it is necessary to choose as few SNR's as possible in order to be able to do as many different experiments as possible. With the choice made, noisy speech files can be pre-stored etc., speeding up experimental work.

### 2.2.1 Segmental SNR

The SNR measured on a long signal consisting of a mixture of speech and some noise signal will be strongly affected by the parts of the signal where speech is absent, likely resulting in an under-estimate. A better measure is the 'segmental SNR'. This is simply an SNR measured *only* over those samples that actually contain the target signal, i.e. speech, ignoring pauses:

$$SNR_{seg} = 10\log(\frac{P_{\mathbf{s}'}}{P_{\mathbf{i}'}}) \tag{2.3}$$

where $P_{\mathbf{s}'}$ is the power of the speech signal excluding pauses and $P_{\mathbf{i}'}$ is the power of the intrusion signal, also ignoring those parts that overlap speech pauses. If not otherwise specifically stated, 'SNR' henceforth refers to this measure.

## 2.3 VAD Requirements

Several requirements are desired to be met in the implemented systems, mostly originating from the hearing aid context.

### 2.3.1 Time constraints

The classification of the input signal should not take longer than 200 ms. This is so that a hearing aid can take action corresponding to the VAD signal fast enough that the hearing aid user is not discomforted.

Whether or not the classification is done on a sample-by-sample basis, or on a frame basis is not important, as long as this time constraint is met.

### 2.3.2 Robustness to noise

The speech detector should be robust to a wide range of Signal-to-Noise Ratios *and* to several different *types* of noise. While many articles on voice activity detection do operate with varying SNR, some only consider (typically) white, Gaussian noise (e.g. [24]) while others also consider different noise types, e.g. [5]. White, Gaussian noise refers to signals whose samples in the time-domain are independent (white) and where each sample is normally distributed (Gaussian).

Although these types of signals are found in real-life situations, many everyday noise types are extremely dissimilar to white Gaussian noise.

### 2.3.3 Computational speed

This is not a main requirement, as the implemented systems are not intended to be directly useable in a physical system, such as a hearing aid. However, it is still preferable to consider computational speed in any VAD design choice, as the hearing aid platform is a *potential* target for future implementation.

## 2.4 Terminology

Speech detection is interchangeably referred to as VAD. 'Detection' and 'classification' is likewise used interchangeably.

'VAD signal' is used both of the signal containing the true description of the class of each frame or sample, and also for the estimated output from a classifier; the context determines the specific meaning.

# Chapter 3

# Data

This chapter describes the data used for all experiments. This involves both speech and noise data, the mixing of the two, splitting into segments and the correct class labelling of these segments.

## 3.1 Speech

Since the class of signals referred to as 'speech' can be very diverse, a definition will be given of the class as used here for the target signals of this project.

The speech data is made up of both male and female (adult) speech, in equal proportions. Only 'normal' speech is targeted. This excludes all other forms of voice activity, such as whispering, singing and screaming.

## 3.2 Characteristics of speech

To humans, speech is a very characteristic audio signal. This may partly be because our audio perception is finely tuned to this particular class. But even just looking at a spectrogram of speech convinces of the unique characteristics of speech compared with other audio signals - see figure 3.1.

These observations of characteristics form the basis for deciding how to proceed with the first steps towards designing a VAD.

### 3.2.1 Voiced and unvoiced speech

An obvious distinction is between 'voiced' and 'unvoiced' speech. Voiced speech mainly occurs when uttering vowels, while unvoiced speech refers to most consonants (such as the 's' in 'say'). The former consists (in the spectral domain) of small repeating patterns ('pitch lines'), especially at frequencies lower than 4 kHz, while unvoiced speech is more similar to white noise.

The voiced segments last up to 400 ms while the unvoiced segments are typically around 100-200 ms.

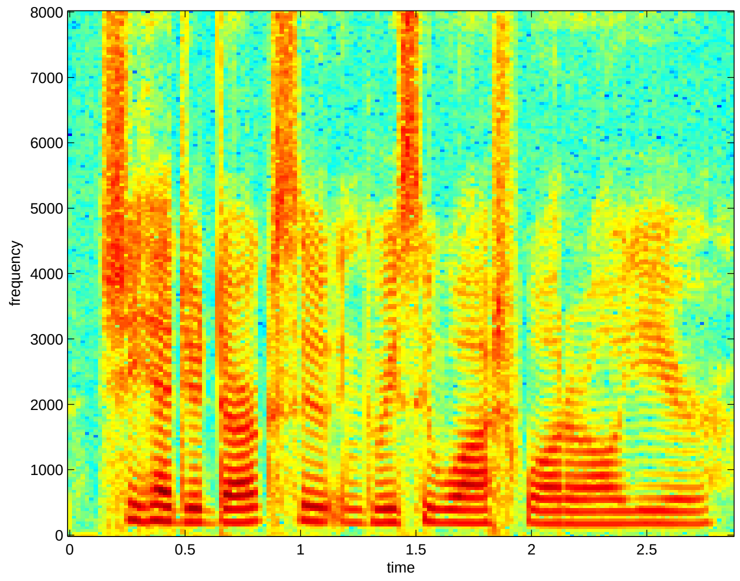

**Figure 3.1.** Spectrogram of a female speaking the sentence 'She had your dark suit and greasy wash-water all year' without any noise (TIMIT)

### 3.2.2 Frequency modulation

As time progresses, there is a characteristic frequency modulation whereby the horizontal stripes in the spectrogram (see figure 3.1) move slightly up and down.

### 3.2.3 Harmonic relations

For voiced speech, there is a rather precise 'harmonic' relation between the frequency peaks or stripes. The first peak has a frequency called the 'first formant' or 'F0' somewhere between 150 and 200 Hz, and all other peaks are located at F0 plus multiples of F0.

### 3.2.4 Unvoiced speech

Unvoiced speech drops of at frequencies higher than 8 kHz and also at frequencies lower than 3 kHz. Thus it does cover not all frequencies (as e.g. white Gaussian noise).

### 3.2.5 Common onset

Both voiced and unvoiced speech are seen to have a certain time-frequency appearance with common onset across many frequencies.

## 3.3 Audio sources

There are several so-called speech and noise ’corpora’ that are available both commercially and free. They differ widely in intended purpose and content.

The ’Aurora’ database[1] is one of the most widely used speech corpora. However, it only contains spoken digits (’one’, ’two’ etc.), not sentences. The corpus that was used for this project was instead the *TIMIT clean speech corpus* which contains a great amount of very varied speech, which is exactly what is needed for this VAD work.

### 3.3.1 TIMIT clean speech corpus

The TIMIT clean speech database, hereafter referred to as TIMIT, is an acoustic and phonetic speech corpus that has been put together for evaluation of speech processing systems [11].

It is used for the generation of all speech samples in this project.

The version of TIMIT available for this project contains 10 different sentences, spoken by 382 men and 159 women, although for a few speakers, fewer sentences are available[2]. All speakers speak the same 10 sentences.

Each sentence contains continuous speech, but a few pauses are also marked in some sentences (also depending on the speaker). Each sentence last around 2-4 seconds.

The sampling frequency is 16 kHz.

Background noise level differs widely between recordings, but is so low as to be negligible and the ’clean speech’ label is wholly justified.

### 3.3.2 Phonemes

Phonemes are the ’building blocks’ of speech - they are the semi-stationary segments that make up each spoken word. In order to distinguish between voiced and unvoiced speech, it is necessary to examine the speech signals at the phoneme level.

Details of TIMIT processing (phoneme extraction etc.) can be found in appendix A.

### 3.3.3 NOISEX

This is a database of noise audio signals. It contains a realistic babble clip. But the clip is so short, that it was decided to create babble from TIMIT instead in the interest of variety.

This was done by mixing several layers of TIMIT speakers, each layer consisting of several people speaking simultaneously. The result sounds very similar to the NOISEX babble.

[1]http://www.elda.fr/proj/aurora.html

[2]This is a technical issue dealt with by the custom-written extraction software

## 3.4 Intrusion signals

Most realistic audio signals are highly non-stationary (i.e. statistical properties vary with time). Therefore, the intrusion signals used here are also non-stationary, although white Gaussian noise is also used.

The types of noise used were chosen because they occur in normal everyday situations and have very different characteristics.

It is important to use a variety of different sounds to train and test the classifiers on for at least two reasons. One is that the system will otherwise be fitted to a relatively small set of sounds that are unrepresentative of real life and thus unable to cope with real life situations. The other - and main - reason is that it is desirable to discover just how much the noise type effects the performance of the system.

The types of noises mixed with the speech that is to be detected may be even more important than the SNR for determining the performance of the VAD. For instance, the G.729 standard VAD was shown in [29] to be rather 'overfitted' to white noise environments and having serious trouble with vehicle- and babble-type noise.

### 3.4.1 White noise

This is simply a signal where each sample is identically normally distributed and is statistically independent from all previous and following samples. Each sample is drawn from the following distribution:

$$p(x) \quad = \quad \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{1}{2}\frac{(x-\mu)^2}{\sigma^2}\right) \qquad (3.1)$$

where $\sigma^2$ is the variance of the signal and $\mu$ is the mean. For the signals used here, the variance was set to 1 and the mean to zero.

Figure 3.2 shows the same sentence as before (figure 3.1) mixed with white noise at SNR 0.

### 3.4.2 Traffic noise

This was designed by combining a random section from a recording from the inside of a volvo car with a recording of highway traffic, at different relative amplitudes for each of 100 30-second sound clips. To this were added shorter recordings of a traffic jam and a helicopter fly-by, also at random relative amplitudes and at random time points. This produced 100 30 second clips with certain similarities, but no clip was identical.

Traffic is generally very low-frequency, so it is relatively inaudible at a given SNR compared to e.g. white noise. This leads to an intuitive expectation that perhaps traffic noise will be the easiest noise to detect speech in, but the dominance of low frequencies (similar to speech) perhaps makes up for this in difficulty.

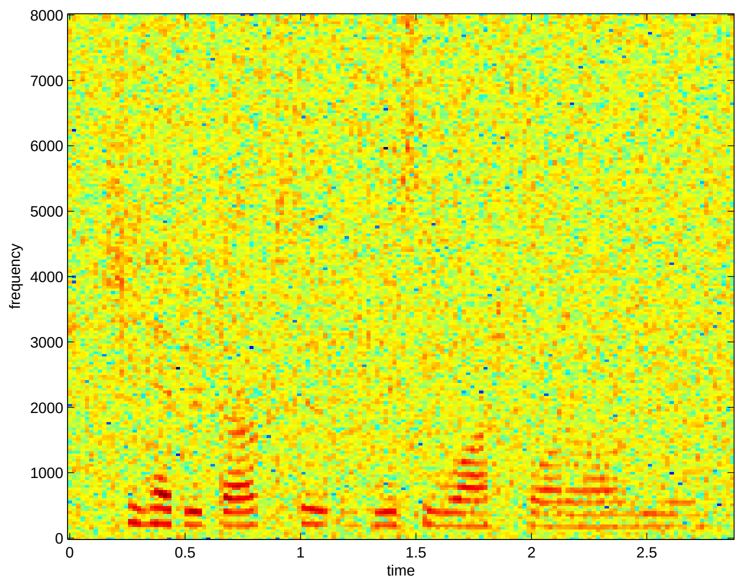

**Figure 3.2.** Spectrogram of a female speaking the sentence 'She had your dark suit and greasy wash-water all year' in white noise at SNR 0 (TIMIT).

Figure 3.3 shows the same sentence as before (figure 3.1) mixed with traffic noise at SNR 0.

### 3.4.3 Babble

'Babble' is a term used to describe 'noise' consisting of many people speaking simultaneously. 'Many' is meant here to mean enough that it is very hard to make out any particular sentence being spoken. A typical real-life situation where this is found is in crowded restaurant environments.

Figure 3.4 shows the same sentence as before (figure 3.1) mixed with babble noise at SNR 0.

Babble is clearly the worst noise type of all, as it consists of a mixture of signals of the target signal class(!).

### 3.4.4 Transients

It is highly desirable that the VAD should be robust to transient noises, such as "clicks" occurring at frequencies roughly corresponding to the fastest syllabic rate, i.e. the rate at which speakers produce phonemes, which is around 5 Hz.

Therefore, clips were created from the 'clicks.au' file from the "Martin Cooke 100" data set. This was done by re-sampling each short click sequence and stringing them together so as to produce 30 second clips consisting of short click-sequences of individually (slightly) varying frequency.

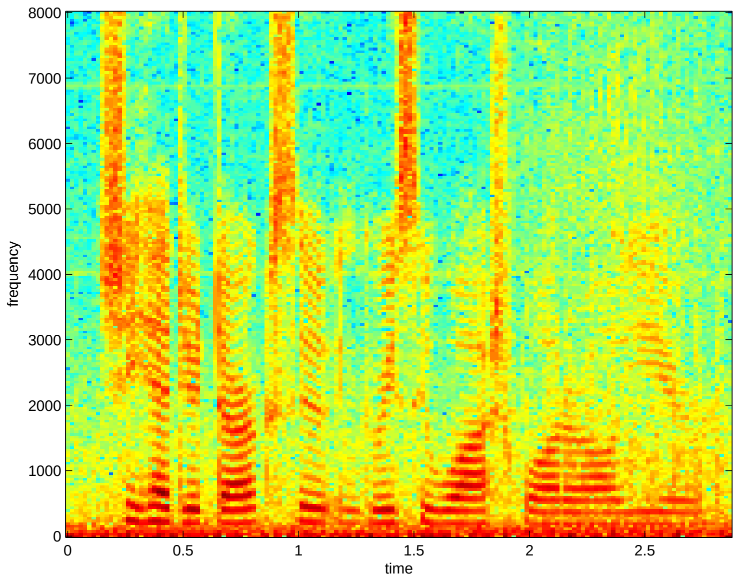

**Figure 3.3.** Spectrogram of a female speaking the sentence 'She had your dark suit and greasy wash-water all year' in traffic noise at SNR 0 (TIMIT). Note the dominance of the low frequencies, similar to speech.

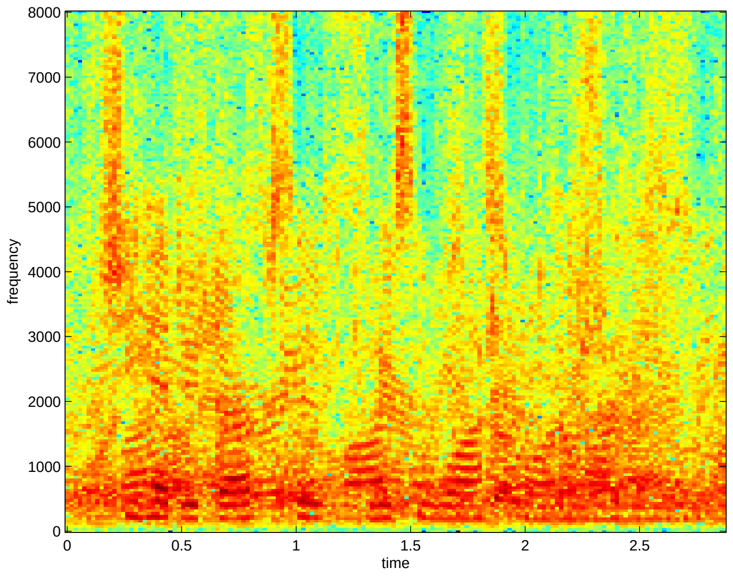

**Figure 3.4.** Spectrogram of a female speaking the sentence 'She had your dark suit and greasy wash-water all year' in babble noise at SNR 0 (TIMIT). Note the nearly complete degeneration of the target speech signal.

Figure 3.5 shows the same sentence as before (figure 3.1) mixed with this 'clicks' noise at SNR 0.

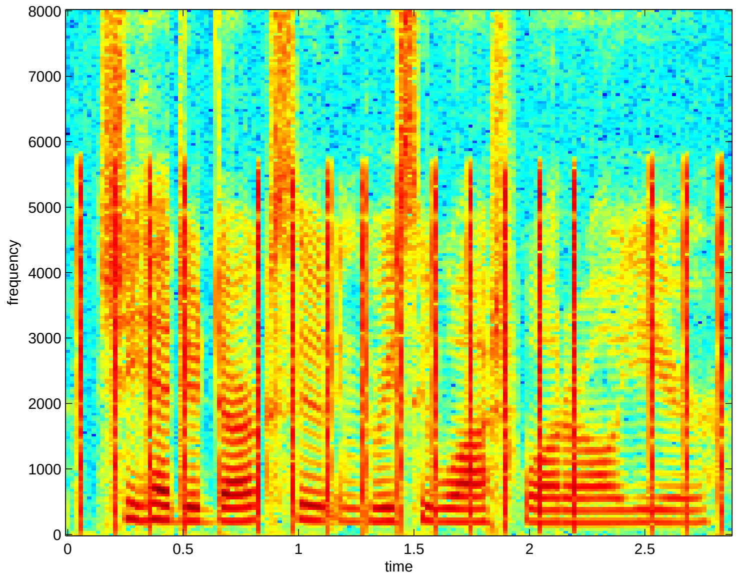

**Figure 3.5.** Spectrogram of a female speaking the sentence 'She had your dark suit and greasy wash-water all year' in 'clicks' noise at SNR 0 (TIMIT).

## 3.5 Combining speech and noise

Construction of the data set was done by combining speech and noise. For each type of noise (white, traffic, clicks and babble), a set of data was created for each choice of mean SNR

Speech was constructed into artificial 'conversations', so that each clip of 30 seconds contains 3 different TIMIT persons speaking interchangeably. A random delay (uniformly distributed from 0 to 2 seconds) was inserted between each sentence. This was done as a natural way of introducing realistic variation. Thus each 30-second clip is unique.

100 30-second clips were generated for each combination of speech, noise and SNR. These clips are of course also individually unique.

Each 30-second clip then contains 480.000 samples (at 16 kHz).

The compensatory effect caused by speakers changing their speaking style due to the presence of noise (known as the 'Lombard effect' ([7]) can of course not be taken into account with this form of synthetic data. It would naturally have had some effect on the results, but probably on a very small scale.

## 3.6 Preprocessing

Real-life audio data, such as that received by a hearing aid, is extremely dynamic. The sound environment can change abruptly such as while a person is listening to a distant speaker, a nearby person speaks directly into that person's ear. This corresponds to changes in $\lambda_\mathbf{s}$ and $\lambda_\mathbf{i}$ in 2.1.

In practice, the analog-to-digital converter of the physical VAD platform (e.g. a hearing aid) has a limited range and resolution. Therefore, the input signal must somehow be normalized in amplitude. To do this, estimates of the signal's mean and variance are required, $\hat{\mu}$ and $\hat{\sigma}$. One way of doing this if based on recursive estimation, requiring only the current sample for estimation together with the estimate of the previous sample. This method is detailed in chapter B. A single parameter, $\lambda$, then controls how quickly the variance- and mean estimates adapt to changes in the actual signal.

With $\lambda$ chosen to be 0.75, a reasonable adaptation speed is achieved. The adaptation must not be so fast as to distort single sentences, but should on the other hand not be too slow and able to adapt quite fast to sudden amplitude changes.

All data was then amplitude normalized in this way. Results are shown in figures 3.6 to 3.9. From the last figure, it is evident that any VAD relying too simply on signal energy will suffer greatly from the effects of normalization. This is only desirable, since the goal is to design a VAD that is robust to noise.

Normalization does not change the segmental SNR of the signal, as the speech and noise are scaled together.

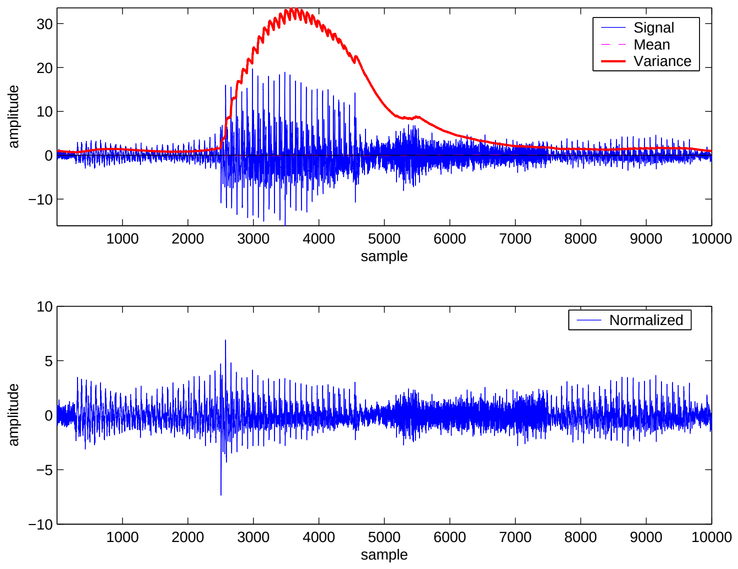

**Figure 3.6.** A segment of speech with the estimated variance signal (top) and the resulting normalized signal (bottom)

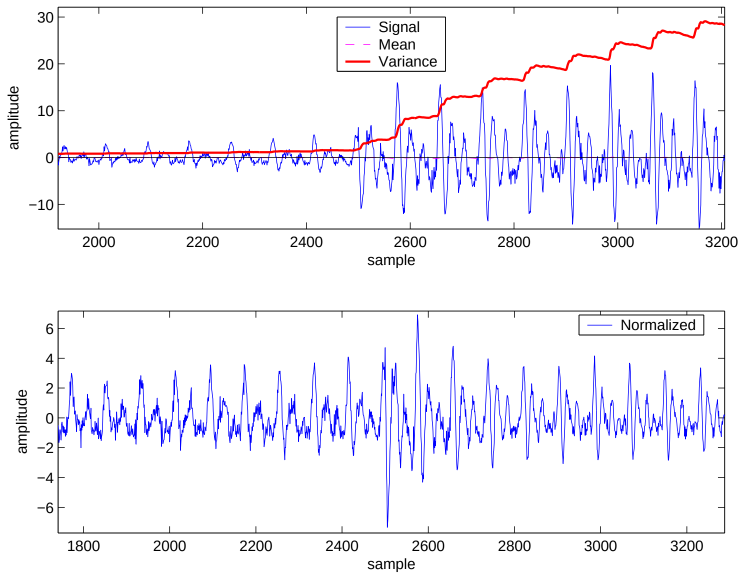

**Figure 3.7.** A close-up of a part of the signal where the un-normalized variance changes rather abruptly (above) together with the estimated variance. Bottom: the signal is (quickly) normalized but not degenerated.

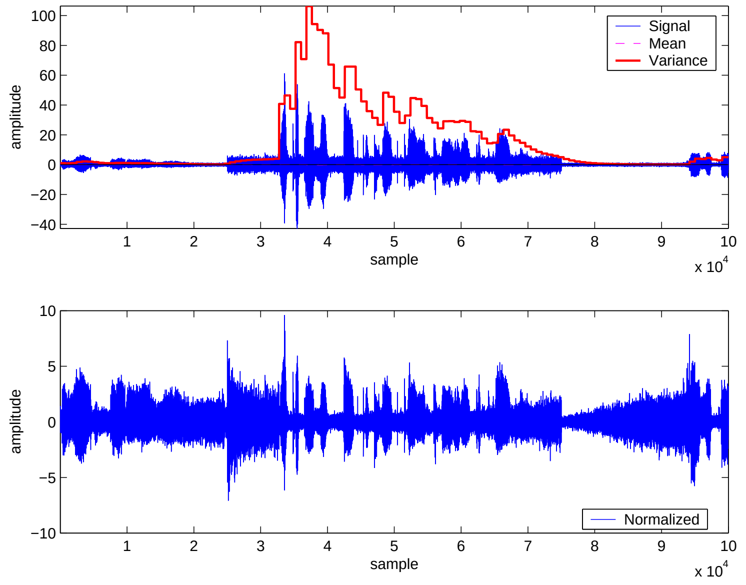

**Figure 3.8.** Normalization over blocks instead of single samples. The resulting normalized signal is shown in the bottom part.

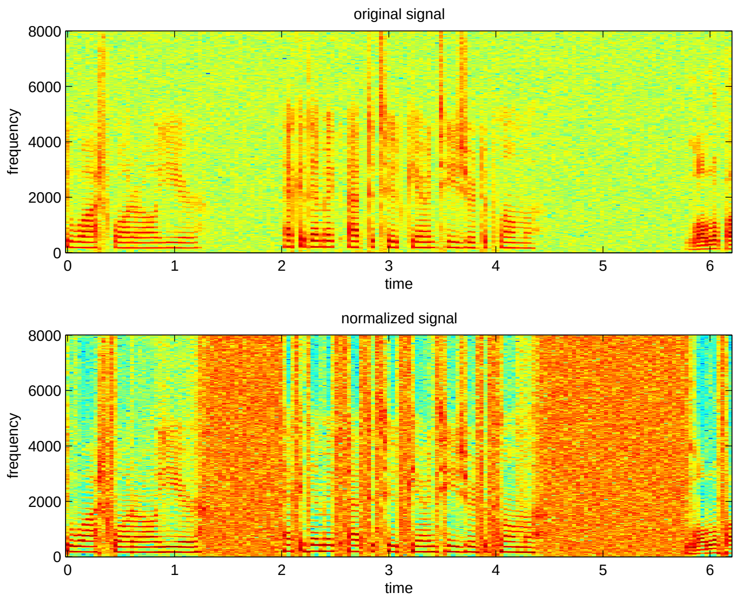

**Figure 3.9.** Spectrogram of an unnormalized (top) and a normalized speech signal (white noise at SNR 10). Note the strong effect of the normalization.

With this normalization scheme, 'clipping' will occur on abrupt amplitude increases, meaning that the signal amplitude will exceed the range of the physical system. However, this is not a consideration for this project. The key point with normalization is that it provides realistic signals that are tough to classify in a robust way.

# Chapter 4

# Probabilistic Classification

Classification problems can be approached in many ways. Here, a probabilistic approach is taken. A detailed description of theory and techniques can be found in [3], and only a short review of the most relevant issues as they pertain to the implemented system will be given here.

The basic goal of probabilistic classification is to *map* an input signal $\mathbf{x}$ to an output or outputs, namely the probability that $\mathbf{x}$ belongs to any given class $C_k$, $P(C_k|\mathbf{x})$. This is called the *posterior probability of class membership* since it is based on a given input signal.

In the present case, only one particular class is interesting, namely the class of audio signals that contain speech, as defined in chapter 3, called $C_{VA}$. Since $\mathbf{x}$ either contains speech or it does not, it must hold that

$$P(C_{VA}|\mathbf{x}) + P(C_{NVA}|\mathbf{x}) = 1 \tag{4.1}$$

where $C_{NVA}$ jointly represents all other classes of audio signal that do not contain speech. Therefore, it is only necessary to consider $P(C_{VA}|\mathbf{x})$.

From Bayes' rule we have that

$$P(C_k|\mathbf{x}) = \frac{p(\mathbf{x}|C_k)P(C_k)}{\sum_j p(\mathbf{x}|C_j)P(C_j)} \tag{4.2}$$

which means that $P(C_k|\mathbf{x})$ can be found from $p(\mathbf{x}|C_k)$ and vice versa. $p(\mathbf{x}|C_k)$ is the distribution of $\mathbf{x}$ given that it belongs to class $C_k$. Note that capital $P(\cdot)$ refers to probabilities while lower-case $p(\cdot)$ refers to a probability distribution ($C_k$ are discrete while the $\mathbf{x}$ are real-valued).

This is the starting point of probabilistic classification and means that there are two main approaches for proceeding, namely trying to learn $P(C_k|\mathbf{x})$ directly or learning $p(\mathbf{x}|C_k)$ and then deriving the target, $P(C_k|\mathbf{x})$, from that (which requires an estimate of $P(C_k)$). Note also the significance of $P(C_k)$ which is the prior probability of class $k$, i.e. the probability that $\mathbf{x}$ will belong to $C_k$ before $\mathbf{x}$ is observed. If the prior probability of any class is high, then that class will have an increased $P(C_k|\mathbf{x})$ for any given input, $\mathbf{x}$.

Learning $P(C_k|\mathbf{x})$ directly for each class is sometimes referred to as a *discriminative function* based approach. This is due to that having $P(C_k|\mathbf{x})$ for each class is sufficient information to discriminate between them - e.g. choose the class with the highest $P(C_k|\mathbf{x})$. In the present case of only one class begin sufficient for classification, this term is somewhat less meaningful but can still be used to refer to the particular approach.

The mapping from input to output ($\mathbf{x}$ to $P(C_k|\mathbf{x})$) can be seen as a parameterized function:

$$P(C_k|\mathbf{x}) = P(C_k|\mathbf{x}, \theta) \tag{4.3}$$

with $\theta$ being the parameters. If $p(\mathbf{x}|C_k)$ is the target, then this can similarly be seen as:

$$p(\mathbf{x}|C_k) = p(\mathbf{x}|C_k, \theta) \tag{4.4}$$

Then, in practice, there are two different tasks to be undertaken: *inference* and *decision making.* The first of these is concerned with learning the parameters of the mapping from a *training* data set. This contains 'true' input-output pairs, i.e. for each input the correct output is known.

Decision making then corresponds to 4.3 and 4.4 where $\mathbf{x}$ is some new input, the parameters $\theta$ are those that have been found in the inference step, and $P(C_k|\mathbf{x})$ (using 4.2 in the case of 4.4) is then an estimate of the probability that $\mathbf{x}$ belongs to class $C_k$. This estimate may be referred to as a 'decision'. For these decision to be 'good' for never-before-seen data (i.e. $\mathbf{x}$ is not identical to any input in the training set, which will be typical for high-dimensional data), the system must have the ability to *generalize.* This means that the inference step is not about memorizing the training data set but to infer parameters that can be used to obtain correct outputs for new data. This requirement has many implications, one being that that the number of training data should preferably greatly exceed the number of parameters, i.e. the dimensionality of $\theta$.

## 4.1 Inference

The goal of training the system is to obtain a set of parameters $\theta$ that are able to generalize, i.e. produce correct outputs for new inputs.

To achieve this, the parameters are trained on the training data set that contains the necessary input-output pairs. During training, the parameters are progressively changed to reduce the *classification error.*

For this, some sort of *error function* is needed first of all to measure the current classification error which is a function of the network output $\mathbf{y}$ - representing $P(C_k|\mathbf{x})$, the input $\mathbf{x}$ and the known (correct) output or *target*, $\mathbf{t}$. Typically, the error function is chosen so that minimizing it lead to the *maximum likelihood* solution, i.e. the parameters are chosen so that the likelihood of the training data set is maximal.

On a side note, in what is sometimes referred to as "Bayesian" decision making, no specific choice of a single model is made. Instead, the uncertainty about the parameters is taken into account by including prior distributions on these and then integrating them out:

$$P(C_k|x) = \int_\theta P(C_k|x,\theta)p(\theta)d\theta \tag{4.5}$$

However, this is a completely different approach and is not treated further here.

## 4.2 Generalization and overfitting

If a system is too complex, meaning that too many parameters ($\theta$) are available, it is possible to learn too much, so to speak, from the training data set. In fact, with enough parameters, the mapping can actually be a memorization, and nothing has really been learnt. On the other hand, if the system is too simple (few parameters), it will not be able to learn the properties of the data that are necessary in order to be able to generalize the decisions to new data. This is sometimes referred to as the *bias and variance tradeoff*, for reasons discussed (at length) in [3]. The main reason for all this is that the data are stochastic, consisting of an underlying structure - that the system *should* learn - and some additional noise, that it should *not* learn.

There are numerous ways of reaching some sort of balance in this tradeoff. These include the 'forward selection' and 'backwards elimination' of classical statistics. There, each parameter is tested to see if it should be included or excluded, and the number of parameters is set in a (in some variants) somewhat heuristic manner.

Another way is to employ a second data set called a *validation set*. Then, several different mappings, increasingly complex, are trained on the same training set. After training, the error is then measured on the validation set. The system that performs best on the latter is then chosen as the best one. It is generally found that a certain complexity is optimal for the type of data and problem at hand.

Of course, this suffers from overfitting to the validation set, but since there is only one 'meta' or 'hyper' parameter (how many parameters to include in $\theta$), this is usually not a problem (although it pays to keep the issue in mind).

A principled way of exploring possible architectures using the validation set approach is to either *grow* a system or to *prune* one. In growing, the simplest possible network is selected to begin with, and this is then gradually made more complex. In pruning, the most complex network is selected first, and this is then gradually reduced in complexity.

For these ideas to be implemented, two things are needed. First, at method of either growing of pruning and second, a criterion for determining which network, i.e. level of complexity, is optimal.

The specific ways chosen to do this are covered in chapter 9 which deals with a particular type of mapping, namely 'linear networks', where selection of a system architecture (complexity) is necessary.

## 4.3 Thresholding

If a binary decision is required, $P(C_k|\mathbf{x})$ can be *thresholded* to produce one simply as

$$B_n = \begin{cases} 0 & \text{if } P(C_k|\mathbf{x}) < t, \\ 1 & \text{if } P(C_k|\mathbf{x}) \geq t \end{cases} \tag{4.6}$$

where $t$ is a threshold value, $0 \leq t \leq 1$.

## 4.4 Targets

The targets, $\mathbf{t}$, need not be binary (0/1). They may represent the probability that $\mathbf{x}$ belongs to a certain class ('soft targets'). This is relevant for the present systems that classify on a frame basis. Here, one classification decision is made for several time-domain samples, each having a corresponding true, binary $C_{VA}$ value. Therefore, a frame target is used instead that is the fraction of samples in that particular frame that belong to $C_{VA}$. This is not a crucial issue, since most frames are either all 1 or all 0 (due to being very short in time).

# Chapter 5

# Feature extraction

The term 'feature extraction' refers to the process of *transforming* the input signal in some way in order to obtain one or more of the possible goals described in the following. This can be seen as a mapping from the original - in this case time-domain - space to 'feature' space. A key fact to keep in mind here is that this transformation can not create any *new* information but it *can* 'get rid of' some (for the present classification problem) useless information that is part of the information content of the input signal. [6] provide an interesting method based on 'conditional mutual information' for designing efficient feature combinations in a principled way.

Feature extraction is the first step in the classification pathway. The raw signal $\mathbf{x}$ is transformed by the feature extraction in order to extract useful information etc. The raw signal may of course also be kept, which is the special case of the feature extraction being a unity transformation.

Several different features may be "simultaneously" extracted. Again, information is *discarded* - the trick is to discard information that is not helpful or relevant for the discriminator to do it's job.

The most relevant features to extract depend on the current context. Thus, a system with feedback from the decision maker to the feature extractor is conceivable, although not trivial to design. This would be a 'doubly adaptive' system, able to learn the mapping from features to output and also to deduce from the output what the current optimal features should be. In a hearing aid context, this could be applied by having a complete 'Environment Detection' system looking for say music- and speech-sensitive features if the output was currently consistent with that type of environment. However, this line of thought is not pursued further.

The only way to find out which features are (the most) useful is through experimentation, that is measuring the performance of the classifier.

## 5.1 Reduction of dimensionality

With $\mathbf{x}$ having an increasing number of dimensions, exponentially increasing numbers of data points are needed to 'cover' the input space, providing sufficient examples of input to learn from. For instance, with 36 dimensions, 3000 data points only amount to 1.25 points per dimension. This phenomenon is often referred to as the 'curse of dimensionality'.

This requirement for data is the reason why it is generally desired to work with as few dimensions as possible (e.g. reducing input dimensionality through principal component analysis or other techniques) and also to use learning structures that have strong *generalization* capabilities, so that they can perform well even if trained only on scarce data.

One purpose of feature extraction can be seen as the reduction in dimensionality of the raw input, $\mathbf{x}$.

Of course, this would also generally lead to an decrease in computational speed, which is always desirable.

## 5.2 Concentration of information

By utilizing prior knowledge, it is possible to extract features that are known to contain 'concentrated' information that is helpful for solving the problem at hand. In fact, the ideal feature is a one-dimensional signal that is identical to $P(C_k|\mathbf{x})$(!) - but this is bending the concepts. In practice, the features should contain as much relevant and as little non-relevant information as possible and the following classifier will use these to get to an estimate of $P(C_k|\mathbf{x})$.

## 5.3 Post-processing of features

There might be some benefit in applying some processing after the features have been extracted. In [26], all features are transformed by taking their logarithm. This is done to improve their spread but also to make them conform better with a normal distribution, which was necessary for that application. One consideration should always be that of normalization. Even though the raw input signal has been normalized, there is no guarantee for the scaling etc. of the resulting feature signals. In the present case, using the cross-correlations of filterbank outputs as features, these were not found to either so large or so small as to give numerical problems, so they were not 're-normalized'. Another consideration is the *relative* scaling between the features, but since the features in the present system were input to an adaptive system able to re-scale each feature appropriately itself, this was not an issue.

## 5.4 Derived Features

Infinitely many features can be derived from each feature extracted from the signal. Complexity depends on how high-level the derivation is. E.g. a peak detection algorithm on a spectrum can be quite costly.

## 5.5 Time-derivatives

Time-derivatives can be calculated for any feature that operates on windows over the signal. This is a fast operation and should probably be considered for all features.

Higher-order time derivatives can of course also be calculated, but at exponentially increasing computational cost.

## 5.6 Statistical moments

Taking the mean of a feature over (a certain length of) time is simply a smoothing operation.

The variance of a feature may be more discriminative than the feature itself or its mean (see [26]).

Covariance between features could possibly reveal something useful.

Higher-order moments could also be calculated, although in practice one would have to stop at say order 3.

## 5.7 Auto- and crosscorrelation

These are 2nd order statistic. They could be carried out on the the 'raw' input signal or on any other feature or set of features. The autocorrelation can be used to find periodicities and is sometimes used for pitch tracking.

Cross-correlation is also sometimes used as a 'grouping cue' (used to attribute low-level features to higher order objects, such as a sound source) in Computational Auditory Scene Analysis (CASA).

## 5.8 Specificity of features

Some features are explicitly designed to hide and remove characteristics of the signal that are not relevant to a specific task. For instance, according to [30], the MFCC (see appendix E) - by smoothing the time-frequency image in a certain way - hide and remove some signal characteristics that might be relevant for other tasks outside the speech domain.

Figure 5.1 shows a diagram of the basic classification systems (see also the previous chapter). From the input signal $\mathbf{x}$, some features $\mathbf{f}$ are extracted. These are then either input to a 'direct' classifier, yielding $P(C|\mathbf{f})$ (top of figure). They may also be given to a modeler (of the class-conditional probability distribution) and then also (indirectly) give $P(C|\mathbf{f})$. Learning the parameters (both the classifier ('CL' in the figure) and the modeler ('PM') are parameterized) is done iteratively, adapting the parameters throughthe use of error functions ('EF') and the correct probability of class membership ($\mathbf{t}$). The output might also be thresholded to produce a binary signal (not shown).

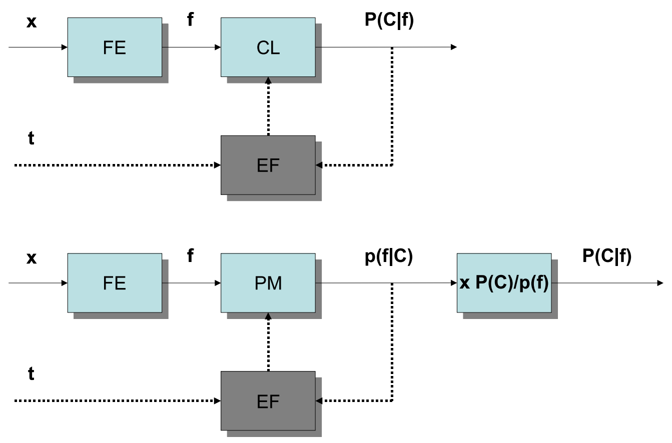

**Figure 5.1.** Top: direct classification. Bottom: classification through modelling of the class-conditional probability distribution. Dashed arrows are only used during learning (inference).

# Chapter 6

# Use of prior knowledge

A unifying trait of all approaches to solving classification problems is the application of prior knowledge about the problem domain. For instance, in speech detection, the knowledge about the characteristics of speech has given rise to all kinds of features that can be extracted from speech signals, that are designed to 'capture' such characteristics, in order to facilitate classification.

## 6.1 Selection of features

The feature extraction process seen as a transformation of the input signal clearly has great potential. One way of loooking at this is that a non-linear transformation might transform data that is not linearly separable into something that is. Usually, however, a 'battery' of features is chosen where each has some potential for capturing a particular characteristic of the target class.

Whatever features one would like to use for their known discriminative power prior, one must be careful of the amount on computation spent on extracting them. This should be compared with the "unsupervised extraction" alternative, namely giving the decision maker (say, a multi-layer perceptron) access to enough "raw" input to learn appropriate features itself. Such a decision maker, being highly flexible and trained to classify in the best manner possible, might learn to extract even stronger information than a time-consuming (in design- and extraction time) manually designed 'super' feature extraction.

So, as a general rule, feature extraction should be kept on the simple side. It must be remembered that the feature extraction must be performed even during decision making on each incoming data point $\mathbf{x}$.

## 6.2 Division into sub-classes

If the target class is known to be divisible into sub-classes, then there is a simple way of taking advantage of this knowledge. A separate classifier is simply designed for each sub-class, and these can be trained (inferred) separately. In decision making, the outputs of the classifiers can be compared, and the one with the highest $P(C_k|x)$ estimate can be chosen over the others (assuming the classes are mutually exclusive).

This concept is nicely applicable to the speech domain, where the voiced and unvoiced are distinct and mutually exclusive (speech is either voiced or unvoiced). Potential difficulty with these particular classes lies in that voiced speech is easily perceived as speech, while unvoiced is not - typically resembling simply noise when heard in isolation.

# Part II

# Methods

# Chapter 7

# Survey of Methods

## 7.1 Introduction

This chapter discusses and describes some of the options for designing a VAD. These options regard the area of probabilistic classification methods, but also the feature extraction step. Most VAD's can generally be described as being a combination of a certain choice of feature extraction method and a certain choice of classification method.

A good VAD can be designed by extracting 'strong' features, putting together a powerful learning system (e.g. a multi-layer perceptron or other non-linear methods) or a combination of both.

Each algorithm corresponds to a certain focus on and weighting of these areas. The majority of articles written on VAD and related topics (such as SNR estimation) are highly focused on a very particular feature or set of features. This features is then often used as input for rather simple machine-learning systems, or the parameters mapping input to output are simply manually tuned. An example of the latter is actually the ITU-T VAD (see [1]). Of course, many have used more principled and powerful classification methods.

## 7.2 Features

There are a great many suggestions for features to be extracted in the speech detection literature, some of which are reviewed very briefly here. Those features that are chosen to be used for the present system are described here, while the remainder are described in appendix E. A good introduction to speech signal processing is given in [21].

### 7.2.1 Filterbanks

The basic idea of filterbanks is similar to that of the Fourier transform and may indeed be implemented using the FFT (Fast Fourier Transform). The input signal $x$ is filtered by a set of filters (in parallel), giving a multivariate, transformed signal $\mathbf{f}$ (see figure 5.1).

Many filterbanks are biologically inspired, e.g. 'gammatone' filterbanks. They can be calculated by cascading low-pass filters with differing cut-off frequencies. The combined output gives a time-frequency image. Malcolm Slaney's "Auditory Toolbox" (`Matlab` code is available) contains many filterbank models of the human ear, so that various types of time-frequency images can be created.

### 7.2.2 Filterbank crosscorrelations

If the outputs of a filterbank are pair-wise cross-correlated, a derived feature-signal is obtained that may hold strongly discriminative information about the possible presence of speech. This idea is based on the observation of common frequency onset of speech (see chapter 3) and is also the basic concept for the OTI VAD (see [4]).

For the present system, the cross-correlation signals are squared in order to obtain a phase-independent signal.

### 7.2.3 Linear filterbank

The simplest filterbank is one where each filter has the same bandwidth and the filters are spaced linearly across some frequency range. However, for speech, the result is usually not very useful, containing little information about the presence of speech; see figure 7.1.

It is possible to adjust placement and bandwidth manually, but it is quicker and probably better to use the so-called 'mel-scale' instead. This is a non-linear frequency-scale (placement and bandwidth) and it has been used to model the human ear (which of course is rather good at detecting speech). The mel-scale can be found in [21], page 1223.

### 7.2.4 Mel-scale filterbank

The main idea with this scale is to achieve finer resolution at lower frequencies (where most of the speech energy is to be found) and coarser resolution at high frequencies. The filters range from 133 to 6565 Hz.

Figure 7.2 shows a mel-scale filterbank consisting of 9 filters, while figure 7.3 shows one made up of 18.

Figures 7.4 and 7.5 show the output from these filterbanks together with their (squared) cross-correlations. Clearly, they contain some relevant information. The latter figure also shows the effect of normalization (see 3.6). Even with this, the cross-correlation image for speech is distinctive.

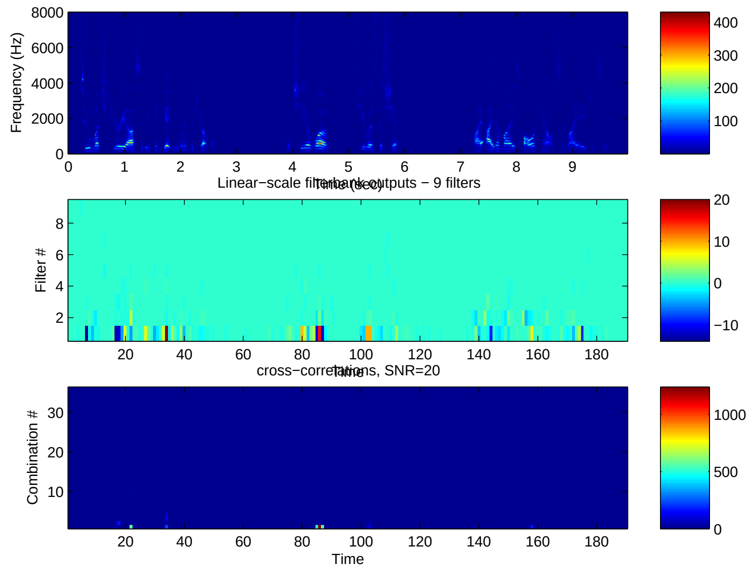

**Figure 7.1.** Linear filterbank outputs and crosscorrelations with 9 filters. Clean speech (spectrogram on top). It is clearly seen, that this 'naive' filterbank (middle) contains little discriminative information about speech presence, as do the cross-correlations (bottom).

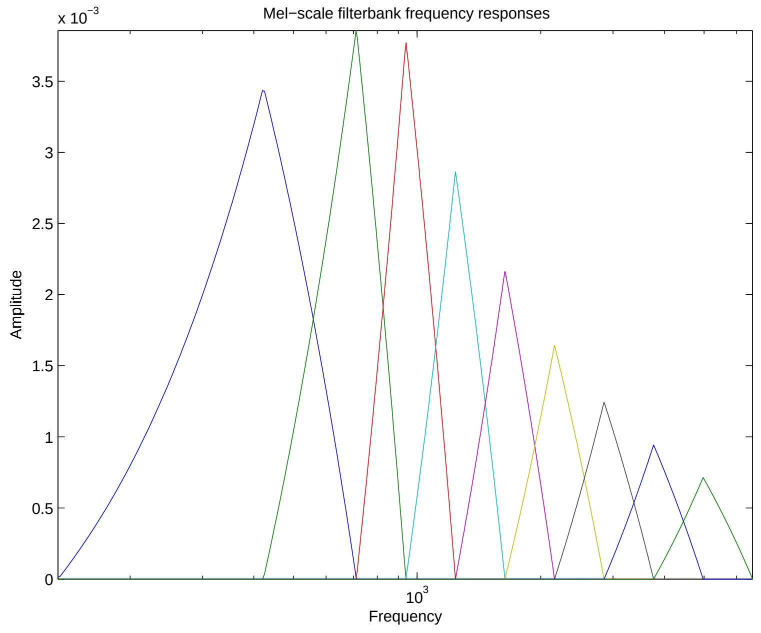

**Figure 7.2.** Mel-scale filterbank with 9 filters. Note the logarithmic x axis; it ranges from 0 to 8000 Hz.

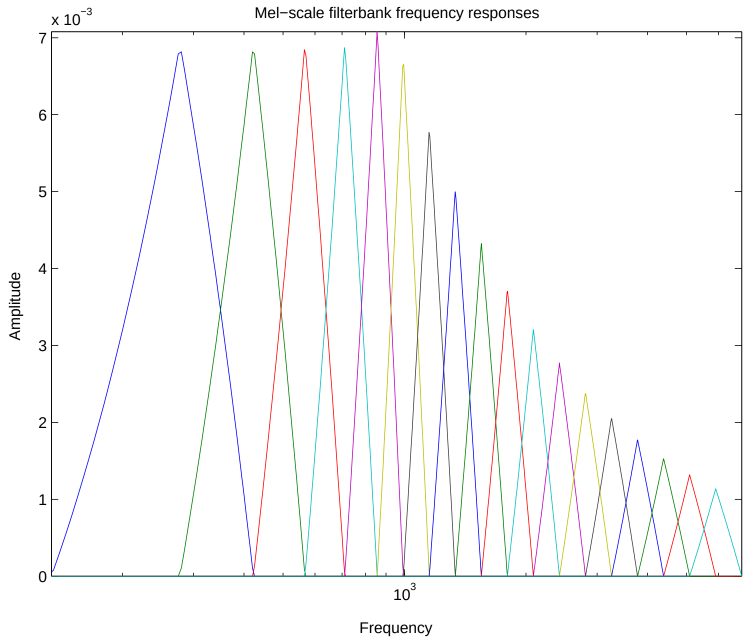

**Figure 7.3.** Mel-scale filterbank with 18 filters. Note the logarithmic x axis; it ranges from 0 to 8000 Hz.

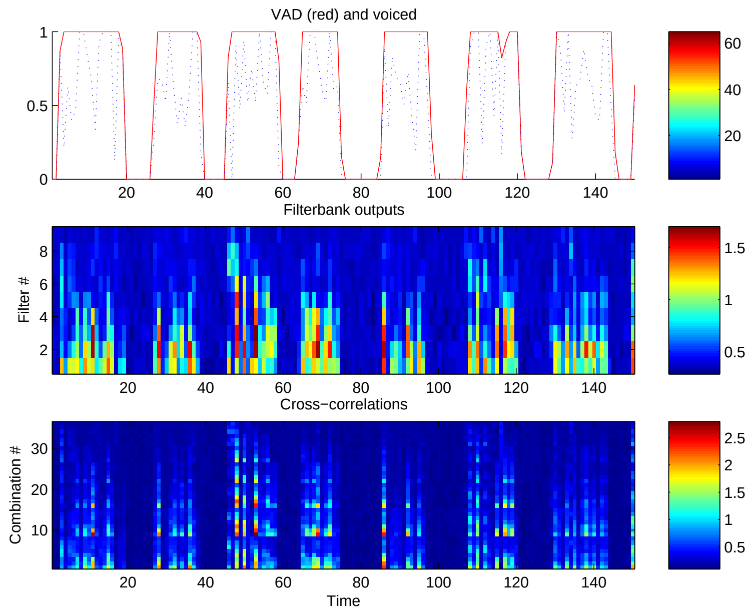

**Figure 7.4.** Mel-frequency filterbank outputs (middle) and crosscorrelations (bottom) with 9 filters. The true VAD signal is shown in the top part (dashed line represents the soft targets, see 4.4). White noise, SNR=10.

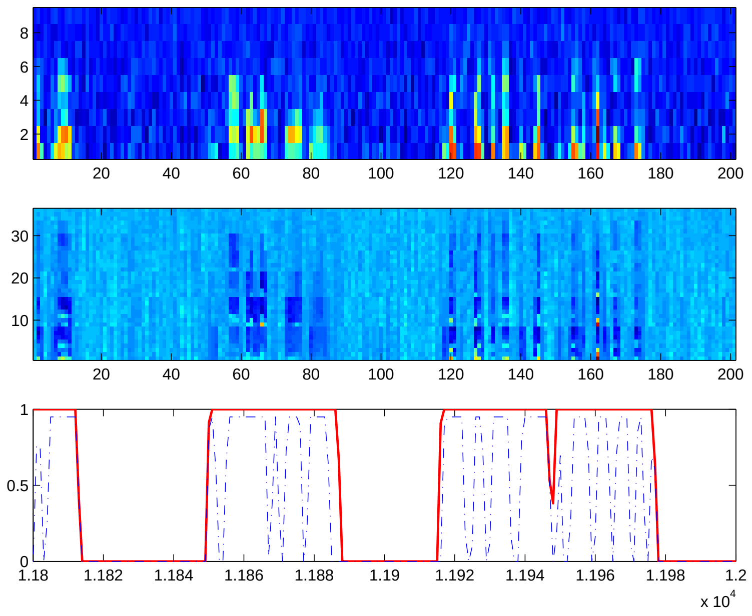

**Figure 7.5.** Mel-frequency filterbank outputs and crosscorrelations with 9 filters. Top: Filterbank outputs, middle: cross-correlations, bottom: true VAD signal. This is based on a normalized input signal. White noise, SNR=10 (normalized).

## 7.3 Choice of features

Given the choice between implementing a range of some of the 'promising' features that have been described in appendix E, or to rather invest the effort into the development of good learning systems, i.e. algorithms and models, and sticking with the proven mel-frequency filterbank cross-correlations feature set, it was decided to follow the latter course. Based on experience with the OTI VAD , it is known that cross-correlations between the power signals of filterbanks contain much information regarding the presence or absence of speech. This feature framework was then singled out for use in the VAD to be implemented.

It should be noted that much of the information content in many of the features mentioned in appendix E can still be expected to be contained in the cross-correlation features.

## 7.4 Classification methods

VAD algorithms are in general highly focused on feature extraction and selection and less on signal modelling and pattern recognition or machine learning theory and techniques. Many VAD algorithms described in the literature simply compare a (typically one-dimensional) feature with a threshold value to classify samples into speech or non-speech.

As mentioned in chapter 4, there are basically two approaches to building a probabilistic classifier, those modelling $P(C_{VA})$ directly and those modelling $p(\mathbf{x}|C_{VA})$.

There are of course many ways of implementing each of these. In the 'direct' section are for instance found linear neural networks and multi-layer perceptrons (see [3]).

In the distribution-modelling section is any conceivable signal model; 'radial-basis networks' are a famous example (see also [3]). State-based approaches (such as those employing hidden Markov Models) may fall in either category.

In the present case, a linear neural network is implemented. This is not as powerful as a non-linear learning system, but it is on the other hand robust and fast (in terms of computation time both during learning and decision-making).

Further, some work is done toward implementing several versions of a distribution-modelling classifier, using Independent Component Analysis (ICA) techniques. This is a radically different approach and might be expected to give very different performance results from the linear network. Together, the two approaches are intended to shed light on the speech detection problem each in their own way.

# Chapter 8

# The ITU-T VAD and the OTI VAD

Two VAD's are used as benchmarks to compare the VAD's developed in this project with. One is a telecommunications standard VAD, the other the VAD specified in patent document [4]. This chapter describes these two VAD's without going into detail.

## 8.1 The OTI VAD

This is the VAD described in patent document [4]. The basic concept behind this VAD belongs in the feature extraction domain.

The idea is an exploitation of the initial observation (see chapter 3) that speech exhibits a common onset within a certain frequency band. It is based on the use of a filterbank, and the feature extraction process ends up with a 1-dimensional feature, the 'synchronism' signal, which is then thresholded to produce the VAD signal.

See [4] for details on this VAD.

## 8.2 The ITU-T G.729 standard VAD

The International Telecommunications Union (ITU) has adopted a VAD algorithm to be used within its speech coding and transmission framework [1]. The main purpose of this is to allow for a technique known as 'discontinuous transmission' or DTX, where transmission is stopped if no speech is detected. This of course allows a much more efficient use of bandwidth. Together with other sub-systems, this VAD makes up the 'silence compression scheme' that is known as 'Annex B' of the ITU-T G.729 standard.

### 8.2.1 Method

The ITU-T VAD is heuristically designed and based on a range of energy-based and spectral features. It is a linear discriminator, where the discriminant functions have been chosen manually (in a simple way, see [1]), based on visual inspection of data points, instead of being learnt in a machine-learning sense.

The basic idea is to use two VAD algorithms. The first - and simpler - one is set to very high specificity, i.e. it does not detect unless there is a very high probability of voice activity. Assuming that the noise is stationary relative to the dynamics of the voice activity, the spectral features measured during noise-only periods can be used as a sort of noise model. This can then be used to make a more advanced VAD. If - in a period of uncertain voice activity - the current features diverge too much from the 'noise model' features, then this is taken to be caused by the presence of speech.

This concept is quite heuristic but works quite well in practice, and the ITU-T VAD is often used as a benchmark for new VAD algorithms (e.g. see [13]).

The algorithm is available over the internet as source code (c written for the linux platform) from ITU; see:

`http://www.itu.int/rec/recommendation.asp?`

`type=items&lang=e&parent=T-REC-G.729-199610-I!AnnB`

This code was modified to output the VAD flag and was compiled and run under Cygwin on a winXP PC.

This algorithm only outputs binary VAD decisions so it is not possible to obtain ROC curves, but instead points can be placed on the ROC plane (see 11.2).

The algorithm operates on frames of 10 ms length, which is very fast compared with the hearing-aid range (less than 200ms). But various heuristic smoothing techniques using previous frames are used to obtain a better VAD output. Due to these mechanisms, it is not possible to modify the code so as to extract any VAD output level for ROC curve generation.

## 8.3 Other VAD algorithms

Many different VAD algorithms have been built in the last many years.

The GSM (currently 'phase 2+') standard also uses a VAD to allow for discontinuous transmission. This VAD is somewhat simpler than the ITU-T VAD - for instance it does not really employ any noise model.

[29] use a quite principled approach to VAD, modelling the Discrete Fourier Transform coefficients of both the noise and the speech signals as normally and independently distributed. [24] develop a simple SNR estimation algorithm for situations with white noise and speech only.

Other principled and inspiring VAD algorithms can be found in [12], [30] and [13].

# Chapter 9

# Classification using linear neural networks

The 'linear neural network' is one of many available options for designing a probabilistic classifier that is based on and learns from data. A good description of this approach is given in chapter 3 of [3] (where the general structure is called a 'single layer network').

The central characteristic of this classifier is that it is based on the idea of a discriminant function, as described in chapter 4. It is a relatively fast method, both in terms of inference and decision making. It is therefore often a good idea to apply this method before more advanced and/or cumbersome methods. Also, there are several ways around the apparent limitations inherent in the 'linear' characteristic, such as using powerful features or combining output from several networks (which is still relatively fast).

The output of a linear network is a particular form of the parameterized expression in equation 4.3 that can be written as:

$$y = g(\mathbf{w}^T\mathbf{x} + w_0) \tag{9.1}$$

where $y$ is the estimated $P(C_{VA}|\mathbf{x})$. Thus, the output is some function on a linear combination of the input elements, plus an additional term, which is called the *bias*.

Repeating Bayes' rule for convenience

$$P(C_k|\mathbf{x}) = \frac{p(\mathbf{x}|C_k)p(C_k)}{\sum_j p(\mathbf{x}|C_j)p(C_j)} \tag{9.2}$$

and dividing by $p(\mathbf{x}|C_k)p(C_k)$ yields

$$P(C_k|\mathbf{x}) = \frac{1}{1 + \dfrac{\sum_j p(\mathbf{x}|C_j)p(C_j)}{p(\mathbf{x}|C_k)p(C_k)}} \tag{9.3}$$

Now, setting

$$a = \ln \frac{p(\mathbf{x}|C_k)p(C_k)}{\sum_j p(\mathbf{x}|C_j)p(C_j)} \tag{9.4}$$

(note the inversion of the fraction) then reveals the functional form:

$$P(C_k|\mathbf{x}) = \frac{1}{1 + exp(-a)}$$

so that the function $g(a)$ in equation 9.1 is

$$g(a) = \frac{1}{1 + \exp(-a)}$$

This allows the output of the network to be interpreted as the posterior probability of class membership. This is very nice in itself, but also allows training to be done using 'natural' training data, namely either 0's and $1's$ (in the case of known 'true' class membership) or continuous values between 0 and 1 in the case of uncertain training data or fractions in the case of frames (see 4.4).

The linearity of the network lies in that $a$ in 9.4 per 9.1 is made to be a linear function of the input. There is no guarantee that 9.4 as a function of $\mathbf{x}$ is linear, and thus the linear network may not be able to learn a good mapping from input to output. The architecture of the network is shown in figure 9.1. Note that the bias weight allows the output probability to be non-zero even if the input vector is zero, which is of course a good ability as there is no prior assumption that a zero input should give a zero output.

## 9.1 Preparation of training data

Time-domain input is normalized prior to feature extraction. This means that the squared cross-correlation features that extracted from the time-domain signal will also be normalized, although on another scale. This was found to be close to zero-mean and unit variance.

As all training sets are based on time-domain signals normalized in the same way and are selected as subsets of the randomly permuted total set of data, no further normalization is necessary.

As classification of frames is independent of other frames, frames that make up the training sets can be permutated. This can also be beneficial for training speed.

## 9.2 Training

As described in chapter 4, the network must learn to map from input to output with the ability to generalize. The weights $\mathbf{w}$ correspond to the $\theta$. To train the network, an error function needs to be chosen. An algorithm to change the

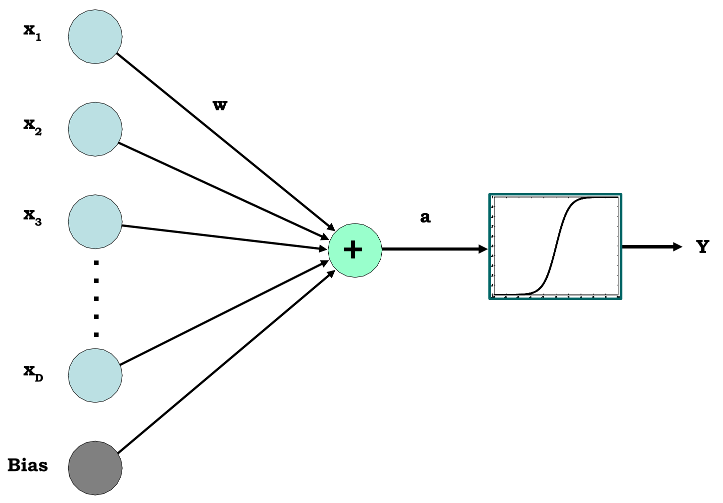

**Figure 9.1.** Linear network with logistic activation function and one output

weights in some way to reduce the error must also be designed. Finally, the *number* of weights is a parameter in itself and must be chosen. Alternatively, the structure of the network (including the number of weights) may be based on the data and adapted to reduce generalization error. One way of doing this is called 'pruning' and is described later on.

### 9.2.1 Error function

There are of course infinitely many possible error functions. The one actually used is usually chosen for reasons of simplicity and mathematical tractability, but mainly because it is highly appropriate for classification problems.

For regression-type learning, the quadratic error function is typically used. This is defined as

$$E = \sum_{n=1}^{N} (y_n - t_n)^2 \tag{9.5}$$

where N is the size of the data set, $y_n$ are the outputs and $t_n$ the corresponding targets.

For classification problems, another error function is frequently used that is called the *cross-entropic* error function. This is defined as

$$E = -\sum_{n} \{t_n \ln y_n^n + (1 - t_n) \ln(1 - y_n^n)\} \tag{9.6}$$

(Partial) reasoning and derivation for this is given in [3], pp. 230-231.

Looking past the intricacies of equation 9.6, the basic reason for using it over the quadratic variant is that it 'punishes' errors in a more intuitively appropriate way. It punishes small errors lightly, but if the output is opposite of the target, the error is very high. 'Opposite' here means that the output is near 1 when the target is near 0 and vice versa.

This error function is depicted in figure 9.2 and a comparison with the quadratic error function can be seen in figure 9.3, showing why it might it is an appropriate error function to use for classification.

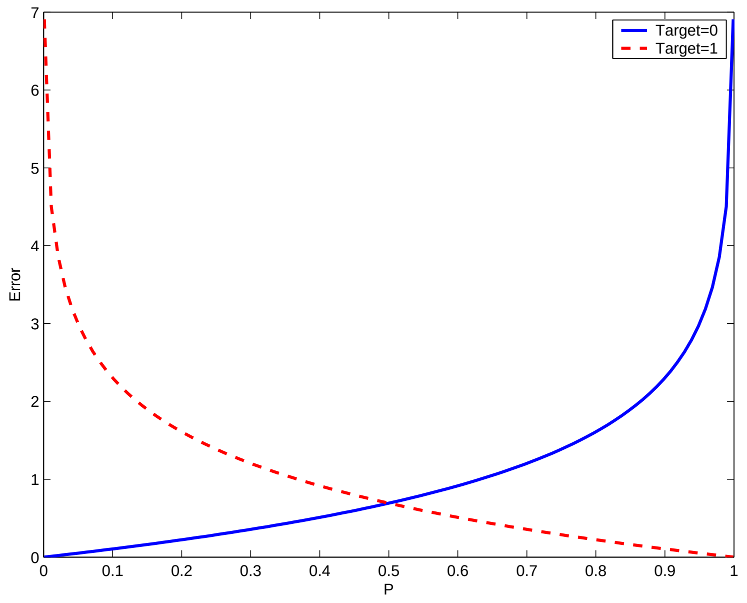

**Figure 9.2.** Cross entropic error function. The x axis is the output (estimated probability of class membership). Note the sharp increase in error as the output moves opposite to the target.

### 9.2.2 Training algorithm

Having chosen an appropriate error function, it is next necessary to derive formulas for changing the network parameters (i.e. weights) so as to reduce the error.

The error corresponding to the network output can be written as a function of the output

$$E = f(y) = f(g(a)) \tag{9.7}$$

where y is the output of the network.

Differentiating this composite function with respect to an individual weight, applying the chain rule of differentiation, yields

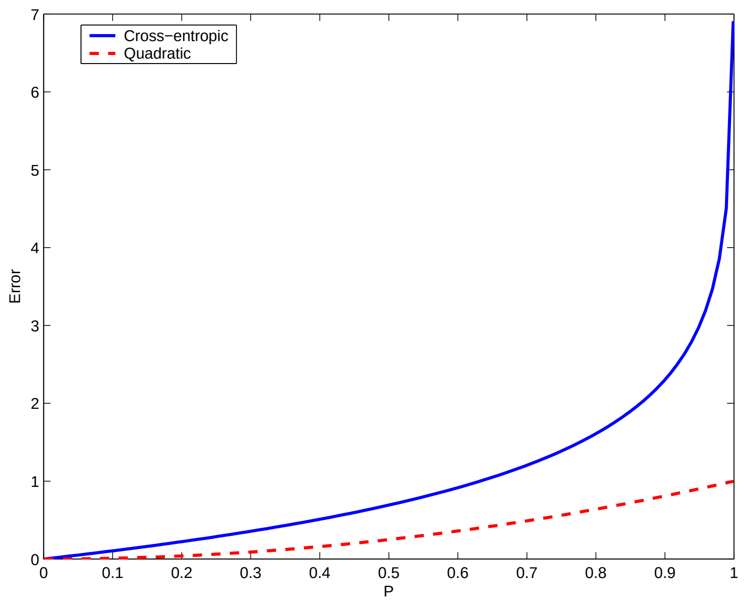

**Figure 9.3.** Cross entropic error function compared with the quadratic error function. The former penalizes highly incorrect classification much more severely, and the quadratic error does not distinguish much between small and large errors of estimated probabilities.

$$\frac{\partial E}{\partial w_i} = \frac{\partial E}{\partial a}\frac{\partial a}{\partial w_i} \tag{9.8}$$

where $a$ is the input to the logistic function

$$a = \sum_{i=1}^{N} w_i x_i + w_0$$

(note that in the linear network, there is no dependence between individual weights, as there is in a multi-layer perceptron.)

From the last expression it is clear that

$$\frac{\partial a}{\partial w_i} = x_i$$

Similarly, $\frac{\partial E}{\partial a}$ can be written in the composite form

$$\frac{\partial E}{\partial a} = \frac{\partial E}{\partial y}\frac{\partial y}{\partial a} = g'(a)\frac{\partial E}{\partial y} \tag{9.9}$$

With the logistic function, $g(a) = \frac{1}{1+\exp(-a)}$ so that

$$\begin{aligned}
\frac{\partial g}{\partial a} &= \frac{-\frac{\partial}{\partial a}(1+\exp(-a))}{(1+\exp(-a))^2} \\
&= \frac{1}{(1+\exp(-a))}\frac{\exp(-a)+1-1}{(1+\exp(-a))} \\
&= \frac{1}{(1+\exp(-a))}(1-\frac{1}{(1+\exp(-a))}) \\
&= g(a)(1-g(a))
\end{aligned}$$

which incidentally is very nice with respect to computational cost, in that the derivative can be calculated directly from the output.

Now, using the cross-entropic error function (equation 9.6) gives

$$\frac{\partial E}{\partial y} = \frac{y-t}{y(1-y)}$$

which together with 9.9 yields:

$$\frac{\partial E}{\partial a} = y(1-y)\frac{y-t}{y(1-y)} = y-t \tag{9.10}$$

so that finally,

$$\frac{\partial E}{\partial w_i} = x_i(y-t) \tag{9.11}$$

This simple result stems from the combination of the logistic non-linearity with the cross-entropic error function.

### 9.2.3 The conjugate gradient method

The simplest way of training the network is using 9.8 in a *steepest descent* algorithm, updating the weights according to;

$$w_i = w_i - \alpha \cdot \frac{\partial E}{\partial w_i} = w_i - \alpha \cdot x_i(y-t)$$

However, this first-order approach may lead to very slow learning. Successive gradient descent steps may converge only very slowly, and training can tend to oscillate rather than converge. Luckily, much better alternatives are possible.

Here, a *conjugate gradient* algorithm is employed. Details can be found in chapter 7 of [3], and only an overview and the actual algorithm will be given here.

This solution is based on choosing successive search directions such that these are orthogonal to *all* previous search directions, the limit to the meaning of 'all' in this context being the number of dimensions of **w**. The derivation of the formulas for these mutually orthogonal search directions is quite involved, and can be found in [3].

The key concept in the conjugate gradient method is that the second-order gradient information (contained in the 'Hessian' matrix) is given from the last weight-update direction and the current and last gradient vectors. This means that the directions can be updated without explicit knowledge of the Hessian, which is a tremendous saving in computational cost (certainly with a non-linear, multi-layer network, but also in the linear case).

The search direction at step $j+1$ can be found as

$$\mathbf{d}_{j+1} = -\mathbf{g}_{j+1} + \beta_j \mathbf{d}_j \tag{9.12}$$

where $\mathbf{g}_j$ is the gradient at step $j$ and $\beta$ must be updated according to one of 3 alternative formulas. The elements of $\mathbf{d}$ are the directions for each dimension of $\mathbf{w}$. The (Fletcher-Reeves (FR)) version:

$$\beta_j = \frac{\mathbf{g}_{j+1}^T \mathbf{g}_{j+1}}{\mathbf{g_j^T g_j}} \tag{9.13}$$

is arbitrarily chosen, as no significant difference in convergence times was found over several attempts with different sets of artificial data using the other 2 alternatives ('Polak-Ribiere' and 'Hestenes-Stiefel'). The 3 different version of the update formula for $\beta$ are exactly equivalent in the case of a quadratic error function; for non-quadratic error functions, they can give different results, but this was not found to any marked degree for the data used in this project.

It was found (for artificial data) that learning using conjugate gradients was several times faster than steepest descent learning.

### 9.2.4 Line search

To speed learning further, a simple search may be done along the direction found by the conjugate gradient method. For non-quadratic error functions (as here), this direction may actually not be towards lower cost, and the line search then also ensure that the weights are changed in the correct direction.

A simple line search algorithm was implemented (see `simple_linesearch.m`) that gives an appropriate step size for minimizing the error in a given direction.

### 9.2.5 Batch training

All training is done on the whole training data set, i.e. for each step of adapting the weights, the gradient etc. is calculated for the whole training set.

## 9.3 Overall algorithm

Algorithm 9.3 shows how a linear network is trained, i.e. the weights are inferred, based on training data.

Here, $x_i^n$ stands for the $i$'th input for the $n$'th example and $\mathbf{g}_i^j$ is the gradient for the $i$'th weight at the $j$'th iteration (same terminology for $\mathbf{d}_i^j$).

**Algorithm 1** InferWeights($D, T, j_{max}$) - Infer linear network weights given training input, $D$ and targets $T$, using $j_{max}$ iterations

```
Set **w** to be random (uniformly distributed from -1 to 1)
$j = 1$
{Calculate output}
$\forall n : y_n = g(\mathbf{w}^T \mathbf{x}_n + w_0)$
{First direction is simply the gradient}
$\forall i : \mathbf{d}_i^j = -\mathbf{g}_i^j = \sum_n \frac{\partial E_n}{\partial w_i} = \sum_n x_i^n (y_n - t_n)$
{Search direction for optimal step size}
$\alpha = \text{LineSearch}(\mathbf{d}_i^j)$
**repeat**
    {Update weights}
    $\mathbf{w}_i = \mathbf{w}_i + \alpha \mathbf{d}_i^j$
    {Calculate output with new weights}
    $\forall n : y_n = g(\mathbf{w}^T \mathbf{x}_n + w_0)$
    {Calculate next error gradient}
    $\forall i : \mathbf{g}_i^{j+1} = \sum_n \frac{\partial E_n}{\partial w_i} = \sum_n x_i^n (y_n - t_n)$
    {Update $\beta$}
    $\beta_j = \frac{\mathbf{g}_{j+1}^T \mathbf{g}_{j+1}}{\mathbf{g_j^T g_j}}$
    {Calculate next search direction}
    $\mathbf{d}_{j+1} = -\mathbf{g}_{j+1} + \beta_j \mathbf{d}_j$
    $j \leftarrow j + 1$
**until** $j = j_{max}$
```

This algorithm is implemented by the functions `linlog_init.m` (initialization) and `linlog_train_cg.m` (training). The latter calls `linlog_forward.m` (calculating the output for a given input) and `linlog_gradient.m` (calculates the gradient) and `simple_linesearch.m` (`LineSearch`) (all in folder ANN on the CD[1]).

### 9.3.1 Initializing parameters

The simplest way of choosing initial parameters (weights) is to set them to random values around zero. But since testing the performance of any network is relatively fast compared to actually training one, one may instead initially test several random networks and pick the best one of these to train. This was done in all experiments to speed learning.

Figure 9.4 shows a small test of the linear network classifier. Given some normally distributed data (seen to be linearly separable), it finds the correct 'line of demarcation' between the classes.

Figure 9.5 shows a section of the corresponding estimated probabilities of class membership, $P(C|\mathbf{x})$.

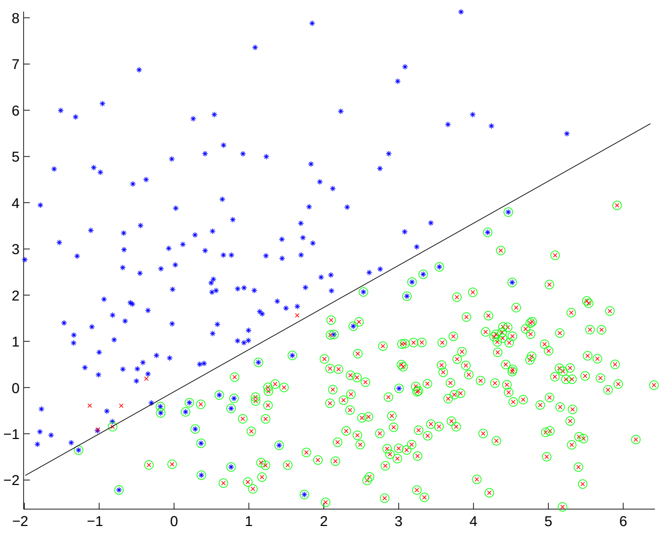

**Figure 9.4.** Classification of artificial data. Nearly all data are classified correctly. The line is the one found by the linear network after training, separating the two classes (stars and crosses are the true classes; the circled points are those estimated to belong to the 'cross' class by the network).

[1]Also available from the author (dj@imm.dtu.dk)

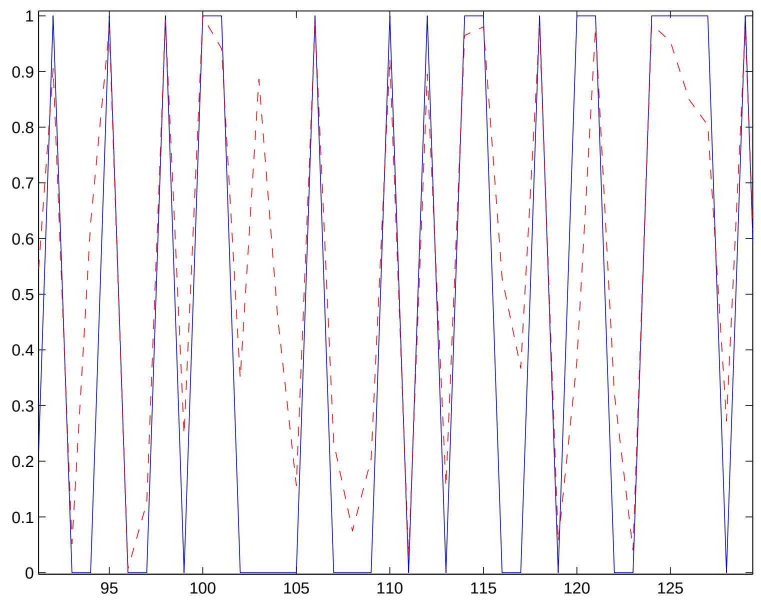

**Figure 9.5.** A section of true (blue) and estimated (red,dashed) probabilities of class membership, $P(C|\mathbf{x})$ after convergence, using artificial data. With one exception, the estimates are quite accurate.

## 9.4 Pruning

As mentioned in chapter 4, generalization performance can be improved by the process of *pruning*. An additional - practical - reason for pruning is that the smaller the network can get, the faster it will be. Compared with another classical technique with the same purpose known as *weight decay*, where all the weights are kept, this is an advantage for a system for which computational speed is an important parameter, as is the case with the hearing-aid target.

Further, *interpretation* of the relative importance of the different cross-correlations between frequencies is of interest in itself.

It is quite conceivable that with an input vector size of only 36, no pruning can be done without *loosing* generalization performance. The network may be so small already that any further reduction can only lead to reduced performance. Pruning has been used in other cases to reduce the number of parameters in systems with an initial size of several thousand parameters, as reported by [3]. So the present case must be said to be a very small scale of pruning.

Still, even with this 'small' scale, exhaustive search of all possible network structures is not possible. The number of possible architectures can be expressed as

$$N = \sum_{f=1}^{F} \binom{F}{f} \tag{9.14}$$

where $F$ is the number of features.

With 9 features, this is 511 possible systems and with 36 it is more than $6 \cdot 10^{10}$.

Pruning is one solution to the problem of searching this architecture space. The idea is to start with the most complex network, using all features as inputs, and train this network until convergence. Then, one weight is cut at a time according to relative importance, and retraining is done after each cut. This is repeated until say all the weights have been cut (possibly except for the bias weight). In this way, only $F$ networks will be created. It is obvious that the *method* that the weights to be cut are chosen is critical for such an enormous reduction to make sense and still find 'good' networks (i.e. close to the performance of the best network that could be found using exhaustive search).

It should be mentioned that pruning has one advantage over another typical method of regularization[2], namely weight decay[3]: the network size is reduced, leading to a smaller and faster system.

How much re-training to apply after each cut is one parameter of this process and this is typically heuristically determined, but reasonable settings can be determined with some experimentation. The most important choice however is the way in which the importance of the weights is estimated, as this will primarily determine which architectures will be created and thus how the 'architecture space' is searched.

The measure of importance for a weight is termed 'saliency'. A natural measure for this is the change in the error function resulting from a small change in the weight. Changing the weights $w_i$ by $\delta w - i$, the change in error can be written as a Taylor expansion as

$$\delta E = \sum_i \frac{\delta E}{\delta w_i} \, \delta w_i + \frac{1}{2} \sum_i \sum_j H_{ij} \delta w_i \delta w_j + O(\delta w^3) \tag{9.15}$$

where $H$ is the Hessian matrix with elements

$$H_{ij} = \frac{\partial E}{\partial w_i w_j}$$

([3], (9.66)).

Assuming that training has converged, the first term in 9.15 vanishes. Further simplification can be had by assuming that $H$ is diagonal:

$$\delta E = \frac{1}{2} \sum_i H_{ii} \delta w_i^2 \tag{9.16}$$

With this approximation, the saliency of each weight $s_i$ is

$$s_i = H_{ii} w_i^2 / 2$$

[2] A term used to denote methods to limit the complexity of networks

[3] which is adding to the error functions for large weights, also restraining complexity

Now setting the i'th weight to zero will approximately change the error by the saliency.

This leads to a pruning algorithm called 'Optimal Brain Damage' (OBD) - see [3], page 361. The version used in this project cuts one weight at a time. It is implemented by the `Matlab` scripts `prune.m` (saliency) and `melfb36_prune.m` (keeping track of what is cut and what is not).

If the full Hessian is used instead of the approximated diagonal one, the resulting algorithm is called 'Optimal Brain Surgeon'. On paper, it should perform better, but in practice, it suffers from matrix inversion and other problems so that OBD often performs better in practice.

It is important to train to completion (convergence) initially, and re-training should be of a similar quality, even though typically fewer re-training steps are needed.

### 9.4.1 The Hessian matrix of a linear network

The Hessian is defined as:

$$H_{ij} = \frac{\partial E}{\partial w_i w_j} \tag{9.17}$$

i.e. it contains the second-order derivatives of the error function with respect to the weights that are the learning parameters of the network.

The Hessian matrix is needed for the Optimal Brain Damage algorithm, and since no derivation of the Hessian matrix for a single-layer (linear) network is given in [3], it is briefly given here.

First off, it may be noted that even though the network is linear, the Hessian matrix is not diagonal. This is due to the non-linear (here: logistic) function that is used. However, the Optimal Brain Damage algorithm is based on the assumption that the Hessian *is* diagonal, so this assumption is carried through here.

The error differentiated with respect to the weights has already been found in equation 9.8. Differentiating again yields

$$\begin{aligned}\frac{\partial (y-t)x_i}{\partial w_i} &= \frac{\partial (yx_i)}{\partial w_i} \\ &= x_i \frac{\partial y}{\partial w_i}\end{aligned}$$

Using previous derivations,

$$\begin{aligned}\frac{\partial y}{\partial w_i} &= \frac{\partial y}{\partial a}\frac{\partial a}{\partial w_i} \\ &= g(a)(1-g(a)x_i\end{aligned}$$

so that finally

$$\frac{\partial E}{\partial^2 w_i} = g(a)(1 - g(a))x_i^2 \tag{9.18}$$

This simple result is the result of assuming a diagonal Hessian.

### 9.4.2 Practical issues

Each time a pruning 'run' is undertaken (as per the OBD algorithm), a different optimal weight set will be found. This is because the OBD is an algorithm that finds a local minimum in 'pruning space', depending on the initial weights, which are random. This actually makes it quite difficult to use OBD to choose a final, single network as the 'best' network; see 12.4.

## 9.5 Division of input space

It is possible to train separate networks for each region of input/feature space. This is strictly not 'Mixtures of experts' ([3]), since this term is usually reserved for the case when a suitable division of input space is *learnt* and not chosen manually. But the idea is very similar.

The obvious division in this context is between voiced and unvoiced speech, as these are known to be qualitatively different.

In the inference phase, one network is trained on examples belonging to one of the regions. In the decision making use, the same data is presented to both networks. Since signals belonging to $C_{VA}$ are *either* voiced *or* unvoiced, the *maximum* of $P(C_{voiced}|x)$ and $P(C_{unvoiced}|x)$ must be the appropriate estimate of $P(C_{VA}|x)$. The intuitive approach of simply averaging over the outputs of the two networks and setting $P(C_{VA}|x)$ to this is proven to be inferior by testing the two approaches (not shown).

# Chapter 10

# Classification using Independent Component Analysis

This chapter describes the Independent Component Analysis (ICA) methods that have been developed for use in speech detection algorithms. A brief overview of ICA is first given, including how the model can be inferred from the data.

## 10.1 Overview

Independent Component Analysis (ICA) in its basic form is concerned with a particular form of modelling signals.

This can be written as

$$\mathbf{x} = \mathbf{A}\mathbf{s} \tag{10.1}$$

where $\mathbf{x} = [x_1, x_2, ..., x_M]^T$ is the (multivariate) mixed signal, $\mathbf{A}_{[M \times M]}$ is the 'mixing matrix' and $\mathbf{s} = [s_1, s_2, ..., s_M]^T$ are the source (also multivarate) signals. In this form, no noise is assumed and the mixing matrix is quadratic. In the time domain, $\mathbf{x}$ and $\mathbf{s}$ might be seen as $\mathbf{x}_n$ and $\mathbf{s}_n$, i.e. the mixed signal and sources at time $n$. The mixing matrix $\mathbf{A}$ is fixed (although it must typically be learnt first in an inference phase). Note also that there is no noise in this model.

The key idea then is the assumption that the elements of $\mathbf{s}$ are statistically independent, i.e. each source $s_i$ is independently distributed from all the other sources.

There are a number of other assumptions usually made for the basic ICA model, and many extensions and variations of the basic model have been developed. A good introduction can be found in [2] while [15] discusses some extensions.

## 10.2 Signal separation with ICA

Typical uses of ICA is to retrieve the sources $\mathbf{s}$ given *only* the mixed signal, $\mathbf{x}$. This is possible due to the assumption of independence on the sources (which must of course be fulfilled in practice!). 'Blind' separation is the special case where there is no available knowledge except for the independence of the sources. For speech enhancement or removal of noise from a noisy speech signal, more than one channel recording of the mixed signal is necessary - the dimensionality of the recorded signal $\mathbf{x}$ must equal (or exceed) that of the sources, $\mathbf{s}$[1].

With only 1 channel available and several sources, separation without further assumptions or knowledge is not possible. Stronger assumptions are then needed than just that the sources are independent. In fact, it is necessary to incorporate any and all prior knowledge that is available about each individual source, the mixing matrix and so on. Only then is it theoretically possible to separate 2 or more sources that have been mixed into a single channel. The quality of the separation depends directly on how well or strongly the prior knowledge is expressed by the signal models and how much knowledge has been put in.

However, this is very hard to achieve, as might be expected. Jang and Lee ([8] have developed an ICA based method for doing single source audio separation (also working with speech data), but the results (available online) are unfortunately quite unconvincing. Roweis ([23]) uses another, simplified approach, based on the use of hidden Markov models, which could can be somewhat successful and can be improved, but that would be a major undertaking.

Of course, if it *were* possible to separate the speech and whatever noise is present in the single-channel recording available, then the VAD classification problem would be practically already solved: the separated - now 'clean' - speech signal could simply be given as input to almost any simple VAD algorithm which would have no problems classifying it correctly. No other processing (except for maybe some rudimentary feature extraction) would be necessary. And naturally, the cleaned-up signal would be a nice bonus(!).

Lots of knowledge *does* exist about speech (and other audio signals), so one may foresee better results in the single-channel separation field in the future.

## 10.3 Classification with ICA

Here, ICA is used for a different purpose, namely to model the class-conditional probability distributions of different signal classes, the $P(\mathbf{x}|C_k)$. Instead of trying to separate a mixed, single-channel signal, the ICA model may be used to design a classifier directly instead.

[1]Although with extended ICA algorithms, $\mathbf{x}$ may be only 2-dimensional and higher-dimensional sources can still be retrieved, given enough data

This is done by having each class $C_k$ correspond to a mixing matrix, $\mathbf{A}_k$ and a source distribution, $p(s_m)$ for each of the M sources ($m = 1, 2...M$). Then for each class,

$$p(\mathbf{x}|C_k) \equiv p(x|\mathbf{A}_k, \mathbf{s}_k) \tag{10.2}$$

Classification then simple requires applying Bayes' rule (equation 4.2):

$$P(C_k|\mathbf{x}) = \frac{p(\mathbf{x}|C_k)P(C_k)}{\sum_j p(\mathbf{x}|C_j)P(C_j)} = \frac{p(x|\mathbf{A}_k, \mathbf{s}_k)P(C_k)}{\sum_j p(x|\mathbf{A}_j, \mathbf{s}_j)P(C_j)} \tag{10.3}$$

$\mathbf{A}_k$ is assumed to be known, but not the $\mathbf{s}_k$. The latter are therefore marginalized:

$$p(\mathbf{x}|\mathbf{A}_k) = \int p(\mathbf{x}|\mathbf{A}_k, \mathbf{s})p(\mathbf{s}_k)d\mathbf{s}_k \tag{10.4}$$

## 10.4 Applying the ICA model to a one-dimensional signal

In the above, $\mathbf{x}$ is multivariate (of dimension $M$). However, the input signal to the VAD is a one-dimensional, time-domain signal. In order to use the multivariate ICA model on this signal, $\mathbf{x}$ is 'filled' with length $M$ *sections* of this signal. The concept is that each segment of the time-domain signal can be modelled as a mixture of $M$ independent sources, the mixing being determined by the elements of $\mathbf{A}$ (see [8]). To better understand the meaning of the latter, equation 10.1 can be rewritten as

$$\mathbf{x} = \mathbf{A}\mathbf{s} = \mathbf{A}_{\cdot 1}s_1 + \mathbf{A}_{\cdot 2}s_2 + ... + \mathbf{A}_{\cdot M}s_M \tag{10.5}$$

where $\mathbf{A}_{\cdot i}$ represents the $i$'th column of $\mathbf{A}$. So the columns of $\mathbf{A}$ can be thought of as 'building blocks' that are put together by the elements of $\mathbf{s}$ to make up $\mathbf{x}$.

Now, since each 'slice' of $\mathbf{x}$ ($M$ following samples) is modelled as a weighted sum of $M$ different segments, each also of length $M$, the result typically is that the elements of $\mathbf{s}$ are very sparsely distributed. This means that most elements will be close to zero and only a few will be 'on' for each segment. One might say that there is a high level of redundance in $\mathbf{A}$.

ICA is expected to be a powerful model for the $p(\mathbf{x}|C_k)$, since the basis functions describe high-order time-domain statistics of the signals of each class. Together with the modelling of the sparse distributions of the sources, this should contain a much discriminative information about the classes.

## 10.5 ICA features

One choice is between modelling $P(\mathbf{x}|C_k)$ directly or first transforming $\mathbf{x}$ by extracting some features, and modelling their distribution instead ($P(\mathbf{f}|C_k)$), see figure 5.1. Given the initial observations about the distinct spectral patterns of speech (see chapter 3), it was indeed attempted also to learn basis functions ('basis images') from the spectral domain. These were found to actually be images of horizontal lines at different angles, as expected.

However, the basis functions found when training on time-domain data also show highly speech-like characteristics (see chapter 12), so it was decided to model the distribution of the un-transformed, time-domain signals.

Amongst many other alternatives, the obvious one would be the filter-bank cross-correlations uses as features for the linear network (chapter 9). This was not undertaken due to time constraints, but would certainly be an interesting next step.

## 10.6 Choosing basis function length

Looking at samples of speech, it is seen that it can be described as being composed of short, characteristic time-domain segments. A few of these typically make up each phoneme. An example of two such pieces is shown in figure 10.1.

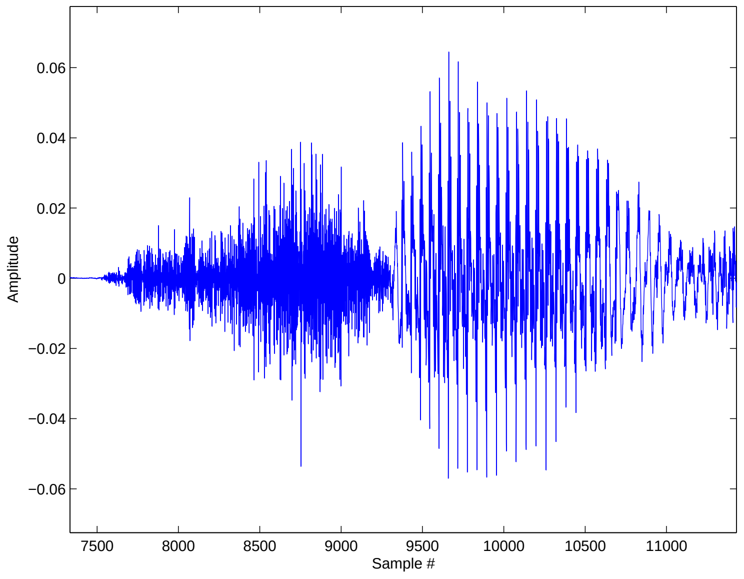

**Figure 10.1.** A section of unvoiced (first part) and voiced (last part) speech (the voiced segment is very high frequency).

Based on such inspections, a few different lengths were tried. It is expected that some basis functions should represent onsets of speech segments, others offsets, and others again the 'meat' of the speech segments (see figure 10.2). 128 samples (at 16kHz) was found to be consistent with these considerations, but learning 128 basis functions each of length 128 samples was found to be excessively time-consuming, so $M = 64$ had to be settled on instead. Of course, this

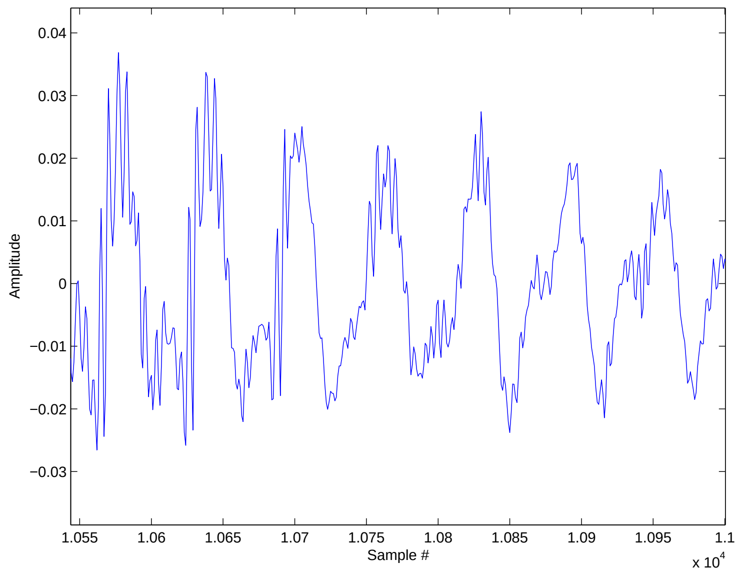

**Figure 10.2.** An example a repeating structure in a section of speech.

is only $64/16.000 = 4$ milliseconds, which is a *very* short time span. However, the results even with such short functions are encouraging, even if widening this time span is a natural next step for improving this approach. It would for instance be possible to reduce dimensionality through Principal Component Analysis, thus still operating with $M = 64$, but having this represent longer sections of the speech signal (see [14]).

Due to the constraint on $\mathbf{A}$ to be square, reducing the length of basis function also the number of functions. So $M = 64$ is actually a quadruple reduction in information compared to $M = 128$, so to speak. However, due to the little time available, it was decided to use $M = 64$ for the ICA algorithms. Figures (XX and XX) show the basis functions found for $M = 64$ and $M = 128$ respectively. They are seen to contain much the same type of information, and $M = 64$ is thus not a catastrophic reduction per se.

## 10.7 Modelling the source distributions

Given equation 10.4, the distribution of $\mathbf{s}$ is clearly a crucial piece in the model. One way of modelling them is to use a parameterized form,

$$p(\mathbf{s}) \equiv p(\mathbf{s}|\theta)$$

whose parameters must be inferred (learned) from data (examples of $\mathbf{s}$).

Since the source signals are expected to be very sparsely distributed, the *generalized Gaussian* distribution is a good choice for modelling these distributions;

see [17] for details.

### 10.7.1 The generalized Gaussian distribution

This distribution can be written as

$$p(x) \propto \exp(-\frac{1}{2}|x|^q$$

which can also be expressed in another form as

$$p(x|\mu,\sigma,\beta) \quad = \quad \frac{\omega(\beta)}{\sigma}\exp -c(\beta)|\frac{x-\mu}{\sigma}|^{2/(1+\beta)} \tag{10.6}$$

where

$$c(\beta) = \left\{\frac{\Gamma(\frac{3}{2}(1+\beta)}{\Gamma(\frac{1}{2}(1+\beta)}\right\}^{1/(1+\beta)} \tag{10.7}$$

$$\omega(\beta) = \frac{\Gamma(\frac{3}{2}(1+\beta))^{1/2}}{(1+\beta)\Gamma(\frac{1}{2}(1+\beta)^{3/2}}$$

This is a normalized form, where $\mu$ and $\sigma$ are the mean and standard deviation of the data. The $\beta$ parameter is a measure of kurtosis[2]. When it is 0, the distribution is equal to a normal distribution. As $\beta \rightarrow -1$, the distribution becomes uniform over the interval from -1 to 1. As $\beta \rightarrow \infty$, the distribution approaches the delta function, $\delta(0)$. With $\beta$ equal to one, the distribution is Gaussian (hence the name - 'generalized' Gaussian).

## 10.8 Learning the basis functions and source distributions

Given a speech signal, the **A** matrix and the distributions of the source signals, $p(s_i)$, are to be learned. Standard algorithms exist that are able to do this, such as the icaML algorithm of the IMM ICA toolbox. However, it is highly desirable for classification purposes that each source distribution should be modelled accurately, since these distributions are the central part of the ICA classifier. Standard algorithms do not do this, since they assume a fixed distribution for all sources, e.g.

$$p(s_i) = \frac{1}{\pi}\exp(-\ln(\cosh(s_i))) \tag{10.8}$$

which is used by icaML. Therefore, an algorithm for learning **A** and $p(s_i)$ using the generalized Gaussian distribution has been implemented, inspired by and following the description in [17]. It is possible to design a flexible version

[2]A measure of the shape of a distribution

of equation 10.8 (see [2]), but as the generalized Gaussian distribution is a very good and flexible approximation of highly super- *and* sub-Gaussian distributions, it is preferred[3].

## 10.9 The generalized Gaussian ICA algorithm - icaEXP

Given a set of training examples, $\mathbf{X}$, this algorithm fits a distribution to each source signal, using the $\beta, \mu$ and $\sigma$ parameters, referred to as $\theta$. It also finds the corresponding $\mathbf{A}$ matrix.

It is implemented as an expectation-maximization (EM) type algorithm. Briefly, this means that the $\mathbf{A}$ matrix is optimized based on the current estimate of the sources, and vice versa. A good introduction to this topic can be found in [18].

The implementation can be found in the `Matlab` function `icaEXP.m`, which calls `learnbeta.m`, which fits a generalized Gaussian distribution to the current sources estimates (both in the ICA folder on the CD). The latter function relies on `genexp.m` and `genexplikely.m` for calculating $p(\mathbf{s})$ etc.

### 10.9.1 Estimating A

In the no-noise model (equation 10.1), $\mathbf{s}$ is deterministically given as $\mathbf{s} = \mathbf{A}^{-1}\mathbf{x}$ and equation 10.4 reduces to

$$p(x|\mathbf{A}) = \frac{p(\mathbf{s})}{|\det(\mathbf{A})|} \tag{10.9}$$

since the Jacobian is $\mathbf{J} = \frac{\partial \mathbf{s}}{\partial \mathbf{x}} = \mathbf{A}^{-1}$.

Equation 10.9 is the starting point for learning $\mathbf{A}$. If the ICA model is to be a good model of $\mathbf{x}$, then (an initially randomly chosen) $\mathbf{A}$ should be changed to make the given examples $\mathbf{X} = [\mathbf{x}_1\mathbf{x}_2...\mathbf{x}_N]^T$ more likely. Maximizing the logarithm of equation 10.6 is equivalent (as the logarithm is a monotonous transformation) and is used instead in practice to avoid numerical problems. T

With each iteration, $\mathbf{A}$ is updated by a form of gradient ascent as

$$\mathbf{A} = \mathbf{A} + \alpha \triangle \mathbf{A} \tag{10.10}$$

where $\triangle \mathbf{A}$ is defined as

$$\triangle \mathbf{A} = \mathbf{A}\mathbf{A}^T \frac{\partial}{\partial \mathbf{A}} \log p(\mathbf{x}|\mathbf{A}) = -\mathbf{A}(\phi(\mathbf{s})\mathbf{s}^T - \mathbf{I}) \tag{10.11}$$

where $\phi(\mathbf{s})$ connects the update of $\mathbf{A}$ to the distribution of $\mathbf{s}$. It is defined as

$$\phi(\mathbf{s}) = \frac{\partial \log p(\mathbf{s})}{\partial \mathbf{s}} \tag{10.12}$$

[3]'Super-Gaussian' distributions have are more peaked and have heavier tails then Gaussians, and vice versa for 'sub-Gaussian'

with each element being

$$\phi(s_i) = \frac{\partial \log p(s_i)}{\partial s_i} = -\eta |s_i - \mu_i|^{q-1} q c \sigma_i^{-q} \tag{10.13}$$

Here, $\eta = \text{sign}(s_i - \mu_i)$, $q = 2/(1 + \beta_i)$ and $c = [\Gamma(3/q)/\Gamma(1/q)]^{q/2}$.

With an estimate of the distribution parameters $\theta$, this gradient can then be calculated and $\mathbf{A}$ updated. The reason for pre-multiplying with $\mathbf{AA}^T$ is that this 'scaling factor' leads to much faster learning (see [10]).

Figure 12.28 shows the basis functions found when icaEXP is given speech data; see chapter 12 for analysis.

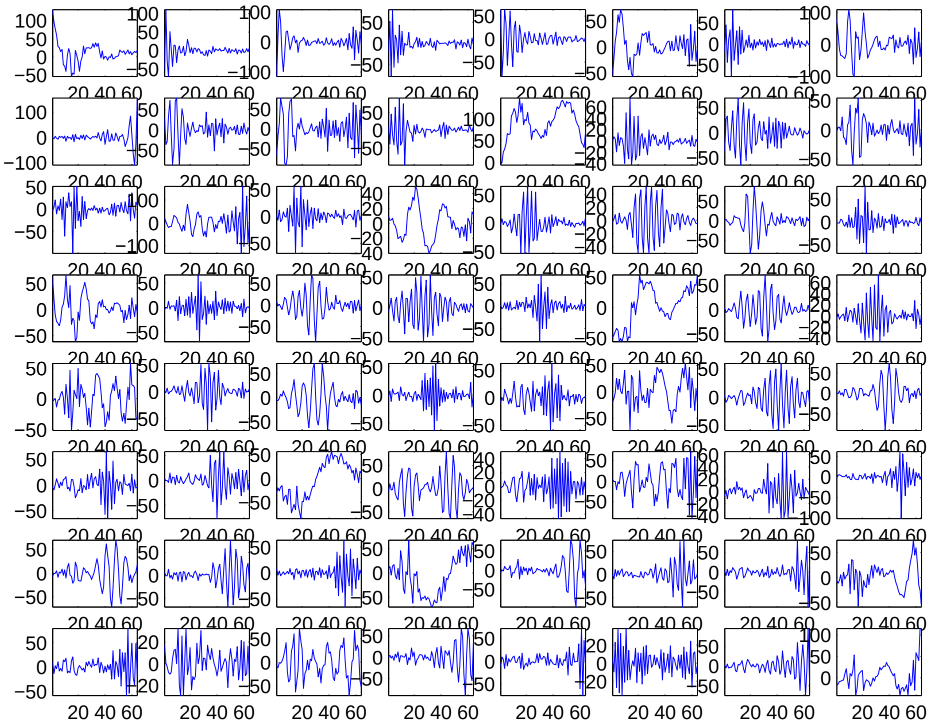

**Figure 10.3.** Basis functions for speech.

### 10.9.2 Estimating $\beta$

With an estimate of $\mathbf{A}$, the $\mu$ and $\sigma$ parameters can be found very simply from samples of $\mathbf{s}$ (mean and variance respectively), which is simply $\mathbf{A}^{-1}\mathbf{x}$.

$\beta$ can be found using Bayes' rule, as

$$p(\beta|x) \propto p(x|\beta)p(\beta) \tag{10.14}$$

where $p(x|\beta)$ is given by equation 10.6.

The optimal $\beta$ can then be found as the one that maximizes the likelihood of $\mathbf{s}$.

Since the form of $p(x|\mu, \sigma, \beta)$ defies analytical differentiation with respect to the parameters[4], numerical differentiation is done instead and used in a straightforward gradient (steepest) ascent algorithm. Figure 10.4 shows some of the distributions found for speech. For these, $\beta$ varies from 2 to 8, yielding very sparse distributions indeed. Each source has a different $\beta$ however, so much information can be gotten from modelling each source distribution separately. It is important to keep clear then that *each* individual source is modelled by a different parameter set, $\phi_i = \{\beta_i, \mu_i, \sigma_i\}$.

[17] recommends using a specific $\Gamma$ distribution for the prior on $\beta$, $p(\beta)$.

However, lots of data can be used instead, as there is practically no limit to computation time and available data (e.g. TIMIT for speech). The phase of learning the $\beta$ parameters (and the $\mathbf{A}_k$'s) takes place *before* the system is used for classification, so time is not an issue.

It was found that with source signals of length 5000 and more, there was no significant difference between the $\beta$'s found with and without inclusion of the prior. Also, it is difficult to see what a non-uniform prior on $\beta$ should be based on, since the model is 'artificial' and nothing is known about the 'true' range of $\beta$, except that it should exceed -1. And all $\beta$'s found by the numerical fitting algorithm where found to do this.

It was found that adapting $\beta$ too quickly lead to degeneration of the basis functions being learned at the same time. Therefore, the learning rate for $\beta$ and the decay of that learning rate were adjusted separately until robust learning of both $\mathbf{A}$ and $\beta$ was seen to take place. Further, initializing with too sharp distributions (high $\beta$'s) initially also led to poor learning.

### 10.9.3 Scaling

After each $\mathbf{A}$-estimation iteration, the $\mathbf{A}$ matrix is re-scaled to unit variance if it is found to be too large[5] However, no consideration is given to the final relative scaling of $\mathbf{A}$ and $\theta$, since the only important thing in this context is that they are correspond to each other. This is ensured by re-estimating $\theta$ after $\mathbf{A}$ is re-scaled in this manner so that the distribution parameters match the scaling of the mixing matrix.

The $\mu$ and $\sigma$ parameters are re-estimated based on *all* source signal data at the end of calculations, since this is very quickly done.

[4] At least as it seems to the author initially

[5] The extended ICA algorithm used in Lee and Lewicki instead re-initializes $\mathbf{A}$ (in their case actually the unmixing matrix, $\mathbf{w}$) at this point, which seems like a great waste of information.

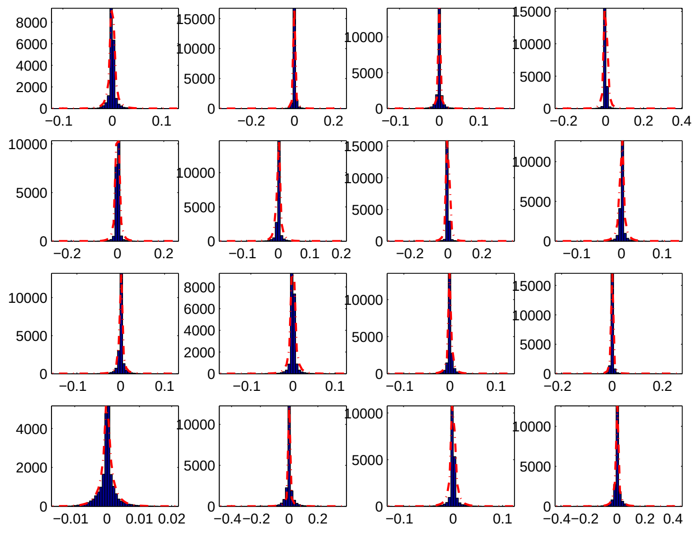

**Figure 10.4.** Several speech source histograms together with their approximated, generalized Gaussian distributions (ignore the scaling).

## 10.10 Gaussian noise - ICA model 1

Now, the simplest approach using a speech model is simply to use 10.1 alone, assuming that it will make a good 'fit' of the signal when speech is present (even if mixed with noise) and less well when speech is not present:

$$\mathbf{x}_{speech} \quad \simeq \quad \mathbf{A}_{speech}\mathbf{s}_{speech} \tag{10.15}$$

where $\mathbf{A}_{speech}$ and the $\beta$'s of the $p_{speech}(\mathbf{s})$ have been learnt beforehand by the icaEXP algorithm, trained on speech examples.

The next step is to add a very simple noise model, namely that of white noise. A very simple way to do this is to propose that the signal is *either* 'clean speech' (same as 10.15) *or* Gaussian white noise only, i.e.

$$\mathbf{x} \quad = \quad \epsilon \tag{10.16}$$

where

$$\epsilon \in N(0, \mathbf{\Sigma}_\epsilon) \tag{10.17}$$

Then it is assumed that the signal will be modelled most closely by 10.15 when speech is present and most closely by 10.16 when it is not. Thus $p(\mathbf{x}|C_{VA})$ is distributed as

$$p(\mathbf{x}|C_{VA}) = p(x|\mathbf{A}_{speech}, \mathbf{s}_{speech}) = \frac{p_{speech}(\mathbf{s})}{|\det(\mathbf{A}_{speech})|} \tag{10.18}$$

and non-speech signals are distributed as

$$p(\mathbf{x}|C_{NVA}) \quad = \quad \frac{1}{2\pi^{M/2}|\mathbf{\Sigma}_{\epsilon}|^{\frac{1}{2}}} \exp\left(-\frac{1}{2}\mathbf{x}^T\Sigma_{\epsilon}\mathbf{x}\right) \tag{10.19}$$

where $M$ is the dimensionality of $\mathbf{x}$. Since all of the signal is noise under this hypothesis, the noise variance $\Sigma_{\epsilon}$ is simply the signal variance. $P(C_{VA}|\mathbf{x})$ is then simply found using Bayes' rule (equation 4.2). This model is used in the VAD implemented in `icaVAD1.m` (ICA folder on the CD).

## 10.11 Modelling noise - ICA model 2

Keeping the either-or assumption on the input signal, other noise types can be modelled through the class conditional distributions over $\mathbf{x}$, exactly as by 10.18, i.e.

$$\begin{aligned} p(\mathbf{x}|C_k) &= \frac{1}{|\det\mathbf{A}_k|}p_k(\mathbf{s}) \\ &= \frac{1}{|\det\mathbf{A}_k|}p_k(\mathbf{A}_k^{-1}\mathbf{x}) \end{aligned}$$

where $k = 1...K$ and $K$ is the total number of noise classes modelled. In the present case, this was done for traffic noise, extending model 1 with this additional model (and corresponding hypothesis) and some results can be seen in chapter 12. This VAD is implemented in `icaVAD2.m` (ICA folder on the CD).

## 10.12 Modelling mixed signals - ICA model 2B

The assumptions above that the signal is *either* clean speech *or* some form of noise (only) is a simplification that is counter to the realities that are to be tackled, as defined in the introduction. Real-life speech signals always contain additive noise, even though the SNR may sometimes be very high. Noise may occur without speech, but rarely the other way around.

One way of learning to handle this would be to train several ICA models ($\mathbf{A}$'s and $\theta$'s) and then keep the either-or assumption, proceeding just as with ICA model 2, only with (many) more models. However, this would require a great many $\mathbf{A}$ and $\theta$ parameters (say 3 for each noise type (SNR 0, 5 and 10)) and so this approach might become too cumbersome in practice.

## 10.13 Mixture models

Only a limited number of models can be trained on different noise and speech combinations (type and SNR). This leads to the idea of ICA *mixture models*, where a fixed number of ICA models are used to model an arbitrary variety of mixed signals - a very general approach. The key difference from the models above is that each model ('cluster') then has a certain probability of representing speech. Thus, there need not be any single model that corresponds to clean speech. With this flexibility, more modelling 'power' should be obtainable with relatively few clusters.

One way of doing this is actually a sort of hybrid algorithm, lending the EM-like inference algorithm from [16] and the ICA models and $p(\mathbf{s})$ distributions from [17].

[16] can be modified to use ICA models for modelling class conditional probabilities instead of Gaussians. The algorithm can then be run in principle as shown in figure 1 in [16]. Instead of the update rules in [16], the ICA learning rules from [17] can be used instead, i.e. icaEXP is used to learn the source distributions of the components.

This algorithm was implemented towards the end of this project, but could not be tested on speech data. It did however produce promising results on artificial data (see 12.5.5).

### 10.13.1 Online learning

It would be extremely nice if the mixture model algorithm were able to learn continuously or online. In an actual hearing-aid implementation, this could be done by collecting new noisy data all the time and updating the model using these. This could be tuned so that learning was robust; possibly, the system could be split into an adaptive and a non-adaptive part. The output of the system would be a weighted sum of the outputs of these separate systems. For instance, original cluster(s) representing clean speech could be left in their original state, especially if they were trained on a great number of data. The same would also apply to an expansion into separate models for male and female speech or for voiced and unvoiced speech, if that was modelled specifically.

## 10.14 Speech compression

A bonus of the ICA VAD algorithm (e.g. model 1) is that it is very easy to turn it into a speech compression algorithm. Given a clean speech signal, these are simply arranged as an $[M \times N]$ matrix, unmixed by multiplication with $\mathbf{w}$, and the resulting (sparse!) source signals $\mathbf{s}$ can then be compressed in a straightforward manner, e.g. simply setting most of the elements to zero and keeping only a few. It was found that with $M = 64$, using just the single highest element in each source vector still left the resulting speech understandable, albeit only just - quite far from loss-less compression. But keeping 8 out the

64 elements produces speech that is quite bearable to listen to and completely understandable - a compression factor of 8. It is quite conceivable that with M set higher, say at 128, higher compression could be gotten with equally good results since 64 samples as discussed previously is a very short segment of speech. Figure 10.5 shows an example of a compressed speech signal.

Of course, with better compression algorithms much better compression could be obtained compared with using the very naive method outlined above.

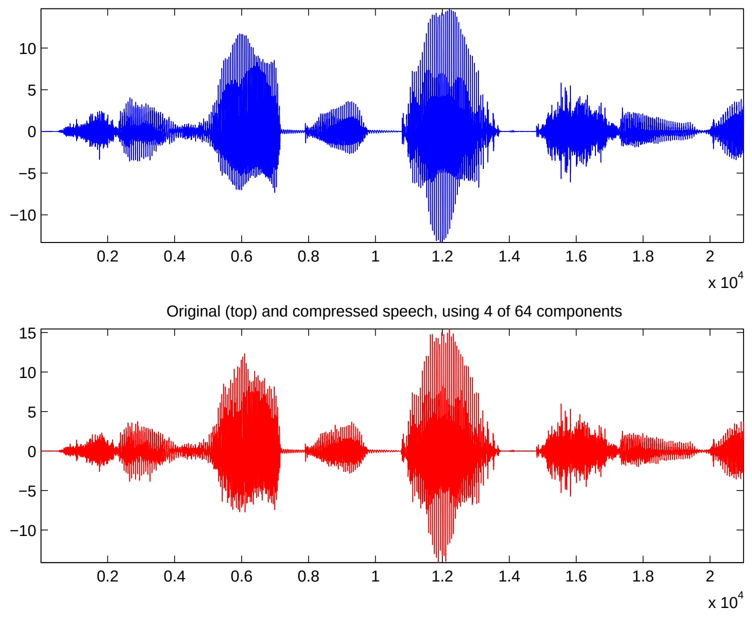

**Figure 10.5.** Original signal (top) and ICA compressed signal, using 4 out of 64 basis functions. The result is quite easy on the ears.

# Part III

# Experiments

# Chapter 11

# Evaluation method

This chapter describes the method used to evaluate performance of the various VAD algorithms.

## 11.1 The confusion matrix

Given a set of data points, $X$, the corresponding class labels assigned by a classifier, $C_{est}$, and the true class labels, $C_{true}$, the *confusion matrix* of the classifier can be calculated. It expresses wrong and correct classification in relation to the true classes.

| | estimated | |
|---|---|---|
| known | 0 | 1 |
| 0 | A | B |
| 1 | C | D |

**Table 11.1.** The confusion matrix

It is assumed that all class labels are binary, 0 (for the first class) or 1 (for the other). The 0's of are called ' negatives', the 1's 'positives'. In the present case, 1's would represent the presence of speech.

The confusion matrix has four entries: A (The number of negatives correctly classified by the classifier as negative, called 'true negatives'), B (The number of true negatives incorrectly classified as positive, called 'false positives'), C (The number of true positives incorrectly classified as negative, 'false negatives') and D (The number of true positives correctly classified as positive, 'true positives').

In general, the confusion matrix can be extended to accommodate any number of classes.

Two traditional measures can be read from this table, namely *sensitivity* and *specificity*.

Sensitivity is the probability of the estimate being 1 given that the true class is 1, so it measures how good the classifier is at getting all the true positives right. In terms of the confusion matrix, sensitivity is D/(C+D). This may also be called the True Postive rate (TP). Specificity is the probability of the estimate being 0 given that the true class is 0, so it measures how good the classifier is at getting all the true negatives right. Ideally, both should be 1, in which case the confusion matrix is diagonal.

A third useful measure is the False Positive rate (FP), which is B/(A+B). It is sort of the opposite of the specificity and together with the TP, it is used to generate a more complete picture of the classifier, namely a 'Receiver Operating Characteristics' (ROC) curve.

The simplest measure of the quality of a classifier is the 'error rate', which is (B+C)/(A+B+C+D), i.e. the fraction of total misclassifications. However, this does not contain as much information as a complete ROC curve.

## 11.2 Receiver operating characteristics (ROC) curves

The linear network and the ICA models output the estimated posterior probability of class membership, i.e. in this case the probability of speech presence given the input ($P(C_{VA}|\mathbf{x})$). To get from this probability to a binary VAD decision signal, a threshold is chosen (4.3). This choice determines the numbers in the confusion matrix. If the threshold is lowered, more examples will be classified as true, and both TP and FP will rise.

The Receiver Operating Characteristics (ROC) of a classifier shows its performance as a trade off between TP and FP.

It is used in cases where there are only 2 classes, or where one wants to examine only 2 classes at a time. A curve of the FP versus the TP is generated by varying the threshold.

The curve always goes through (0,0) and (1,1). (0,0) is where the classifier finds no positives. In this case it always gets the negative cases right but it of course also gets all positive cases wrong. This point corresponds to setting the threshold higher then the largest output for all examples. Similarly, in (1,1), everything is classified as positive. So the classifier gets all positive cases right but it also gets all the negative cases wrong. This corresponds to setting the threshold lower than the smallest output for all input.

A classifier that assigns class labels at random produces a line which lies close to the diagonal connecting (0,0) and (1,1). This is true no matter which proportion of examples is true (prior probability, $P(C_{VA})$). Therefore, this diagonal is always included in graphs as a sort of 'ground zero' comparison.

It is possible to do worse than a random classifier. This means the classifier's answer is negatively correlated with the actual answer, so that its ROC will lie below the diagonal. Its performance can actually be improved by inverting the classifiers output.

One way of comparing classifiers is to calculate the area under the ROC. A random classifier has an area of around 0.5, while an ideal one has an area of 1. However, the most information about the relative performance of two classifiers can still be obtained by inspecting the ROC curves together.

For actual decision making, a threshold has to be selected. This corresponds to fixing the classifier at a point on its ROC curve. Depending on how one wishes to weigh specificity and sensitivity (e.g. false positives might be more catastrophic than false negatives), the best classifier may not be the one with the largest area under the ROC curve.

In this way, both changes in estimates of prior class probability ($P(C_{VA})$) and in the relative cost of misclassifications (cost of misclassifying true negatives versus cost of misclassifying true positives) will simply move the location on the ROC curve for a given threshold.

The variance of each estimate (FP and TP) is binomially distributed (the classification of each example is a Bernoulli trial). Therefore, confidence intervals can be calculated and plotted for each point on a ROC curve. Of course, this can be done for both the FP and TP estimates. However, for the sake of figure legibility, only the bounds on the TP estimate are given; the bounds on the FP estimate will be of the same magnitude and for comparing ROC curves of different systems, reading the TP values is perhaps intuitively easier. Also, the figures will be less cluttered.

An algorithm has been developed that automatically produces a useful ROC curve from any given estimated and true VAD signal. It does this by varying the threshold in a way that fits with the signals to produce evenly distributed ROC curve points. This algorithm is given in pseudo-code form, see algorithm 2. The `RocPoint` helper function simply calculates the fraction of false- and true positives resulting from a chosen threshold level. The `TooDifferent` helper function simply decides whether or not a candidate ROC point is too close to either adjacent point.

This algorithm can be found in the `Matlab` function `roc.m` .
This calls `getconfusion.m` to calculate ROC points (both in myfunctions folder on the CD).

## 11.3 The effect of changes in prior class probabilities

The linear network has a bias unit which allows it to 'pick up' the mean level of speech in the data it is trained on. If the speech content in the training is high, it will learn to increase the bias weight so as to match this content. The other weights mainly determine the real quality of the linear network as a classifier. With new (test) data sets, if the speech content is say lower than in the training set, the bias of the network will cause it to over-estimate $P(C_{VA}|\mathbf{x})$. However, the ROC curve for this classifier will not change. It is possible to move the operating point of the classifier back to the original point on the ROC curve by

**Algorithm 2** ROC (L) - Find ROC points given level set L

```
$R \leftarrow \emptyset$
{ROC Point set}
$T \leftarrow \emptyset$
{Corresponding threshold set}
$l_{min} \leftarrow min(L)$
$l_{max} \leftarrow max(L)$
$R \leftarrow R \bigcup \{(0,0),(1,1)\}$
{Initially known points}
$T \leftarrow T \bigcup \{L_{min}, L_{max}\}$
{Corresponding threshold}
repeat
  {Find the 2 adjacent points with greatest Euclidian distance between them}
  $k, l \leftarrow \arg\max_{i,j}\{\|r_i - r_j\|; |i-j| = 1\}$
  $\mathbf{r_k} \leftarrow R_k$
  $\mathbf{r_l} \leftarrow R_l$
  $t \leftarrow 0.5T_k + 0.5T_l$
  $\mathbf{r_{new}} \leftarrow RocPoint(t)$
  $D_1 \leftarrow Distance(\mathbf{r_{new}}, \mathbf{r_k})$
  $D_2 \leftarrow Distance(\mathbf{r_{new}}, \mathbf{r_l})$
  while $TooDifferent(D_1, D_2)$ do
    if $D_1 > D_2$ then
      $r_l \leftarrow r_{new}$
    else
      $r_k \leftarrow r_{new}$
    end if
    {Update the new point}
    $t \leftarrow 0.5T_k + 0.5T_l$
    $\mathbf{r_{new}} \leftarrow RocPoint(t)$
    $D_1 \leftarrow Distance(\mathbf{r_{new}}, \mathbf{r_k})$
    $D_2 \leftarrow Distance(\mathbf{r_{new}}, \mathbf{r_l})$
  end while
  {Add new point}
  $R \leftarrow R \bigcup \mathbf{r_{new}}$
  $T \leftarrow T \bigcup t$ {Corresponding threshold}
until Desired number of points found
```

simply increasing the threshold sufficiently.

For the ICA models, classification requires an estimate of $P(C_{VA})$. Again, the ROC curve of the ICA classifiers do not change and movement on the ROC curve can be done by either re-estimating $P(C_{VA})$ or changing the threshold.

Similar observations are applicable when dealing with the *loss function*, where false negatives might not have the same cost as false positives. Changes in the loss function do not affect the ROC curve, but do affect the point on the curve where the VAD would be set to operate.

# Chapter 12

# Results

This chapter presents the more important results from the various experiments done with the different VAD algorithms. This covers each algorithm separately, followed by comparative results. First, the linear networks are examined, then the ICA models and the ITU-T VAD and OTI VAD . Some relevant comparisons are then made between the various models and VAD's. Additional results can be seen in the figures of appendix D.

In comparative experiments, the (randomized) data set is exactly the same for each classifier in order to test on as equal terms as possible. The results are given in an order that should facilitate the drawing of the most important conclusions. Results for individual classifiers are given first, and these are then compared.

Initial discussion and analysis is also done in chapter, but the more major discussions and conclusions are referred to the following chapter.

## 12.1 Linear network results

Several issues and questions regarding the linear network classifier are resolved through experimentation.

### 12.1.1 Determining the stopping criteria

How long (how many iterations) should the linear network take before training is stopped? Several criteria for stopping are typically used, such as stopping when the error gradient falls below some threshold. Here, the simplest (and often the only robust) criteria is used, namely stopping after a fixed number of iterations. The appropriate number of iterations naturally depends on the particular type of data that is used for training. However, using 5000 examples (see following section) and batch-training, it was found that around 50 iterations was enough to ensure convergence for the linear networks; see figure 12.1.

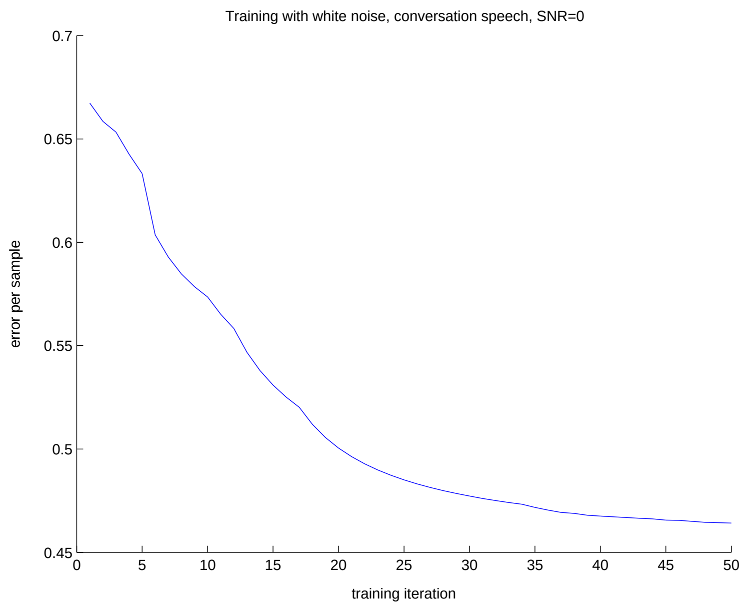

**Figure 12.1.** After around 50 iterations, each using 5000 examples, learning has converged.

### 12.1.2 Determining the training data set size

For each type of network, training to a fixed number of iterations was done using different-sized training sets. This was in order to ensure a sufficiently large training set. Clearly, the larger the training set (compared with the number of parameters in the classifier), the more the latter will be ’forced’ to learn the general structure in the data, unable to memorize each data point (input-output example). After training, each network (having trained using a different-sized training set) is tested on a ’validation’ data set. It is then possible to see how big the training set should be for the classifier to learn robustly. The overall conclusion was that around 5000 examples are enough to assure this ’asymptotic’ learning - see figure 12.2.

Determining the number of training iterations and the size of the training set was of course based on several different noise types and SNR conditions.

### 12.1.3 Preprocessing

The input data was normalized as described in 3.6.

Secondly, frames of length 50ms where extracted. For each frame, features are extracted. The squared filterbank outputs are summed into one value for each filter for each frame, as are all cross-correlations. So for each frame, there are 9 filterbank outputs and 36 cross-correlation outputs available as features.

The filterbank is implemented using a modified version of the mel-scale code from Slaney’s Auditory Toolbox [28]. This uses an FFT to implement the

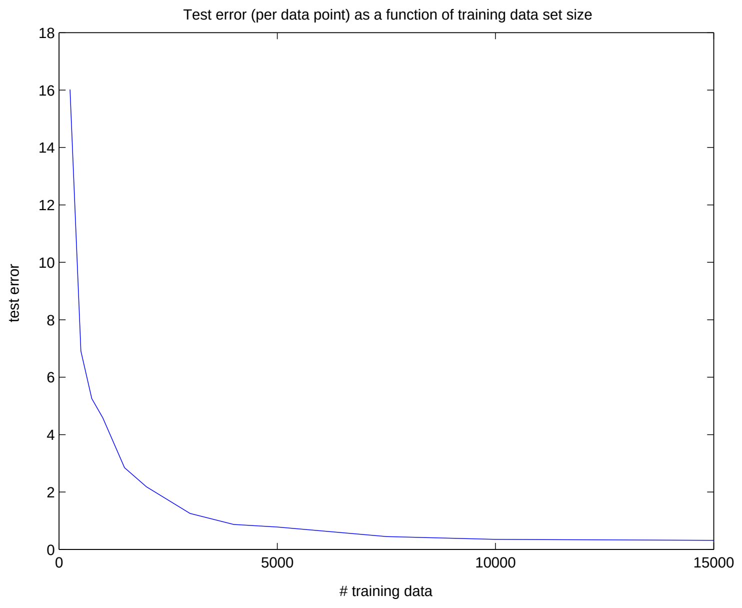

**Figure 12.2.** Using around 5000 examples is enough to ensure reliable learning.

mel-scale filtering. After filtering, each output signal is squared (to get rid of dependence on phase) and finally the elements (corresponding to samples in the frame) are summed to give a one-dimensional signal per frame per filter.

The cross-correlations are done by cross-correlating the squared filterbank output signals and the summing over the elements of each resulting signal, again giving a one-dimensional signal per cross-correlation per frame.

The network will learn an appropriate scaling of each weight. It is important of course that training and test data are normalized in the same way. The input normalization is done in a standard way for all data sets as described in 3.6.

## 12.2 Comparison of features

### 12.2.1 Determining the size of the filterbank

The first question is how much difference the number of filters in the filterbank makes, when used as inputs to linear networks, otherwise trained in the same way. Two different sizes are tested, namely 9 and 18 filters.

Figure 12.3 shows one result, shown as a ROC curve, for a particular noise type and SNR. The difference is seen to be quite negligible, and this was found to be a consistent conclusion across noise types and SNR's.

In the name of simplification, the size 9 filterbank is chosen. Also, 18 filters make for 153 possible cross-correlations, which will be more difficult to examine than the 36 possible ones for a size 9 filterbank.

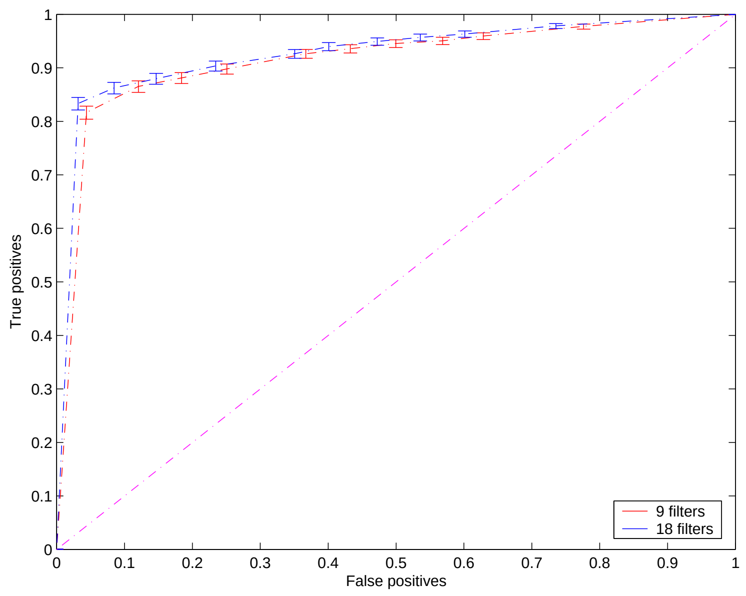

**Figure 12.3.** ROC comparison for networks with 9 and 18 filterbank inputs. White noise at SNR=10, very similar results were obtained for other noise types.

### 12.2.2 9 filter-bank and 36 cross-correlations

The second question is how the 9 'raw' filterbank outputs compare with the 36 cross-correlations when used as inputs to linear networks, otherwise trained in the same way.

Speech does have a unique distribution of power over frequencies, especially for voiced speech (see figure 3.1)

So even though the input normalization will make it impossible to simply use the energy of the signal as a VAD feature, a linear network can still learn to weigh the frequencies in a way that can give some discriminative power in certain cases. For white noise, this works quite well, see figure 12.4. But for other cases, linear separation becomes very difficult, see figure 12.5.

The cross-correlations are able to capture much more information than the raw filter-bank outputs. A linear network can then learn that certain frequencies should be positively correlated for speech, others should be negatively correlated, and others still might be irrelevant. This leads to the idea of pruning, where weights corresponding to irrelevant cross-correlations are removed from the network - see 9.4.

However, 36 parameters is a very low number compared to the number of available training data, so it is expected that pruning will not give much (if any) actual *improvement* of performance, but it might be possible to reduce the network size significantly without *degrading* performance to a marked degree.

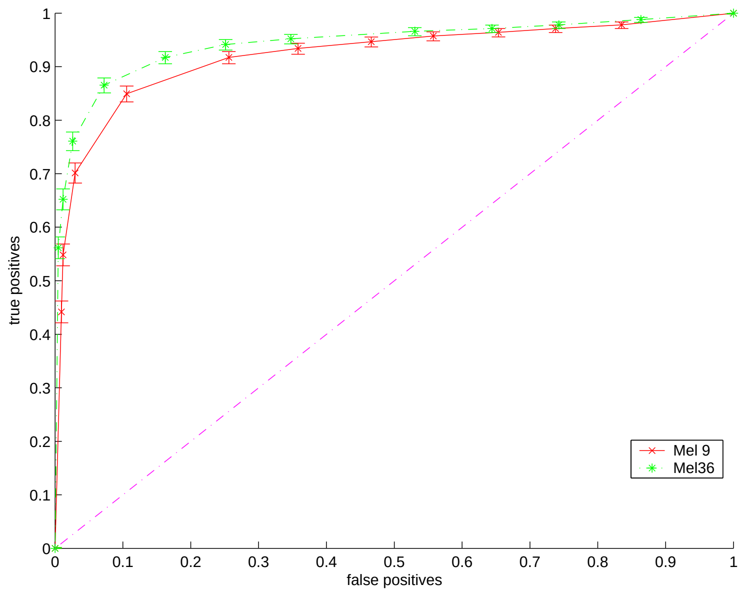

**Figure 12.4.** 9 raw filter-bank power compared with using all 36 possible cross-correlations between them. In this 'easy' case (white noise at SNR 0), the gain is not overwhelming.

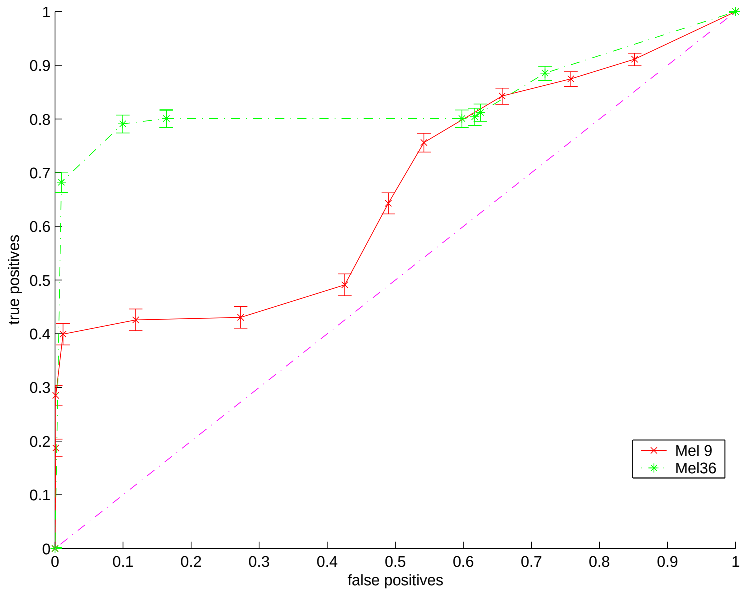

**Figure 12.5.** 9 raw filter-bank power compared with using all 36 possible cross-correlations between them in a 'hard' case ('clicks' noise at SNR 0). Now, the 'raw' features do not discriminate well and the resulting VAD performs poorly.

## 12.3 Separate voiced and unvoiced classifiers

Surprisingly, having two separate classifiers, one for voiced and one for unvoiced speech does not improve upon results, even when using the same features as input, see figure 12.6.

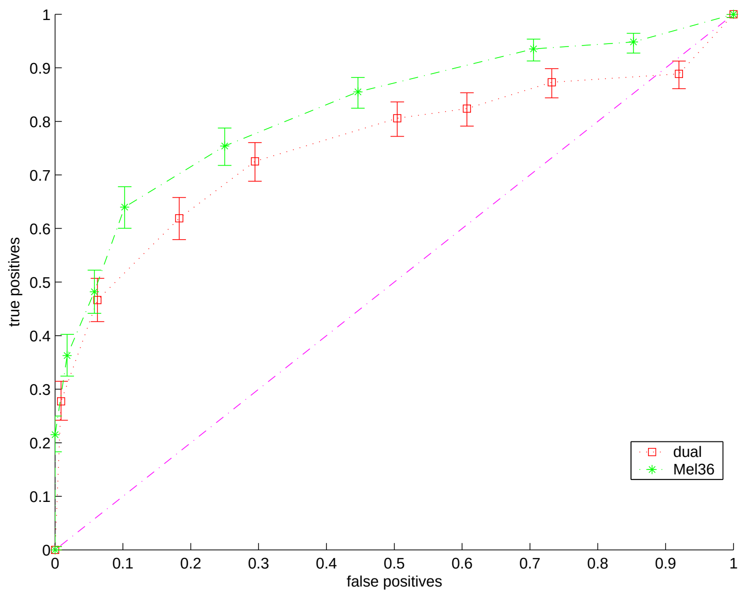

**Figure 12.6.** Single classifier using 36 cross-correlation inputs compared with a combined output from a voiced an an unvoiced classifier

.

The reason for this is undetermined and more work should be done in this area.

## 12.4 Pruning

When pruning (see 9.4), the outcome is different for each 'run', depending on the data set used and the random initialization of the linear network weights to be pruned. To illustrate this, figure 12.7 shows several pruning runs together - each run is shown by plotting the validation error as a function of the number of weights pruned. White noise seems to be the exception, being very 'well behaved' and producing uniform results (figure 12.8).

Each combination of noise type and SNR was investigated by pruning and examining the validation error.

Figures 12.10 to 12.14 show some examples, where each is taken as the best (producing the best single network) of many runs on the same noise type and SNR combination (other figures in appendix D).

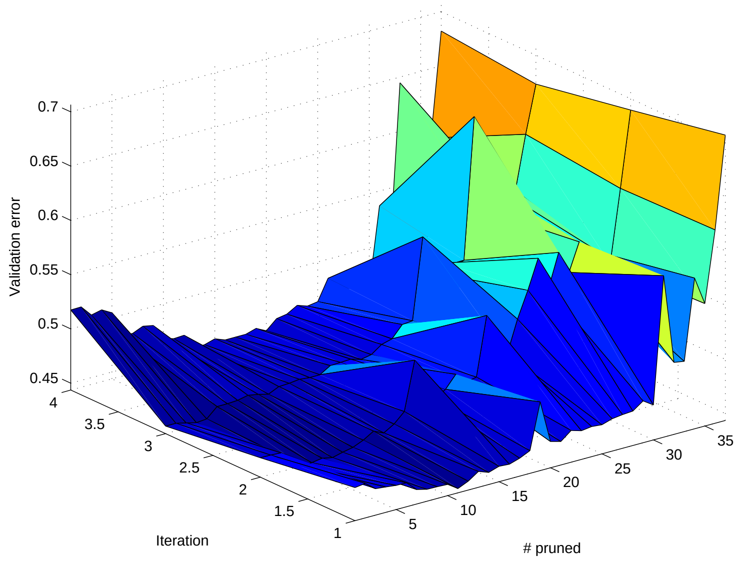

**Figure 12.7.** Validation error as a function of the number of weights that have been pruned away - several different pruning runs are plotted together. Traffic noise, SNR=0.

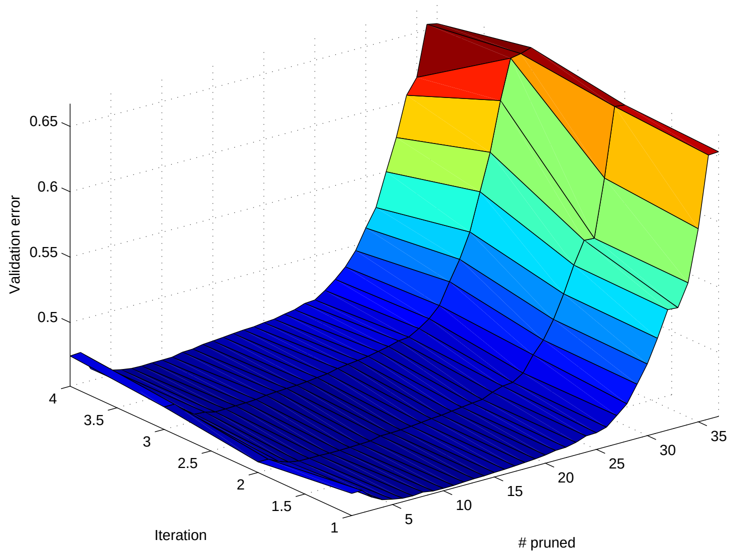

**Figure 12.8.** Validation error as a function of the number of weights that have been pruned away. White noise, SNR=0.

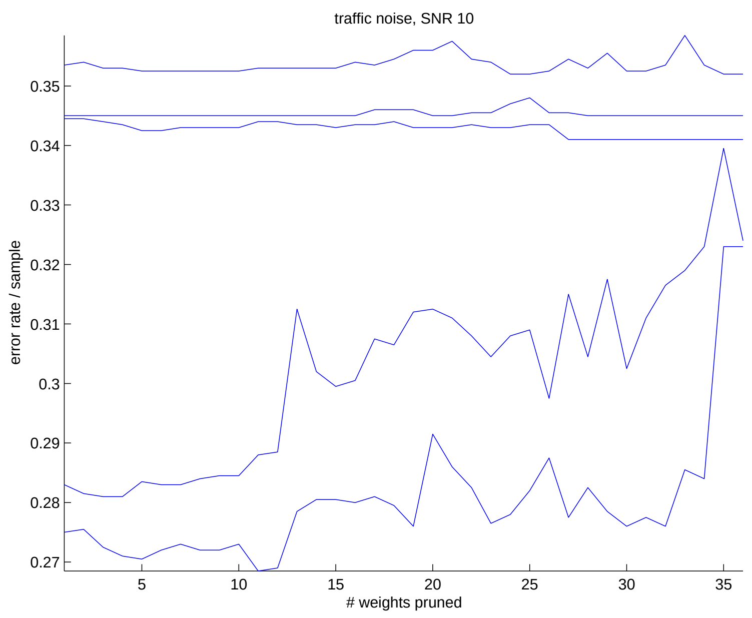

**Figure 12.9.** Several pruning runs in traffic noise at SNR 10. Notice how only a few networks (bottom graphs) are good enough from the beginning to be useful for analysis.

Note that for these experiments, the error rate and not the cross-entropic error is used to measure the error on the validation set. This is done to avoid the possible outlier problems with the cross-entropic error function, where a single miss-classification can produce a huge error, making it difficult to compare results. (The error rate is simply the total proportion of examples that are wrongly classified, measured per example (i.e. total error divided by number of examples)).

The validation error is typically reduced after pruning a few weights but then rises sharply when pruning more than half the weights:

In several cases, the pruning graphs reveal that the network is unable to solve the problem at all. This is when pruning all weights (except the bias weight) is insignificantly worse than pruning only a few weights. The error rate per point then is around 0.35 which is the mean proportion of non-VAD samples in the test sets. What the network is actually learning in these cases is simply the prior probability of the VAD class, $P(C_{VA})$. Only in the cases where the network is able to reach an error significantly lower than 0.35 can it be said to have learnt anything useful, as the bias on its own can learn $P(C_{VA})$.

For traffic, clicks and babble noise, SNR 0 pruning results only show that the problem is too difficult (error rate does not reach much less than 0.35); appendix D.

As an example of learning during pruning, figure 12.13 shows a pruning run with babble, where the network is finally able to learn the correct bias, only

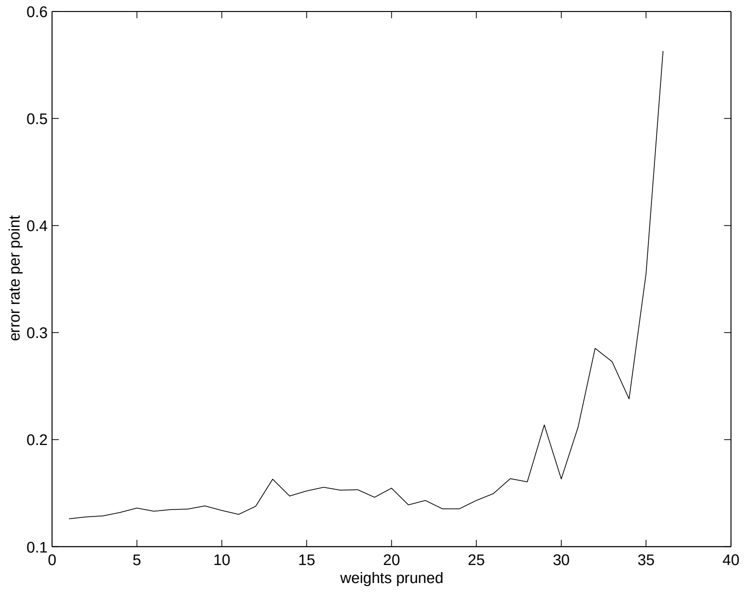

**Figure 12.10.** Validation error as a function of number of weights pruned. White noise, SNR 0.

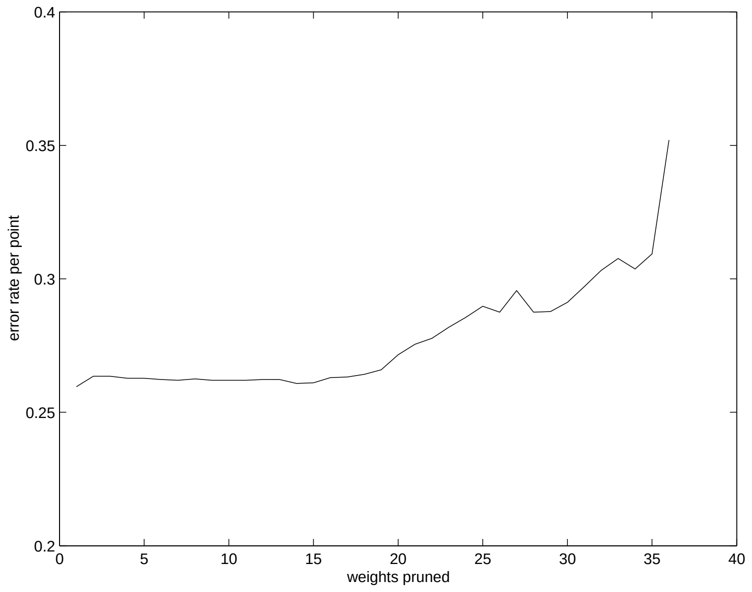

**Figure 12.11.** Validation error as a function of number of weights pruned. Traffic noise, SNR 10.

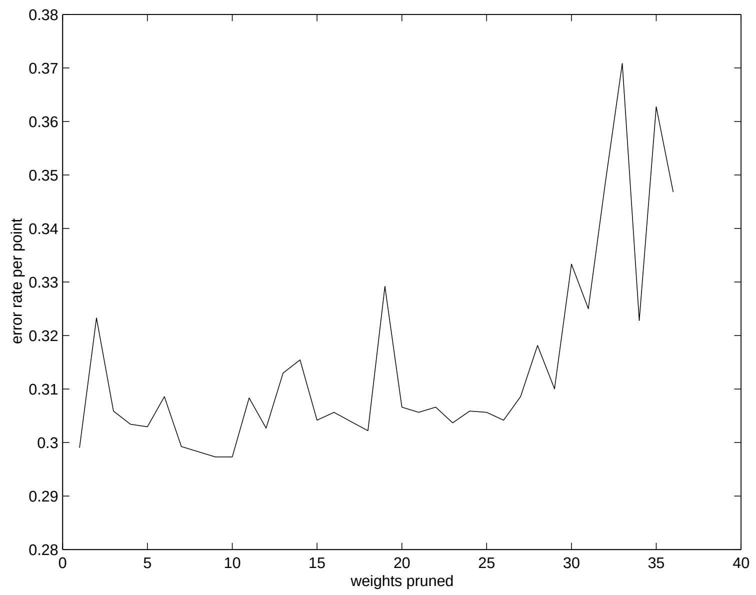

**Figure 12.12.** Validation error as a function of number of weights pruned. Clicks noise, SNR 10.

just making it down to an error rate of 35 percent.

Using training sets that are a combination of all noise types produces unimpressive results, see figure 12.14. This is probably due to the presence of babble noise, which makes the problem too difficult.

Pruning does not seem to produce better networks than not pruning. This is not surprising from the above figures. It would seem that the 36 cross-correlations all contribute to some degree to the discriminative ability of this feature set. Figure 12.15 is one example of comparison; similar results were found for all combinations of noise type and SNR. Here, the network found during pruning is re-trained from scratch (random initialization) in order to ensure proper learning (see [3], page 362). Also, with lots of training examples relative to the number of parameters (weights), overfitting (4.2) is not a problem.

All in all, it was chosen to keep all 36 cross-correlations as the feature set for the best possible linear network.

Still, the weights that were chosen by the pruning process and the corresponding value of those weights might say something about which frequency correlations are most important. This is illustrated by figures 12.16 to 12.20. These show the final networks found by pruning for different noise types and SNR's. Each network is the result of selecting the best network after several individual pruning runs. In the figures, red (solid) circles represent weights with positive value while blue (open) are negative. The size of each circle corresponds to the magnitude of the weight.

In the example shown in figure 12.20 which is babble noise at SNR 0, only the correlation between the first and second filters is kept; this is consistent with the

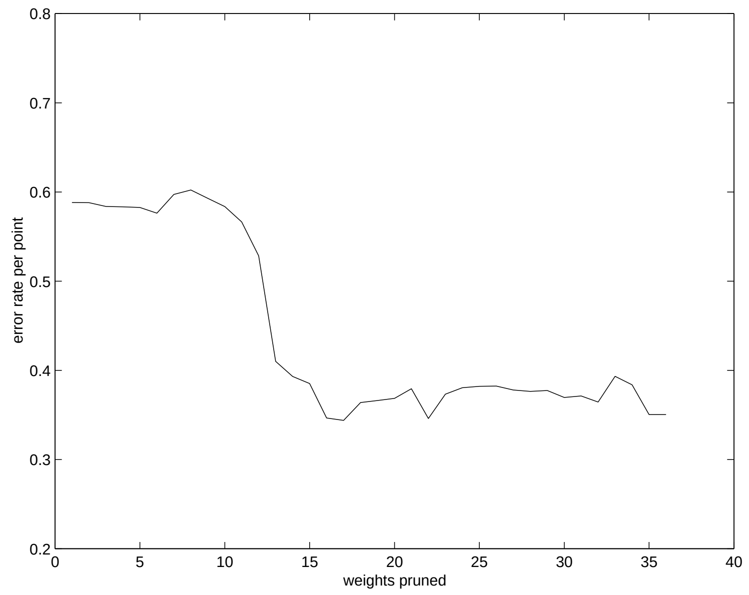

**Figure 12.13.** Validation error as a function of number of weights pruned. Here it seems that the network was not trained to completion prior to pruning, but has managed to 'learn' during pruning. Babble noise, SNR 10.

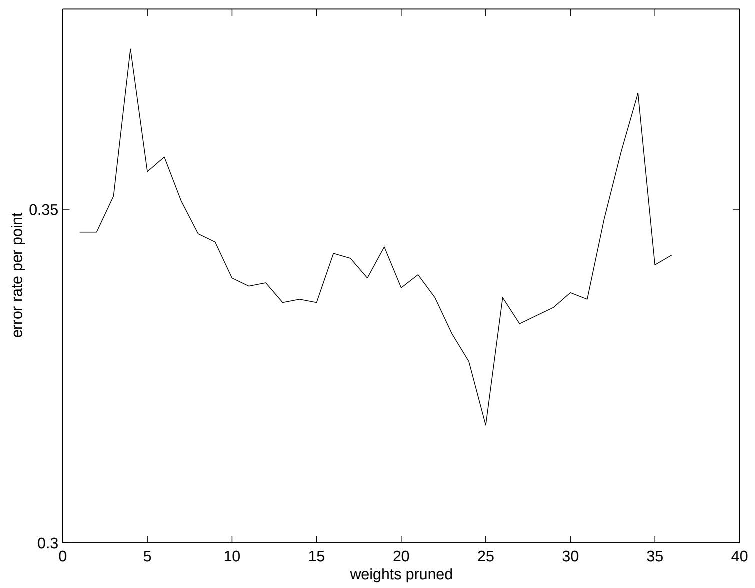

**Figure 12.14.** Validation error as a function of number of weights pruned. The optimum represents the learning of only slightly more than the prior class probability. Mix of all noise types, SNR 10.

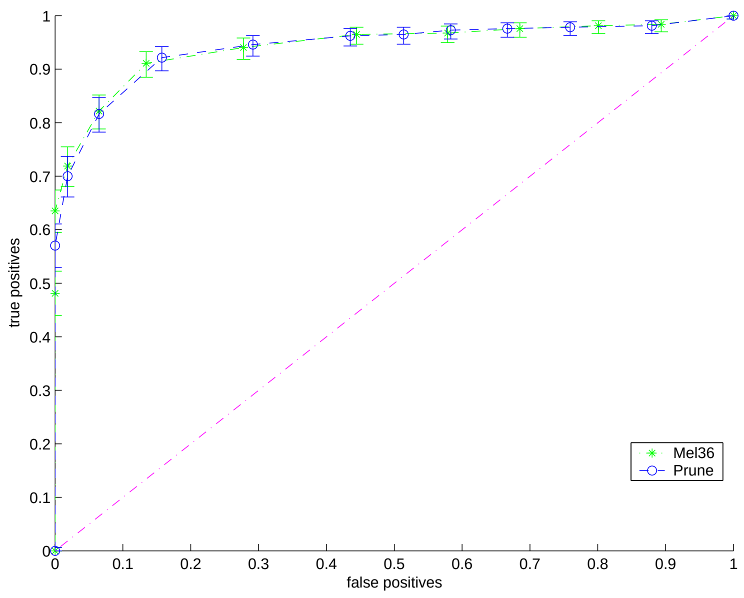

**Figure 12.15.** Performance using 36 cross-correlations versus pruning those correlations. White noise, SNR 10.

results from white noise. The babble network, however, performs very poorly.

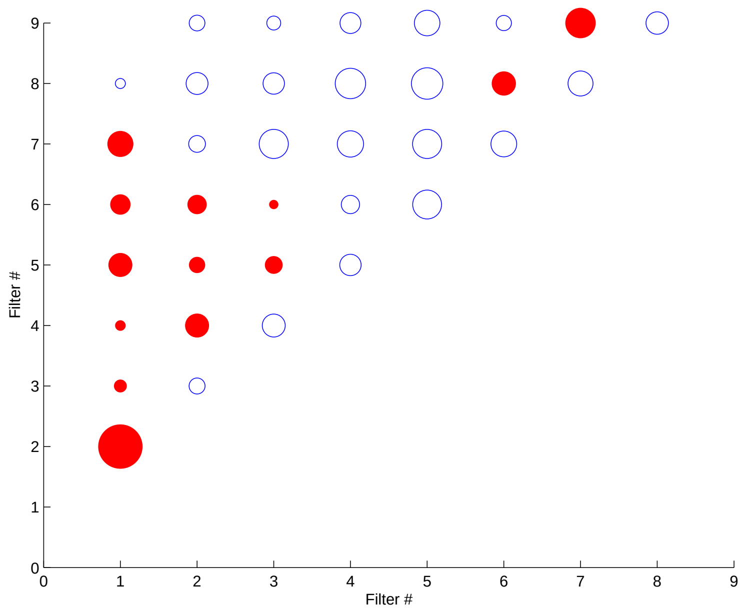

**Figure 12.16.** Relative weighting by a network trained and pruned on white noise data at SNR 0. Notice the strong positive weight between the first two (low-frequency) filters and the pattern of positive and negative weights.

When such figures are generated for each pruning run and then put together, the result says something about how likely each cross-correlation is to end up in the final (optimal for that run) network and what magnitude it typically then has. This is shown in figures 12.21 to 12.24. Every noise type and SNR combination is shown here together, as this facilitates inspection. All circles are now open, and the variation across pruning runs can be seen in the variation of each circle's diameter. Each figure represents 5 different pruning runs.

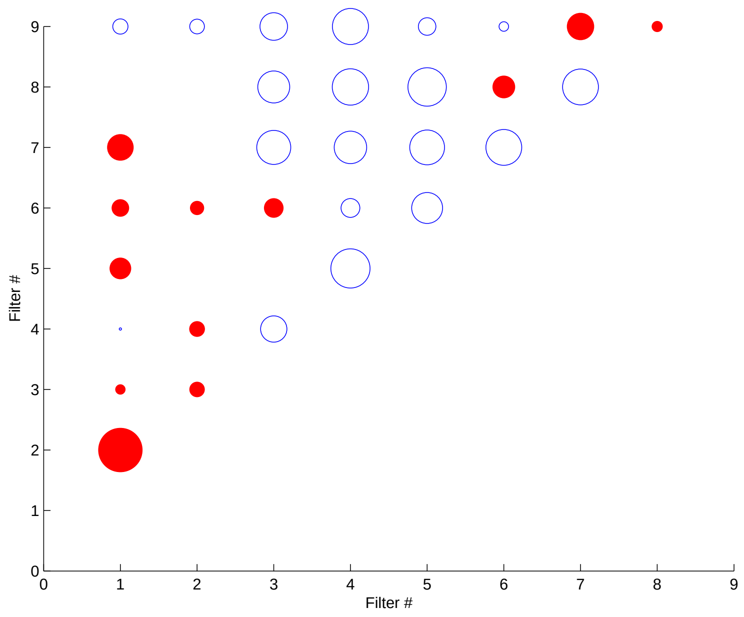

**Figure 12.17.** Relative weighting by a network pruned on white noise data at SNR 10. Notice the strong resemblance to figure 12.16, even though these networks were initialized randomly and trained on different data.

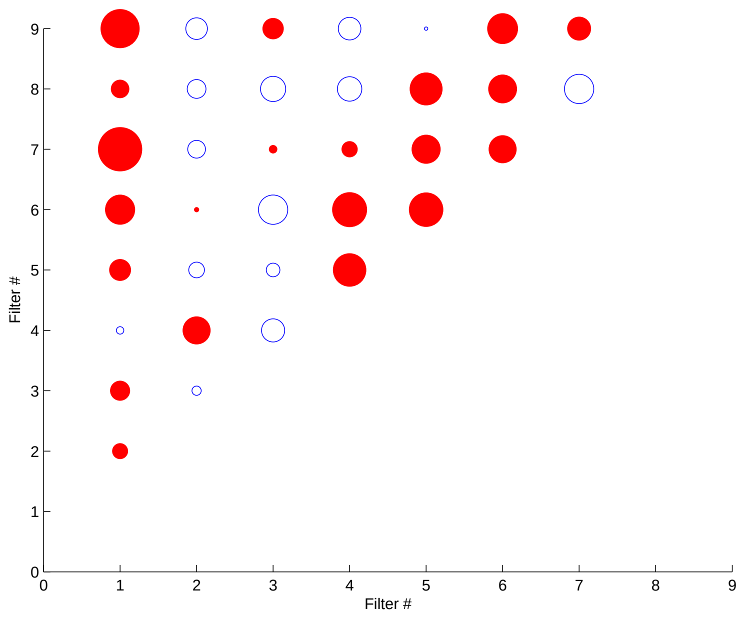

**Figure 12.18.** Relative weighting after pruning with traffic noise data at SNR 10. Notice the difference in weights compared with white noise (previous figures) - both the size and the sign are different.

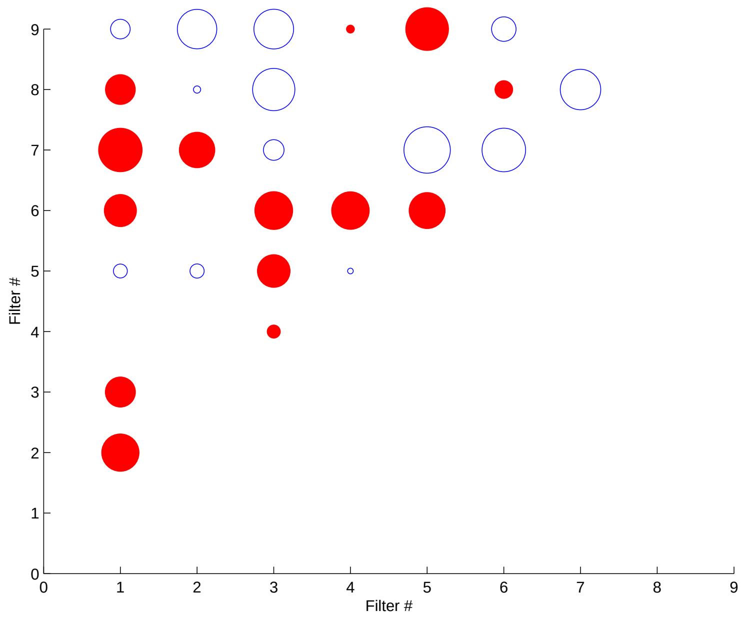

**Figure 12.19.** Relative weighting by a network after pruning with clicks noise data at SNR 10. This network had good performance; notice how the pattern is distinct for each noise type (compare with figures 12.17 and 12.18).

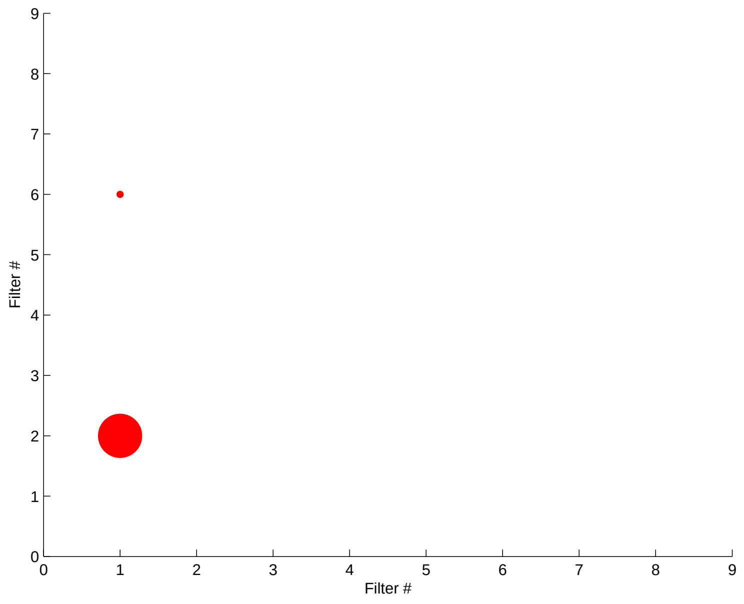

**Figure 12.20.** Relative weighting by a network pruned in babble noise data at SNR 0. This network performed poorly, but the remaining large weight may be a good choice still, see figure 12.17.

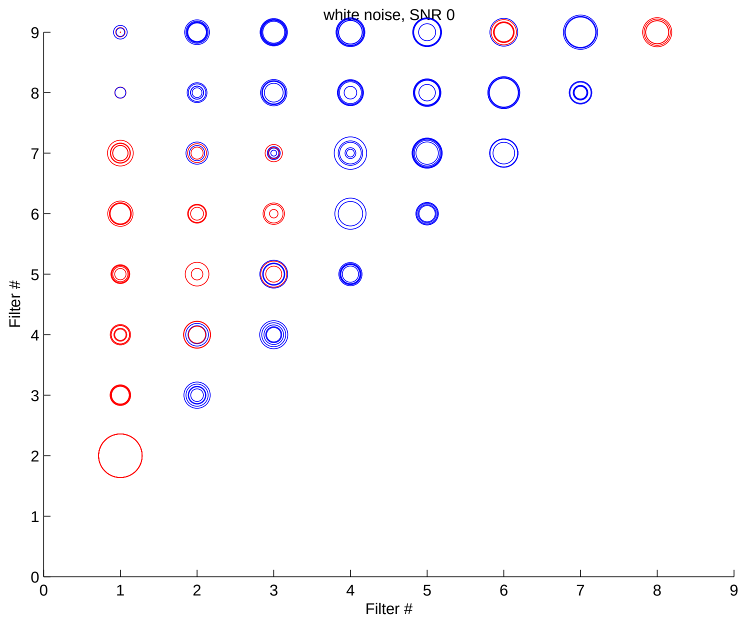

**Figure 12.21.** Relative weighting by several pruned networks with white noise, SNR 0. Notice how for most of the combinations, there is consensus as to sign and size.

Some observations can be made on these pruning results. First, for those networks that are able to learn more than just the prior probability of speech presence (which the bias weight handles), there is some consistency over which weights get chosen and their value. For instance, looking at speech in white noise, the positive correlation between the first and second filterbanks (low frequencies) seems to signify speech. For detecting speech in traffic, other correlations seem to be important. The low-frequency correlation is now not as important, probably because traffic also has much energy content at low frequencies. For those networks unable to learn much, most weights are pruned away and there is some randomness as to what is left.

Other patterns are also seen, but their interpretation is more speculative

The most correct way to select the final network might be to train a network - using the chosen features - on a combined training set containing all noise types and both SNR 0 and SNR 10 mixtures. For a practical system designed to operate in the range from SNR 0 to SNR 10, it would presumably be optimal to train that system with data distributed across this range. However, in the present case, networks were trained on all noise types but each network was only trained on a particular SNR. So to select a final network to be compared with the ICA, OTI VAD and ITU-T VAD , it is necessary to consider both networks trained specifically on a particular noise type (which is an unfair advantage compared with the OTI VAD and ITU-T VAD which only exist in one, general version) and those trained on all noise types at both SNR 0 and 10.

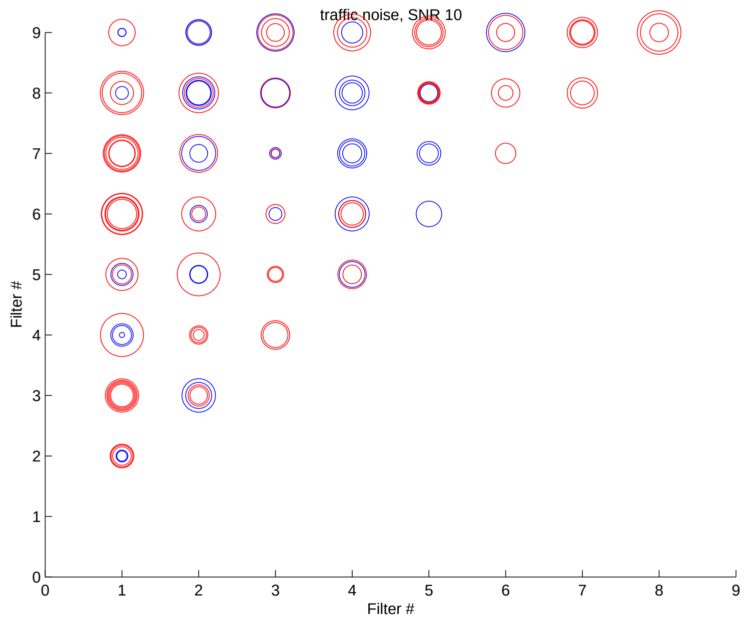

**Figure 12.22.** Relative weighting by pruned networks with traffic noise, SNR 10. With traffic, even at SNR 10, there is less consensus than for white noise between runs (see previous figures).

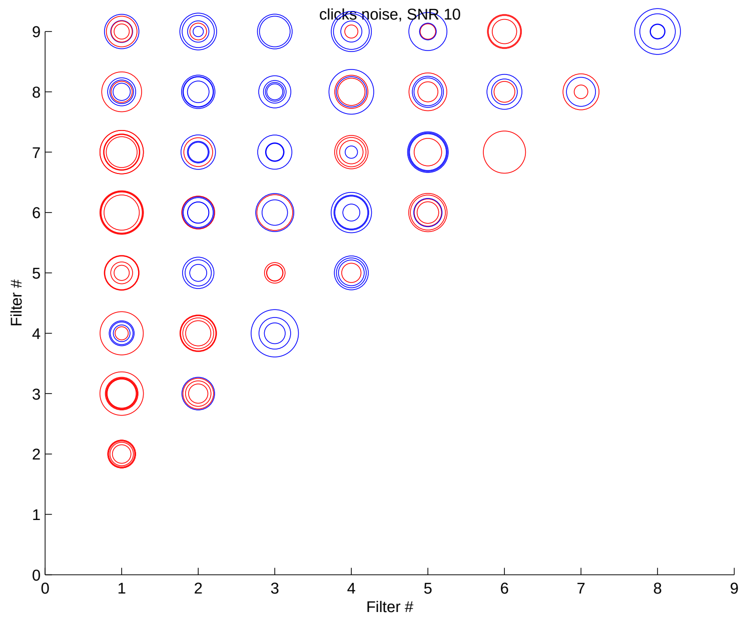

**Figure 12.23.** Relative weighting by pruned networks with clicks noise, SNR 10. There is more variation in the choice of weights by networks of different pruning runs than for white noise (figure 12.21).

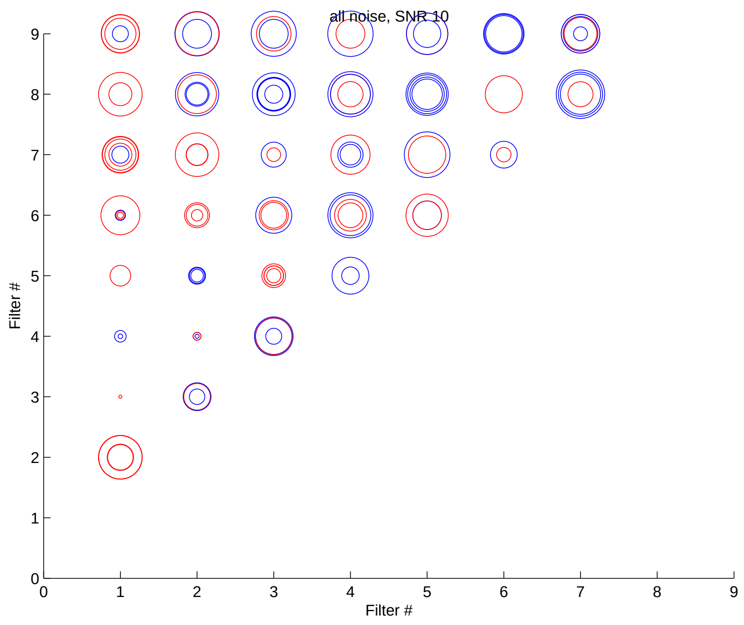

**Figure 12.24.** Relative weighting by pruned networks with a mixture of all noise types, SNR 10. As might be expected, mixing all noise types leads to the most variation.

## 12.5 ICA results

### 12.5.1 Learning ICA models - icaEXP

To learn basis functions and corresponding source distributions for speech, a data set consisting of 10 persons (5 females and 5 males) speaking unique sentences was used. From this, 20000 examples each with a length of 64 samples were taken. On these examples, the icaEXP algorithm was run. The **A** matrix was initialized to the unit matrix **I**. The $\beta$'s were initialized to 0.5, i.e. only slightly super-Gaussian. The algorithm ran for 64 iterations with an initial learning rate of 0.001.

In the case of speech signals, it was found that all sources were super-Gaussian, as expected and found by others (see [8]).

The basis functions found generally contain signals that resemble the modelled signal itself. This resemblance is to be expected given the extreme sparsity of the sources, so that often a single basis function will approximate the modelled signal well. See figures figure 12.28 and figure 12.36.

It was found that for click-type noise, the basis functions found were more or less random, containing no phase information. This is reflected in the fact that ICA model 1 (see 12.5.3) is not 'cheated' by this type of noise, rather classifying it as 'white noise' than speech - so this presents no setback.

Notice the strong similarity between the speech and traffic basis functions.

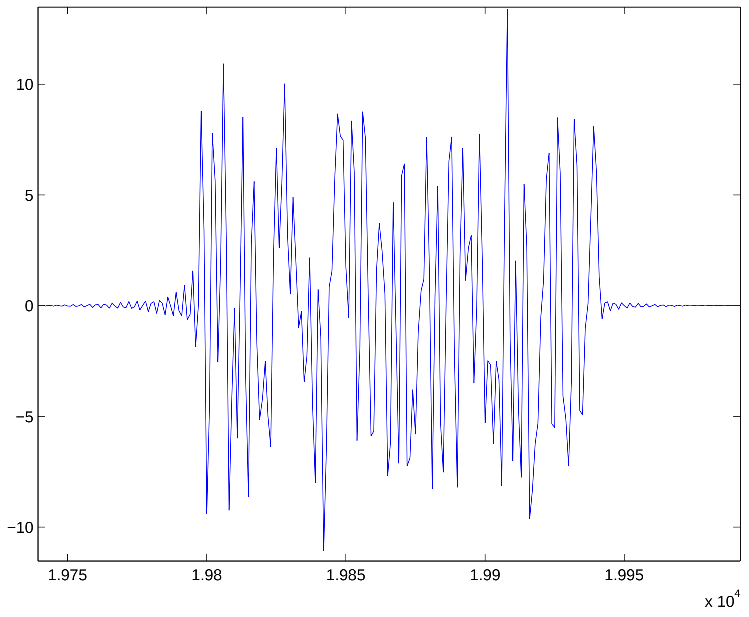

**Figure 12.25.** An example of a click. Notice the random nature of the signal.

Discrimination is still possible, both because there are differences in basis functions, but also because there is significant difference between the distributions of the sources. Figures figure 12.26 and figure 12.27 show a few distributions (histograms and fitted generalized Gaussians) for speech and traffic respectively.

With speech and traffic being the only types of signals used that have significant time-domain structure (as opposed to the clicks and white-noise types), music was tested as a third type of signal, known to be highly structured in time; specifically a performance Beethoven's moonlight sonata. Figure 12.29 shows the basis functions learnt by icaEXP when given 10000 samples from this piece. Note the periodic, phase-specific basis functions together with the frequency-only functions. The latter probably stem from the fact that piano notes (frequencies) often linger on substantially longer than several milliseconds.

### 12.5.2 Basis function interpretation

The basis functions learned from clean, mixed male/female speech clearly look like a good basis for a sparse representation of speech. In figure 12.28, the sub-figure in row 5, column 3 for instance represents voiced speech. Some (e.g. row 3, column 1) obviously correspond to some of the noisy, unvoiced parts of speech.

The functions have a wavelet-like structure, specific with respect to both frequency- and phase content. Some are reminiscent of Gabor filters (plane

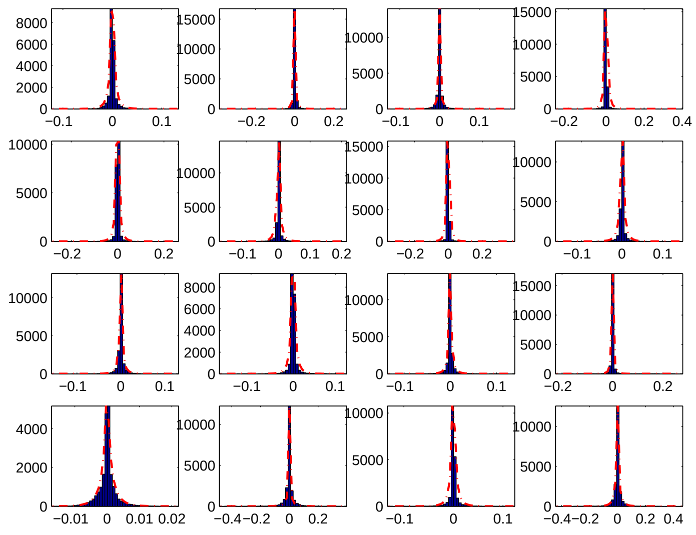
**Figure 12.26.** Some speech source distributions.

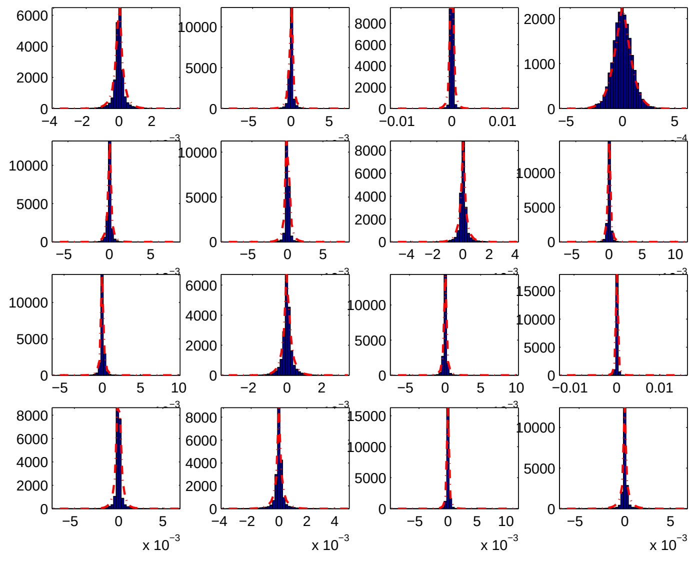
**Figure 12.27.** Some traffic source distributions.

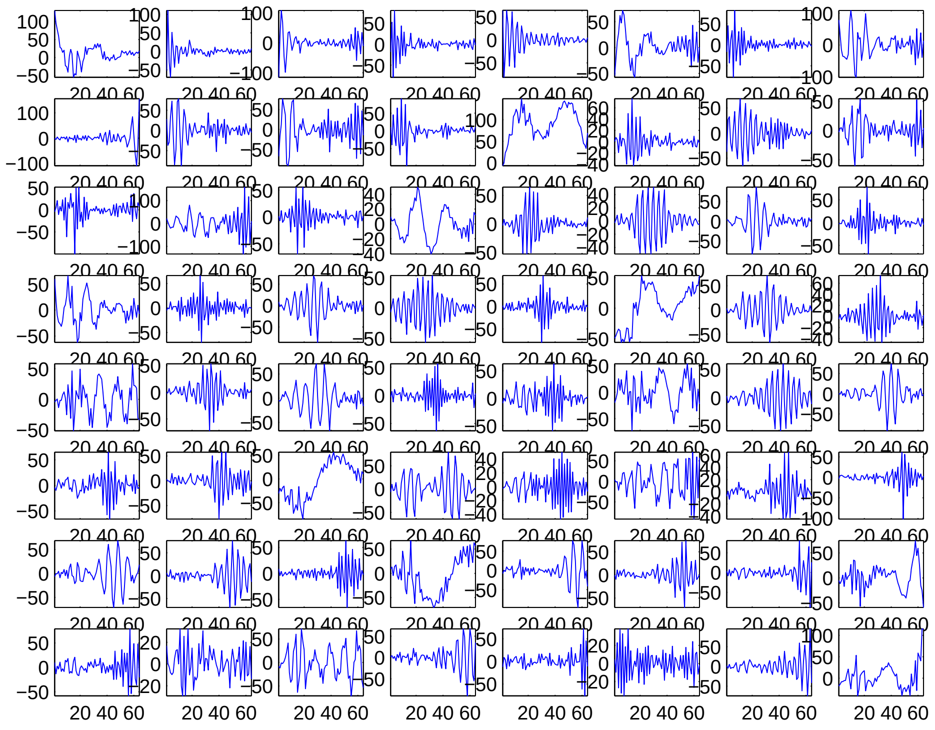

**Figure 12.28.** Basis functions learnt from a mixture of male and female speech.

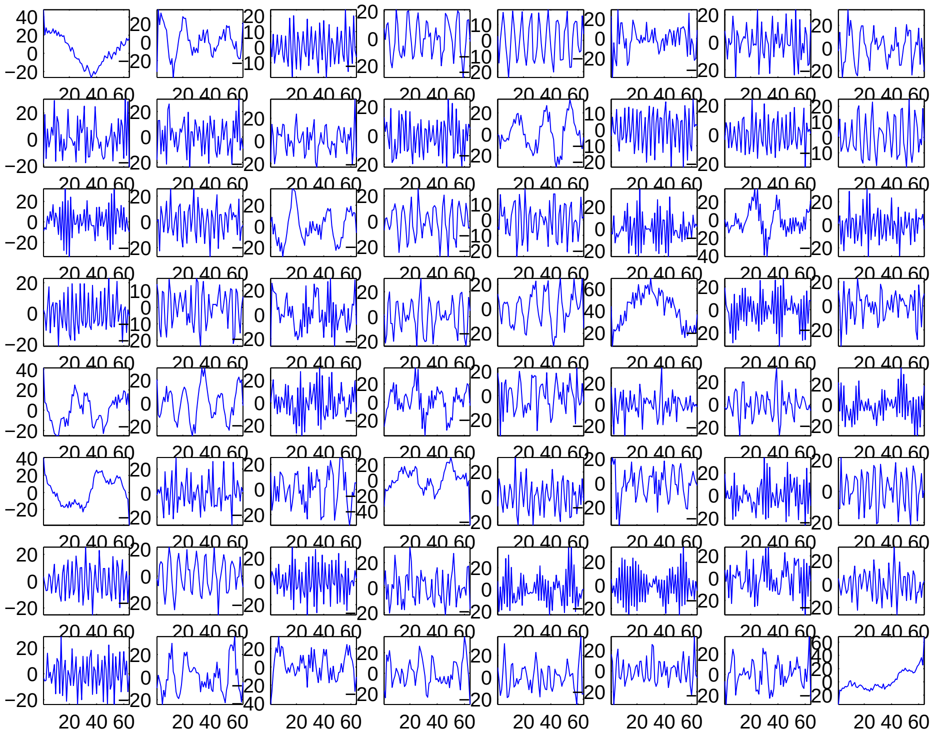

**Figure 12.29.** Basis functions learnt from Beethoven's moonlight sonata (piano music).

waves restricted by a Gaussian), and these very likely correspond to segments of voiced speech.

### 12.5.3 ICA model 1

This concerns the model of section 10.10.

Figure 12.30 shows the output (estimated $P(C_{VA}|x)$) from this VAD for white noise. Notice the sharp fall-offs of the signal, leading to the idea of implementing a holding scheme - shown in figure 12.31. The corresponding ROC curves are shown in figure 12.32 and figure 12.33. These show the potential gain from holding schemes and also that this ICA VAD is very good at detecting speech in white noise (as expected).

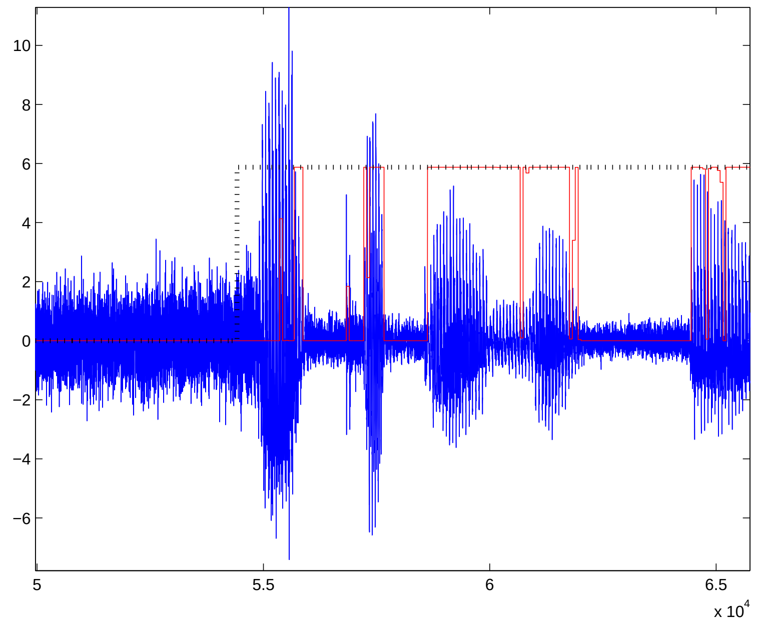

**Figure 12.30.** Output from 1st ICA VAD. Note the large holes in the signal - no hangover scheme is used. White noise, SNR 0. Also shown is the true VAD signal.

This model is surprisingly good at not detecting clicks as speech, as shown in figure 12.34. This is probably because the clicks rather resemble white noise (see figure 12.25).

Figure figure 12.35 shows the overall performance on white noise.

.

### 12.5.4 ICA model 2

This concerns the model of section 10.11. As shown in figure 12.37, model 1 is 'fooled' into believing that traffic is speech (probably because it looks more like

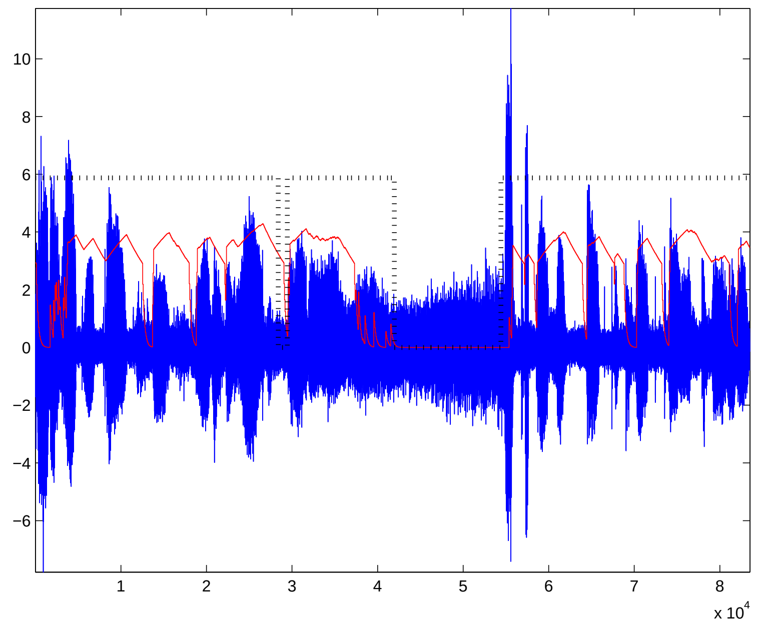

**Figure 12.31.** Same as figure 12.30, but now with a simple hang-over scheme. The 'holes' are covered with no noticeable spill-over to non-VAD regions. White noise, SNR 0.

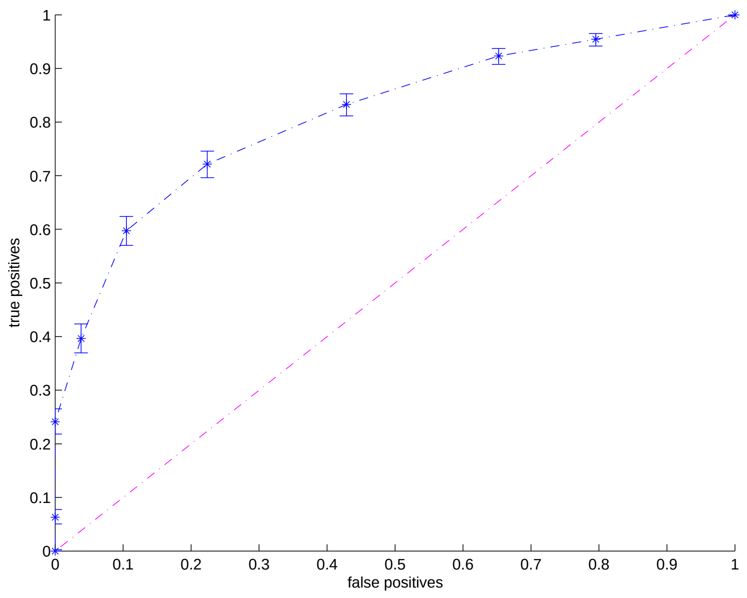

**Figure 12.32.** ROC curve corresponding to figure 12.30. White noise, SNR 0.

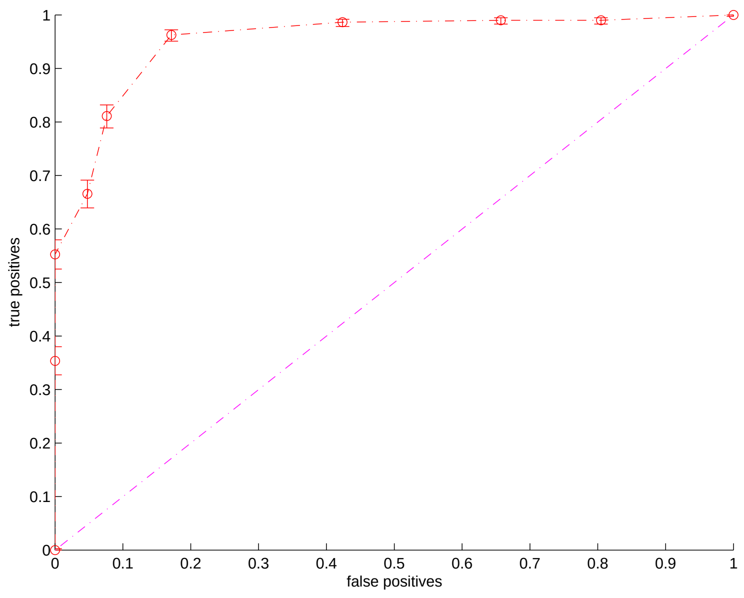

**Figure 12.33.** ROC curve corresponding to figure 12.31. Note the significant improvement compared to figure 12.32. White noise, SNR 0.

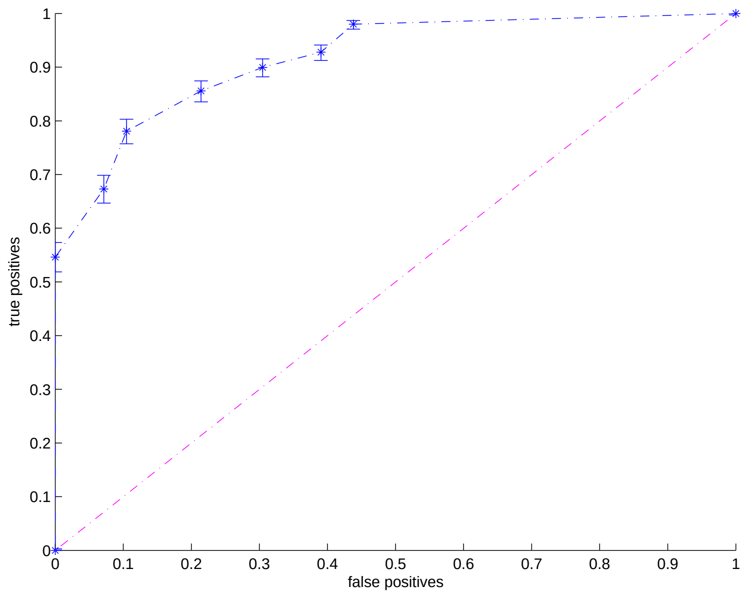

**Figure 12.34.** ICA model 1 is able to see that clicks are not speech - clicks noise, SNR 0.

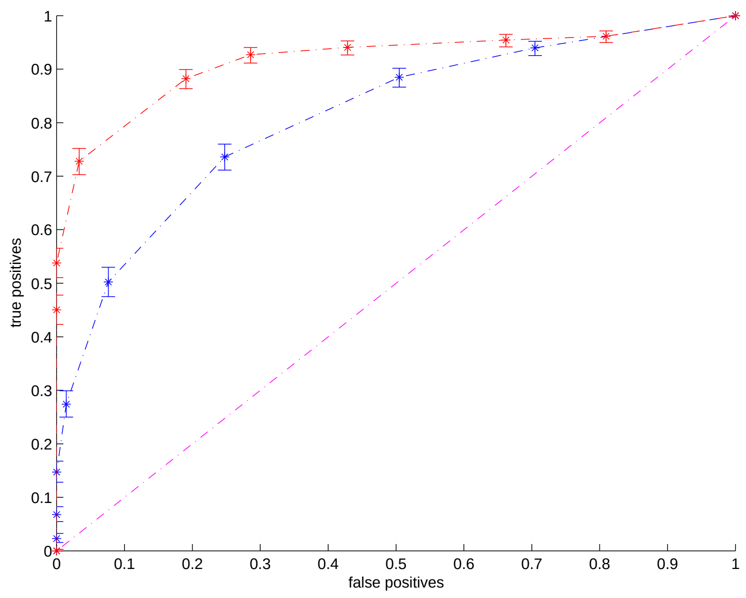

**Figure 12.35.** ICA model 1 on white noise (blue: SNR 0, red: SNR 10)

speech than white noise), while model 2 performs much better; it is often able to distinguish correctly. However, see also figure 12.36.

This model is no better at white noise than model 1 (they both include a white noise model in the same way).

### 12.5.5 ICA mixture model

It was attempted to apply this model to speech data, but the computation time needed was excessive.

Artificial signals were created, however, and tested on the mixture model algorithm. Figures figure 12.38 and figure 12.39 show that the algorithm was able to learn to correctly classify these data. This task was made harder by letting both classes have the same mean (zero).

Babble is not handled well by the above ICA models (not shown). It might have been handled to some extent by training specific babble models, but in general, babble is a very difficult setting for speech detection.

## 12.6 OTI

Figures figure 12.40 to figure 12.43 show the performance of the OTI VAD on different noise types.

This VAD is obviously not very good in white noise settings, and is especially troubled by clicks noise (figure 12.42, see also figure 12.45). However - and quite

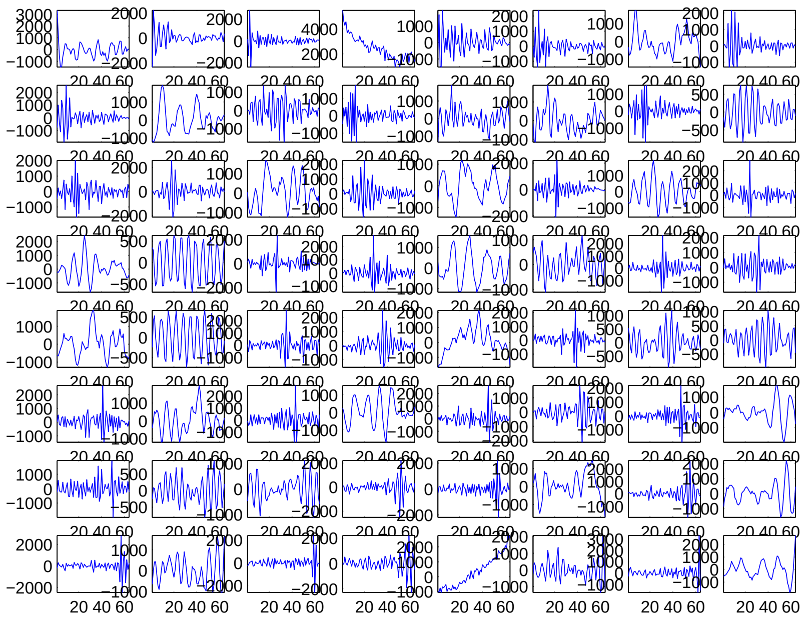

**Figure 12.36.** The mixing matrix found for traffic data. Notice the close resemblance to many of the speech basis functions; these are quite surprising and may explain some of the difficulty in detecting speech in traffic noise.

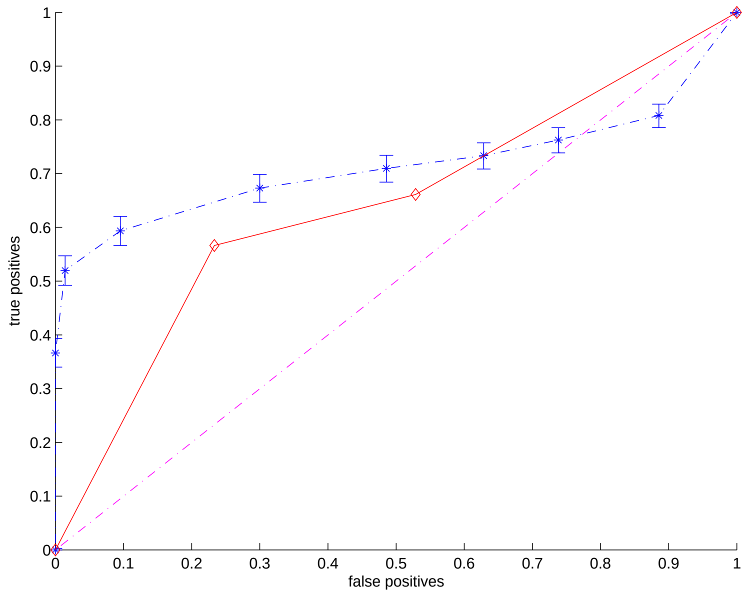

**Figure 12.37.** ICA model 1 and 2 compared. The latter handles traffic (to some degree, but not the difficulties towards the upper right), the former cannot. Traffic noise, SNR 0.

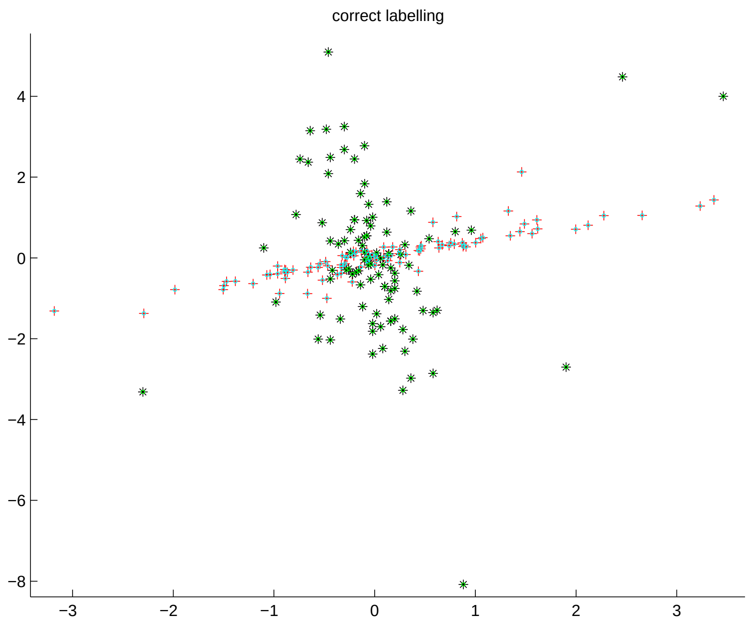

**Figure 12.38.** Correct classification of artificial data.

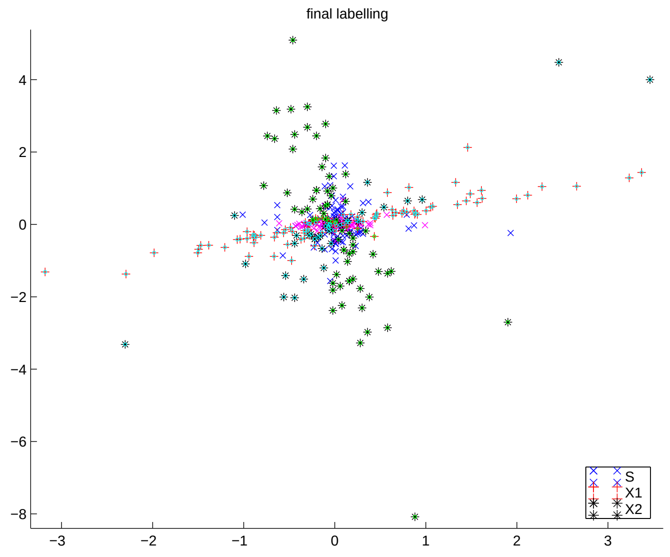

**Figure 12.39.** Estimated classification of the same artificial data as in figure 12.38 by the SIMalgorithm. Note that the central points are ambiguous and would be difficult to classify even by a human. The ones that are 'classifiable' have been classified correctly. (The blue and magenta points are the source estimates and can be ignored).

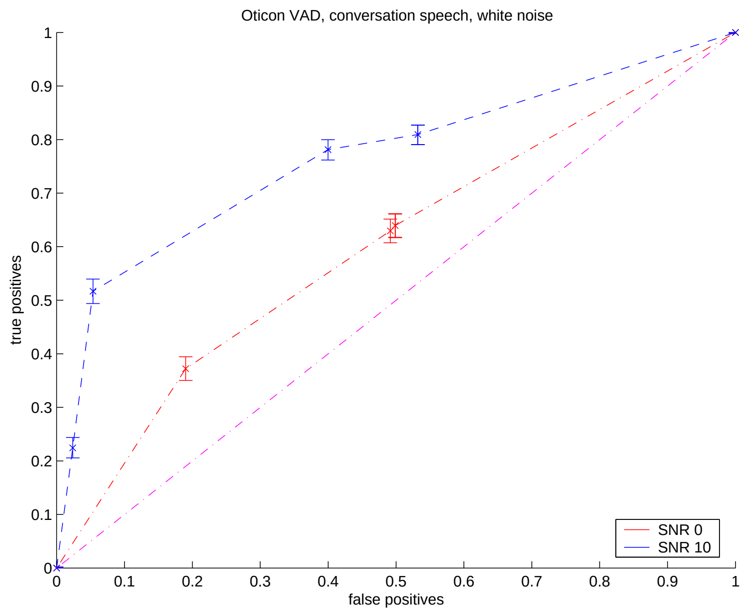

**Figure 12.40.** ROC for OTI VAD with white noise, conversation speech.

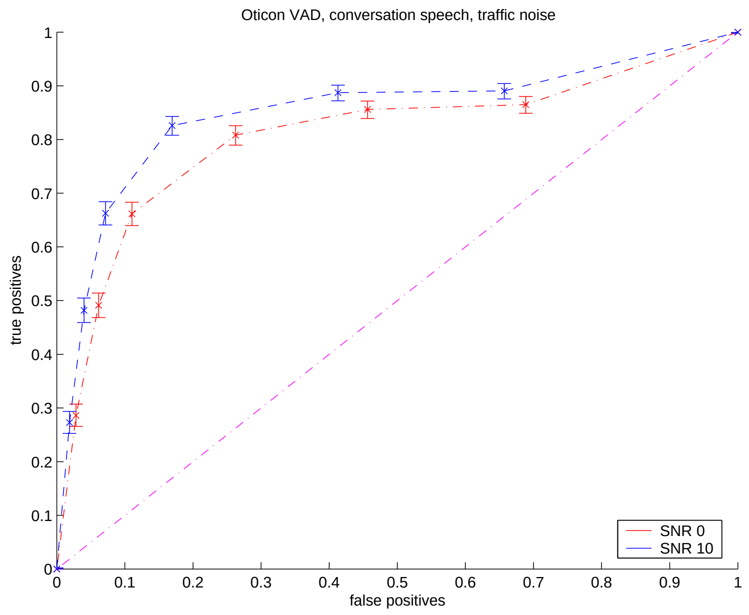

**Figure 12.41.** ROC for OTI VAD with traffic noise, conversation speech.

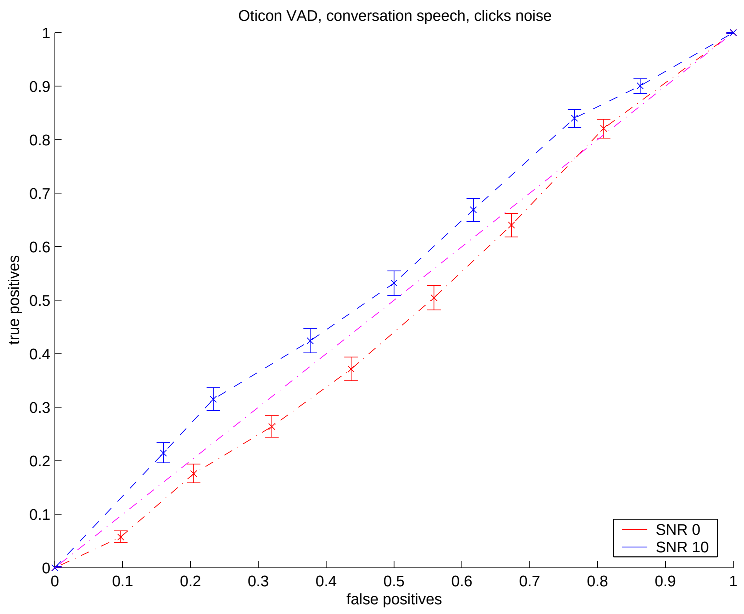

**Figure 12.42.** ROC for OTI VAD with clicks noise, conversation speech.

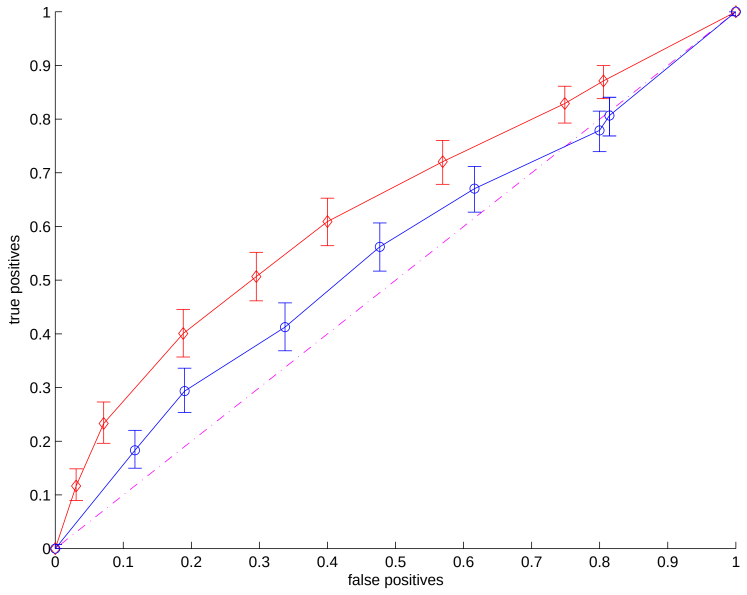

**Figure 12.43.** ROC for OTI VAD with babble noise, conversation speech.

surprisingly - it handles traffic noise very well (figure 12.41).

The output signal is quite 'peaky', see figure 12.44. This VAD would probably benefit greatly from a hang-over scheme of some kind, smoothing the sharp drop-offs of the signal.

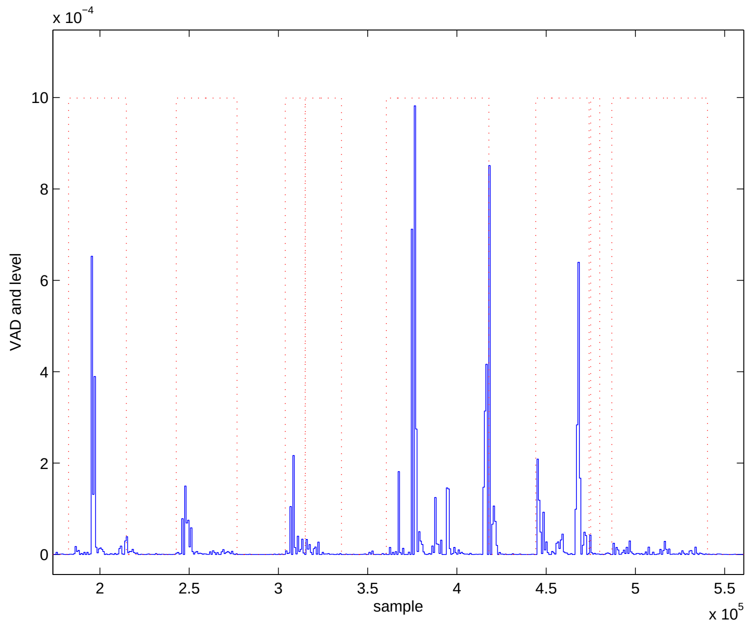

**Figure 12.44.** True VAD and OTI VAD output levels; traffic noise, but similar characteristics for all noise types. Note the 'peakedness' of the output level.

## 12.7 The ITU-T (G.729B) standard VAD

Figure 12.46 shows the performance of this VAD as ROC points for different noise types and SNR's. As mentioned in 8.2.1, only points on the ROC plane can be generated for this VAD. It will be compared with other VAD's in the following, so no analysis will be done here. Note however that it looks to be quite robust to both noise type and SNR.

## 12.8 Linear network and the ITU-T and OTI

The conclusion from the linear network experiments was that using all 36 mel filterbank outputs was a good choice, but it was also mentioned that it would not be fair to compare linear networks trained on a specific noise type with VAD's built for any noise type (such as the ITU-T and OTI VAD ). Therefore, in the following ROC curves, the following are compared:

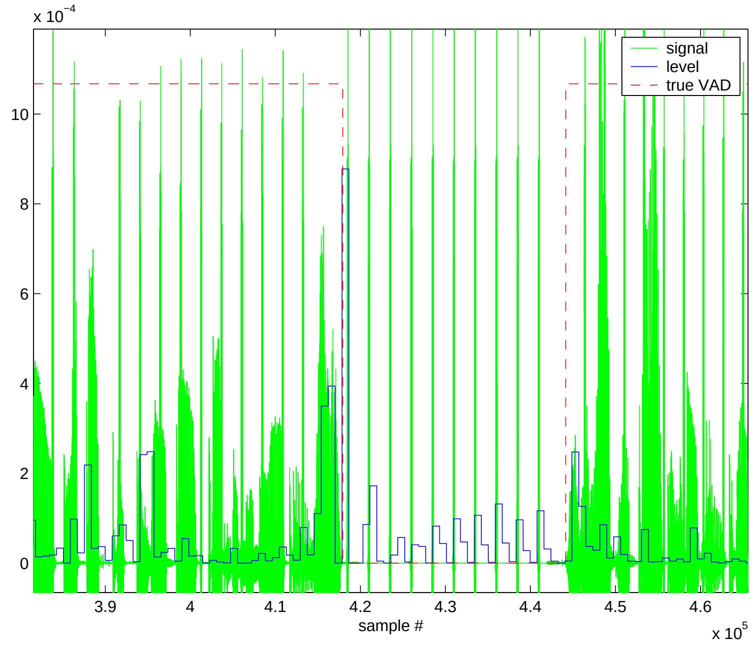

**Figure 12.45.** Problems with click noise. The simultaneous onset in many frequencies for this noise type fools the OTI VAD ; output levels in the non-VAD regions are comparable to those in the VAD regions.

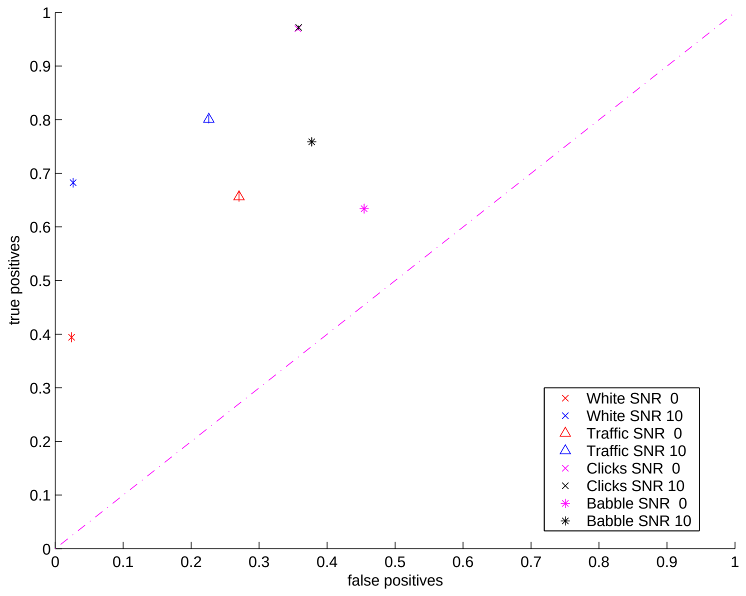

**Figure 12.46.** ROC points for ITU-T VAD (conversation type speech).

- Linear network, 36 cross-correlations, specifically trained on a particular noise type (Mel36)
- Linear network, 36 cross-correlations, trained on a mixture of all noise type at SNR 0 (Mel36All-0)
- Linear network, 36 cross-correlations, trained on a mixture of all noise type at SNR 10 (Mel36All-10)
- The ITU-T VAD
- The OTI VAD

The first is included to illustrate the importance of the noise type in determining performance - it will not be included in the comparison analysis for the reasons given above. See figures 12.47 to 12.54.

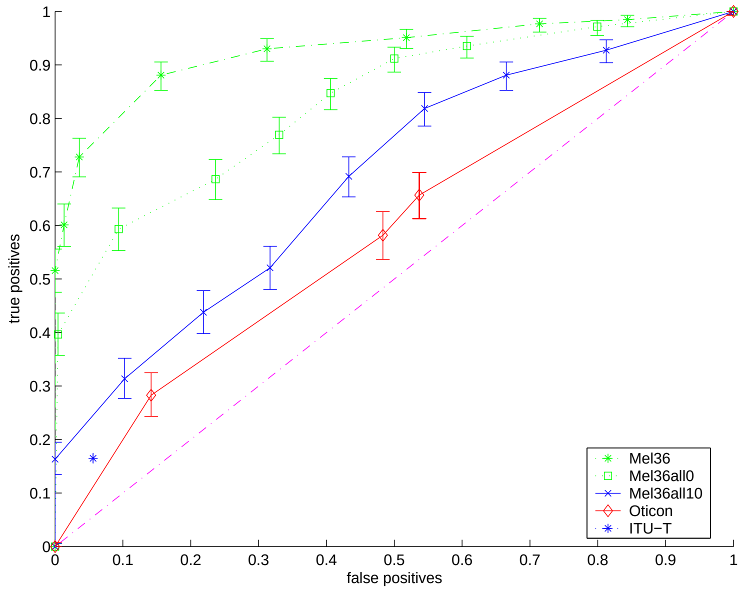

**Figure 12.47.** White noise, SNR 0

Inspecting these figures, it is possible to conclude with some certainty which VAD's best handle the different sound environments.

For white noise, Mel36All-0 performs the best. OTI and ITU are somewhat similar, although ITU is very much 'on the safe side' (low FP and TP) at SNR 0, and OTI is better than ITU at SNR 10.

For traffic noise, OTI is the overall winner. Mel36All-0 ties second place with ITU.

For clicks noise, OTI fails completely (in fact, inverting the output signal would produce a better VAD!). Again, Mel36All-0 performs quite well, but ITU is the best VAD for this environment.

Note for all of the above that the specifically trained linear network (Mel36) performs significantly better than the general ones (Mel36All-0 and -10).

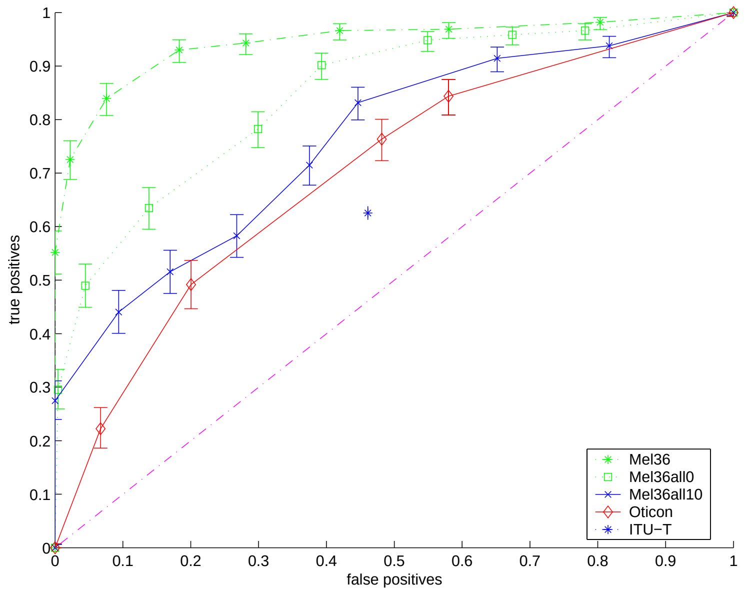

**Figure 12.48.** White noise, SNR 10

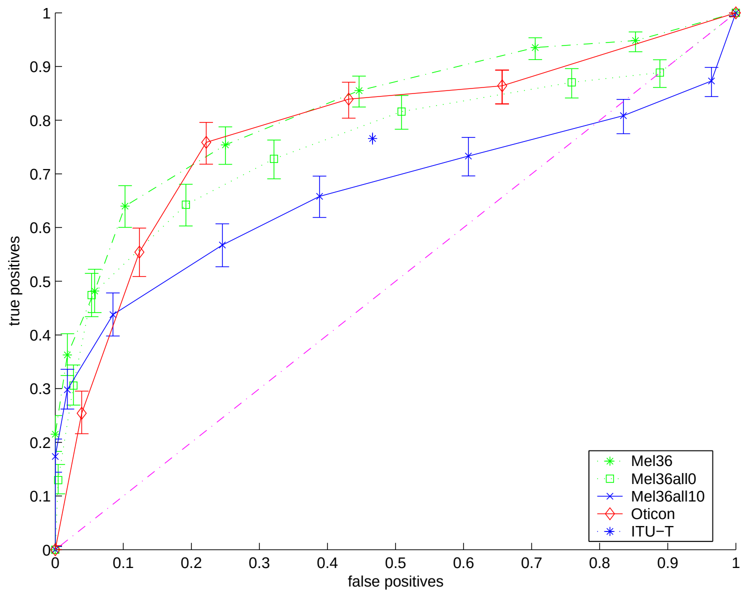

**Figure 12.49.** Traffic noise, SNR 0

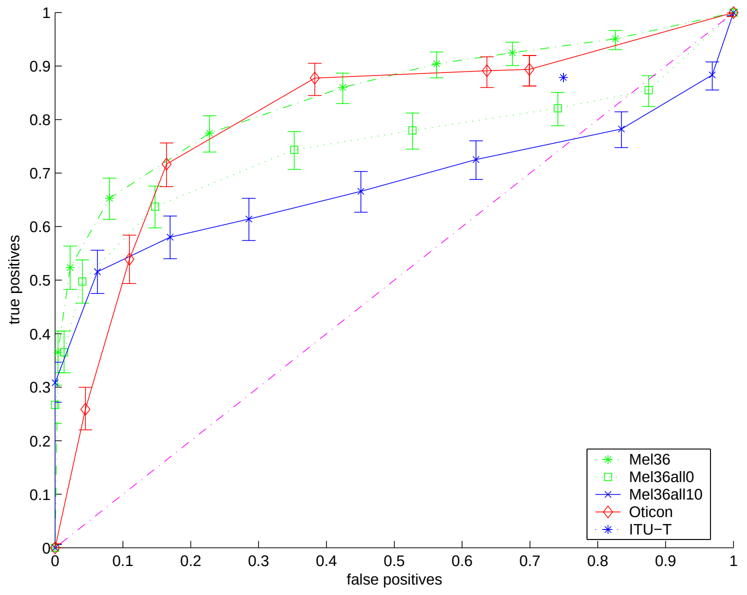

**Figure 12.50.** Traffic noise, SNR 10

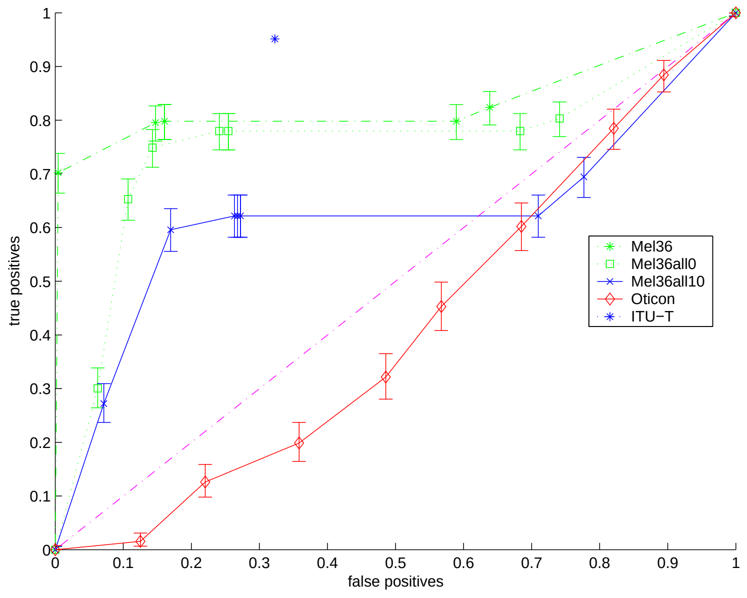

**Figure 12.51.** Clicks noise, SNR 0

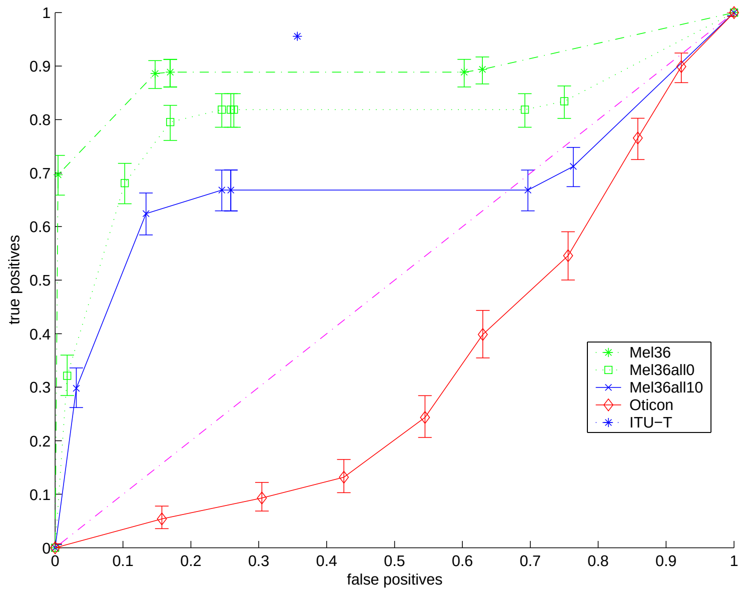

**Figure 12.52.** Clicks noise, SNR 10

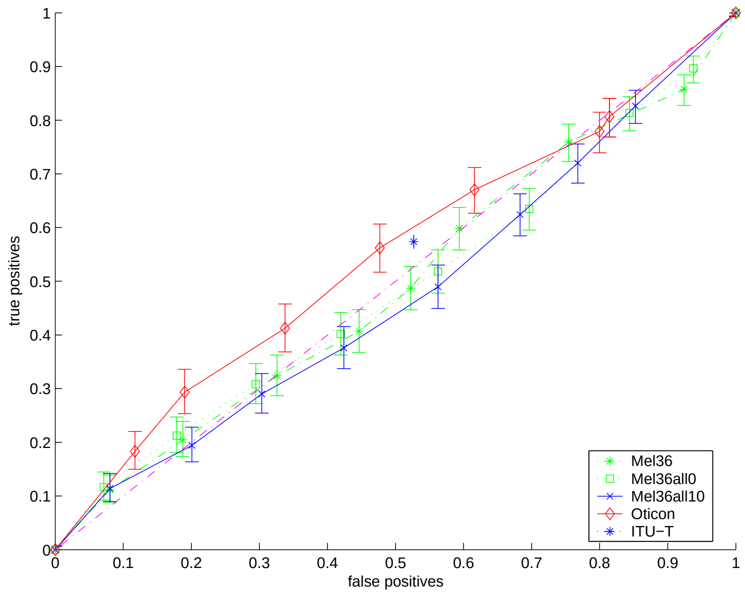

**Figure 12.53.** Babble noise, SNR 0

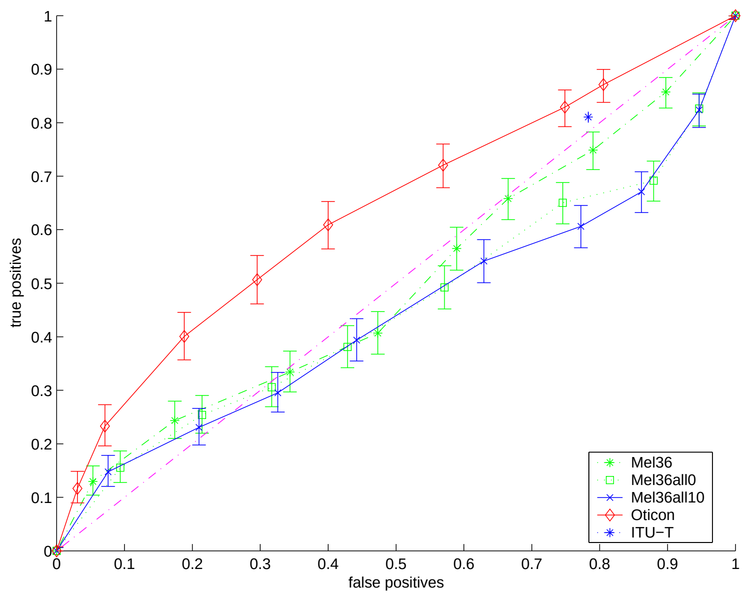

**Figure 12.54.** Babble noise, SNR 10

For babble noise, no VAD is impressive. Still, OTI has some success at SNR 10.

## 12.9 Comparing with the ICA models

Again, comparing the ITU-T VAD to other VAD's is difficult since only one point on the ROC plane is available for the ITU-T VAD, so those comparisons must be done with this reservation in mind.

For white noise, figure 12.35 shows that at SNR 10, the ICA models are actually the best ones.

Including the ICA models, for 'clicks' noise, the ITU-T VAD must still (grudgingly) be declared the winner (figures 12.34 and 12.51).

This VAD uses rather sophisticated hang-over methods, so the ICA models (1 and 2) were improved by implementing a simple hang-over scheme to see if they could then match the ITU-T VAD. The result is shown in figure 12.55; the result for SNR 10 is very similar. The ITU-T VAD still seems to have an edge and it is somewhat surprising that it handles this type of noise so well.

For traffic noise, the ICA models are not as good as OTI, although the difference is not great (figures 12.37, 12.49 and 12.50) and the ICA models actually perform better in the low-FP, low-TP area.

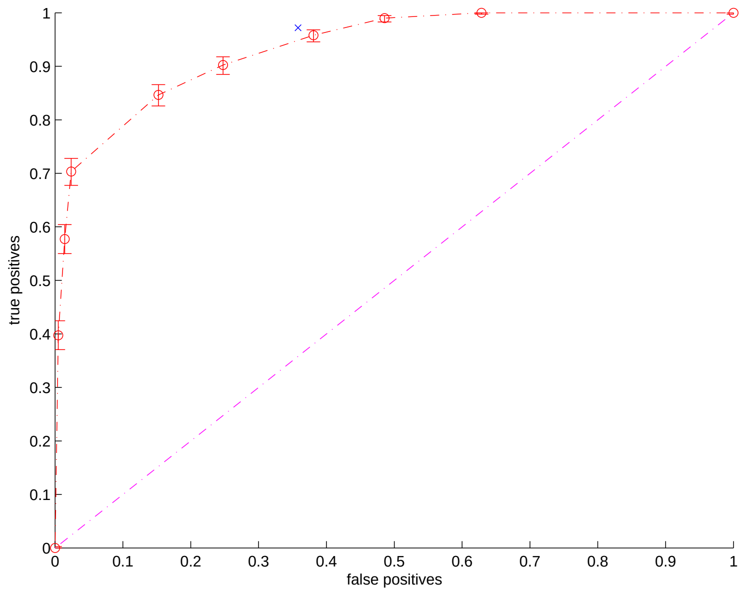

**Figure 12.55.** ICA model 1 improved with a simple hang-over scheme is still slightly worse than the ITU-T for clicks noise (SNR 0).

# Chapter 13

# Discussion and conclusion

This chapter discusses the findings made during this project on the main challenge, which was to build a single system that is robust to all noise types and to both low and high SNR.

First, this work has clearly demonstrated the crucial importance of taking the type of noise into account when developing and testing speech detection algorithms.

Regarding feature extraction, the cross-correlation features have been proven in this work to be very useful and able to be used for noise robust VAD algorithms on their own.

As the systems stand at the end of this project, the linear network using 36 cross-correlations between squared mel-scale filterbank outputs, trained on a combination off all noise types at SNR 0, would be chosen as the best overall VAD. It has no outright weakness and is quite robust to both noise type and SNR. The networks trained at SNR 0 might perform better than those trained at SNR 10 as they are more forced, so to speak, to learn the most appropriate parameters (weights).

The ITU-T VAD handles the clicks noise type very well, but is not well suited for white noise environments, where it operates very cautiously at low SNR.

The OTI VAD ([4]) performs surprisingly well in traffic noise, but has grave trouble with transient (clicks type) noise, as expected.

The ICA models are very good in white noise environments, but have some trouble discriminating between traffic and speech. This may be due to using too short time-domain segments in those models. They are also very good at detecting speech in the clicks noise type, only slightly outperformed by the ITU-T VAD .

All in all, each VAD has strong points compared with all the others, so the actual choice in a practical situation would have to depend on the expected sound environment that the VAD is to operate in.

Although very little optimization was done on the implementations, the linear network classifiers are generally significantly faster than the ICA classifiers. Still, it should be possible to create versions of both that could detect speech in real time.

## 13.1 Future improvements and research

An obvious research direction to take is to investigate the many different features that are suggested by others in audio signal processing. The knowledge gained here could then be applied to improve both the linear and the ICA models.

Also, investigation of other noise types should be done, e.g. music.

A continuation of the linear network method approach would be to use a multi-layer perceptron instead (see [3]). This is a more powerful, non-linear learning system and would certainly be an appropriate next step to research.

From the experimental results one thing is very clear, namely that when tested on a particular noise type, linear network classifiers trained with that particular noise type have a tremendous advantage over classifiers trained on other noise types. The SNR is similarly important, although it seems that training on low SNR is generalizable somewhat to better SNR conditions. Therefore, it would be interesting to see, if it was possible to train classifiers that would estimate the probability of the presence of the different noise types. These estimates could then be used to weigh the outputs of each of the speech detector classifiers (one for each noise type, possibly also different ones for high and low SNR), producing a better and more robust combined VAD.

The ICA framework can be expanded in a number of ways. The ICA classifiers could probably be made more powerful by learning separate models for voiced and unvoiced speech - another example of the inclusion of prior knowledge. Chiefly, however, the supervised ICA mixture model (although only touched upon in this work) is of interest and is deemed to hold some promise in the VAD context. Single-channel source separation is another next step for the ICA approach, in a different but interesting direction.

In conclusion, the goal of building a VAD robust to different types of noise and SNR's can be said to be reached to a significant degree. The two main contributions of this work are the use of principled learning methods with a particular set of features (filterbank cross-correlations) and the use of ICA models for speech detection. Both approaches have produced useful results, and both hold potential for further improvement, some options for which have been laid out.

# Bibliography

[1] H. Su A. Benyassine, E. Shlomot, *Itu-t recommendation g.729 annex b: a silence compression scheme for use with g.729 optimized for v.70 digital simultaneous voice and data applications*, IEEE Communication Magazine (1997).

[2] Anthony J. Bell and Terrence J. Sejnowski, *An information-maximization approach to blind separation and blind deconvolution*, Neural Computation **7** (1995), no. 6, 1129–1159.

[3] Christopher M. Bishop, *Neural networks for pattern recognition*, Oxford University Press, 1995.

[4] C. Ludvigsen C. Elberling, M. Ekelid, *A method and an apparatus for classification of a mixed speech and noise signal*, International application published under the patent cooperation treaty (1991).

[5] Khaled El-Maleh and Peter Kabal, *Comparison of voice activity detection algorithms for wireless personal communications systems*, 1997.

[6] D. Ellis and J. Bilmes, *Using mutual information to design feature combinations*, Int. Conf. on Spoken Language Processing, 2000, pp. 79–82.

[7] Nicholas Evans, *Time-frequency quantile-based noise estimation*, Proc. EUSIPCO, 2002.

[8] Te-Won Lee Gil-Jin Jang and Yung-Hwan Oh, *Single channel signal separation using time-domain basis functions*, June 2003.

[9] Allamanche E.-Hellmuth O. Herre, J., *Robust matching of audio signals using spectral flatness features*, 2001 IEEE Workshop on the Applications of Signal Processing to Audio and Acoustics (2001), 127–130.

[10] Shun ichi Amari, *Natural gradient works efficiently in learning*, Neural Computation **10** (1998), no. 2, 251–276.

[11] W. M. Fisher J. G. Fiscus D. S. Pallett J. S. Garofolo, L. F. Lamel and N. L. Dahlgren, *The darpa timit acoustic-phonetic continuous speech corpus*, Oct. 1990.

[12] Volker Stahl Jan Stadermann and Georg Rose, *Voice activity detection in noisy enviroments*, September 2001.

[13] Wayne Ward Jianping Zhang and Bryan Pellom, *Phone based voice activity detection using online bayesian adaptation with conjugate normal distributions*, May 2002.

[14] M. Joho, H. Mathis, and R. Lambert, *Overdetermined blind source separa-*

*tion: Using more sensors than source signals in a noisy mixture*, 2000.

[15] J. Karhunen, *Neural approaches to independent component analysis and source separation*, 1996.

[16] J. Larsen, A. Szymkowiak, and L. Hansen, *Probabilistic hierarchical clustering with labeled and unlabeled data*, 2001.

[17] T. Lee and M. Lewicki, *The generalized gaussian mixture model using ica*, Proceedings of the Internationall Workshop on ICA (2000), 239–244.

[18] Thomas P. Minka, *Expectation-maximization as lower bound maximization*, 1998.

[19] R.D. Patterson, *Auditory images: How complex sounds are represented in the auditory system*, J. Acoust. Soc. Japan (E) **21** (2000), 183–190.

[20] V. Peltonen, *Computational auditory scene recognition*, 2001.

[21] J. W. Picone, *Signal modeling techniques in speech recognition*, September 1993, pp. 1215–1247.

[22] J.M. Lovekin R.E. Yantorno, K.R. Krishnamachari, *The spectral autocorrelation peak valley ratio (sapvr) - a usable speech measure employed as a co-channel detection system*, 2000.

[23] Sam T. Roweis, *One microphone source separation*, NIPS, 2000, pp. 793–799.

[24] F. Berthommier S. Skorik, *On a cepstrum-based speech detector robust to white noise*, Specom2000 (2000).

[25] H. Sameti, H. Sheikhzadeh, L. Deng, and R. Brennan, *Hmm-based strategies for enhancement of speech embedded in non-stationary noise*, IEEE Transactions on Speech and Audio Signal Processing (1998), 6(5):445–455.

[26] E. Scheirer and M. Slaney, *Construction and evaluation of a robust multi-feature speech/music discriminator*, Proc. ICASSP '97 (Munich, Germany), 1997, pp. 1331–1334.

[27] J.C. Segura, M.C. Benítez, A. de la Torre, and A.J. Rubio, *Feature extraction combining spectral noise reduction and cepstral histogram equalization for robust asr.*

[28] Malcolm Slaney, *Auditory toolbox: A matlab toolboxfor auditory modeling work*, 1998.

[29] J. Sohn, N. Kim, and W. Sung, *A statistical model-based voice activity detection*, (1999).

[30] G. Williams and D. Ellis, *Speech/music discrimination based on posterior probability features*, 1999.

# Appendix A

# TIMIT processing

This appendix describes the TIMIT database in more detail, along with a discussion of phonemes and definitions of voiced and unvoiced speech. The extraction of data from the database is also briefly described.

## A.1 Phonemes

Phonemes are the 'building blocks' of speech - they are the semi-stationary segments that make up each spoken word. In order to distinguish between voiced and unvoiced speech, it is necessary to examine the speech signals at the phoneme level.

Overall, speech consists of two types of sounds: consonants and vowels. Vowels are all voiced by definition. Consonants can be voiced or unvoiced.

Consonants involve interrupting the air that comes out of your mouth; vowels are made by opening the mouth and letting air come out freely.

There are two basic ways of making consonants: voiced and unvoiced. Voiced consonants involve a vibration of the vocal cords. Unvoiced consonants involve no vibration of the vocal cords.

Vowels are made by opening the mouth and letting air come out while the vocal cords vibrate.

There are five types of consonants: stops, fricatives, nasals, affricates, and semivowels. Nasals and semivowels are always voiced while stops, fricatives and affricates can be voiced or unvoiced.

Tables A.1 and A.2 list the phoneme symbols used in the TIMIT corpus as they correspond to the above.

However, even though table **??** groups the phonemes correctly as voiced or unvoiced, 'z', 'zh' and 'dh' may be included in the unvoiced set for machine learning purposes, as they were found to look (in the spectral domain) and sound similar to the other unvoiced phonemes, at least as found and labelled in the TIMIT corpus. These sets will still be referred to as 'voiced' and 'unvoiced'.

## A.2 Extraction of TIMIT data

Initially, much effort was used in trying to read the special 'NISP' format that all TIMIT sound data are stored in. Eventually, `Matlab` functions were found to be available on the web and .wav data could then be extracted.

In addition to the speech itself, a VAD signal containing the true (correct) labelling of each sample was generated by extracting the phoneme codes from phoneme files. Further, a Voiced signal was extracted that contains the true labelling of each sample as either voiced or unvoiced. This signal is only relevant and used for samples that are labelled as containing speech.

Four files are associated with each sentence: a .wav file contains the speech data, sampled at 16kHz. A text file contains a transcription of the words in the sentence. A word file contains sample(time)-aligned word transcriptions and the phoneme file contains the sample-aligned phoneme transcriptions.

| **Voiced phonemes** | | |
|---|---|---|
| Phoneme type | TIMIT symbols | Example word |
| Stops | b,bcl | bee |
| | d,dcl | day |
| | g,gcl | gay |
| | dx | muddy |
| | q | bat |
| Affricates | jh | joke |
| Fricatives | v | van |
| | z | zone |
| | zh | azure |
| | dh | then |
| Nasals | m | mom |
| | n | noon |
| | ng | sing |
| | em | bottom |
| | en | button |
| | eng | washington |
| | nx | winner |
| Semivowels,Glides | l | lay |
| | r | ray |
| | w | way |
| | y | yacht |
| | hh | hay |
| | hv | ahead |
| | el | bottle |
| Vowels | iy | beet |
| | ih | bit |
| | eh | bet |
| | ey | bait |
| | ae | bat |
| | aa | bott |
| | aw | bout |
| | ay | bite |
| | ah | but |
| | ao | bought |
| | oy | boy |
| | ow | boat |
| | uh | book |
| | uw | boot |
| | ux | toot |
| | er | bird |
| | ax | about |
| | ix | debit |
| | axr | butter |
| | ax-h | suspect |

**Table A.1.** TIMIT voiced phoneme transcriptions.

| **Unvoiced phonemes** | | |
|---|---|---|
| Stops | p,pcl | pea |
| | t,tcl | tea |
| | k,kcl | key |
| Affricates | ch | choke |
| Fricatives | s | sea |
| | sh | she |
| | f | fin |
| | th | thin |

**Table A.2.** TIMIT unvoiced phoneme transcriptions.

# Appendix B

# Recursive estimation

Often, some measure needs to be calculated for at a particular time (sample) that is based on that sample and several previous samples. If this is to be calculated for every single sample, direct calculations will be redundant and inefficient as the windows will be overlapping by all but one sample. Instead of calculating a needed measure at each time step or frame by calculating on a window of the signal, 'running' calculation may be done instead. If this is possible, then it is much faster, since the value for each time step or frame can be calculated from the last value only by a simple calculation, eliminating the need for calculating over the entire window each sample. This is done in a recursive manner.

With an exponentially decaying window, the estimation can be written as

$$v(n) = k \sum_{k=0}^{n} s(n) \lambda^{n-k}$$

where $v(n)$ is the estimate one wants to estimate and $s(n)$ is the actual current value of the measure. For instance, when estimating the signal's mean, $v(n)$ would be the mean estimate based on the current and older samples and $s(n)$ would be the current value of the signal, $x(n)$. This means that the current estimate should take previous samples into account, but weighing 'older' samples with exponential decay. This type of window is often appropriate for estimation.

For variance estimation, it is necessary to also estimate the signal mean, $\mu_x(n)$, as $s(n)$ is then

$$(x(n) - \mu_x(n))^2$$

The mean and variance can of course be estimated simultaneously.

The needed normalization factor can be found by setting $s(n) = 1$ for all n, in which case $v(n)$ should also become 1 asymptotically:

$$\begin{aligned}
V(n) &= k\sum_{k=0}^{n} 1 \cdot \lambda^{n-k} \\
&= k\frac{1-\lambda^{n+1}}{1-\lambda} \\
&= 1 \\
\Leftrightarrow k &= \frac{1-\lambda}{1-\lambda^{n+1}}
\end{aligned}$$

using

$$\sum_{k=0}^{n} \lambda^{n-k} = \sum_{k=0}^{n} \lambda^{k}$$

and

$$\sum_{k=0}^{n} 1 \cdot \lambda^{k} = \frac{1-\lambda^{n+1}}{(1-\lambda)}$$

which is known (and easily shown by recursive induction).
From this can be derived a simplified expression for setting $v(n)$. First:

$$\begin{aligned}
V(n) &= \frac{1-\lambda}{1-\lambda^{n+1}}\sum_{k=0}^{n} S(k)\lambda^{n-k} \\
&\backsimeq (1-\lambda)\sum_{k=0}^{n} S(k)\lambda^{n-k}
\end{aligned}$$

as

$$\lim_{n\to\infty} \lambda^{n} = 0, -1 < \lambda < 1$$

Similarly,

$$\begin{aligned}
V(n-1) &= \frac{1-\lambda}{1-\lambda^{n}}\sum_{k=0}^{n-1} S(k)\lambda^{(n-1)-k} \\
&\backsimeq (1-\lambda)\sum_{k=0}^{n-1} S(k)\lambda^{(n-1)-k} \\
&= (1-\lambda)\sum_{k=0}^{n-1} S(k)\lambda^{n-k}\lambda^{-1}
\end{aligned}$$

$v(n)$ can thus be expressed with $v(n-1)$ as

$$V(n) \backsimeq (1-\lambda)\big(\sum_{k=0}^{n-1} S(k)\lambda^{n-k} + s(n)\big)$$

and so

$$V(n) \backsimeq \lambda V(n-1) + (1-\lambda)S(k)$$

The closeness of the approximation depends on the number of samples and the value of lambda. With the values actually used for the current system, the error is negligible.

The normalization used in the preprocessing step of the current system uses a slightly modified version of this in order to speed things up. Instead of working sample-by-sample, it works block-by-block so that $n$ represents the block index and $s(n)$ is the mean (say variance) for the current block. This eliminates a number of multiplications in the order of the block size, at the expense of a courser normalization (each estimate being constant within each block), see figure 3.8.

The corresponding equation for a rectangular window of length L is

$$V(n) = V(n-1) + \frac{1}{N}(S(k) - S(k-L+1))$$

but is not used in this project.

For the autocorrelation, $S(k) = r_{xx}(k,\tau) = x(k)x(k-\tau)$.

The recursion formula would then be applied for all relevant $\tau$'s.

# Appendix C

# Software

More than 400 `Matlab` functions, helper functions and scripts were written in the course of this project. Some of these are mentioned in the report in connection with pseudo-code descriptions as they form key parts of the implementation of various algorithms. A selection from all these is on the CD that is available with this report (or from the author, dj@imm.dtu.dk).

The most important folders are:

## ANN

Functions for the linear neural network; training, pruning etc.

## experiments

Functions for running and analyzing experiments.

## ICA

Functions specific to Independent Component Analysis methods.

## myfunctions

Helper functions, used by many of the other functions, e.g. `roc.m` for calculating ROC curves.

## TIMIT

Functions for reading and extracting TIMIT data.

## VADs

Several linear network VAD's implemented as separate functions.

# Appendix D

# Additional figures

This appendix contains additional figures describing the results of the experiments (see chapter 12). Any observations can be found in the caption to each figure.

## D.1 Pruning

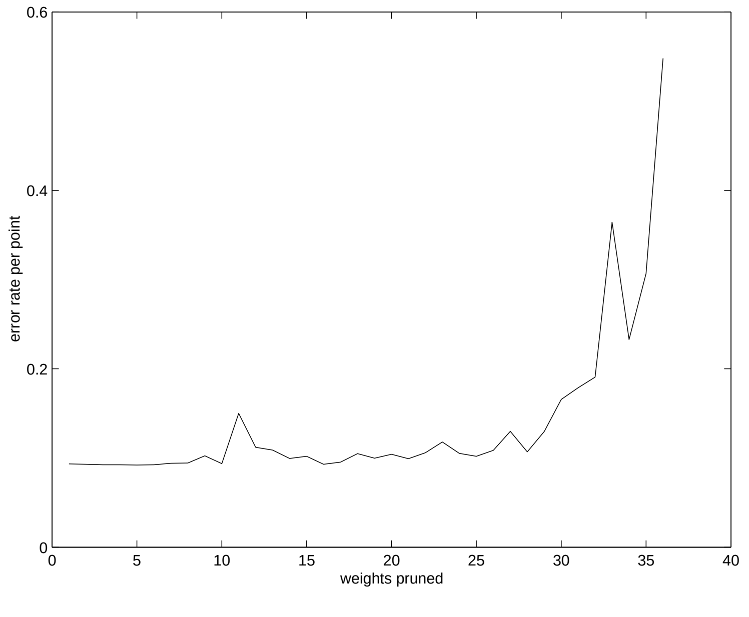

**Figure D.1.** Validation error as a function of number of weights pruned. White noise, SNR 10.

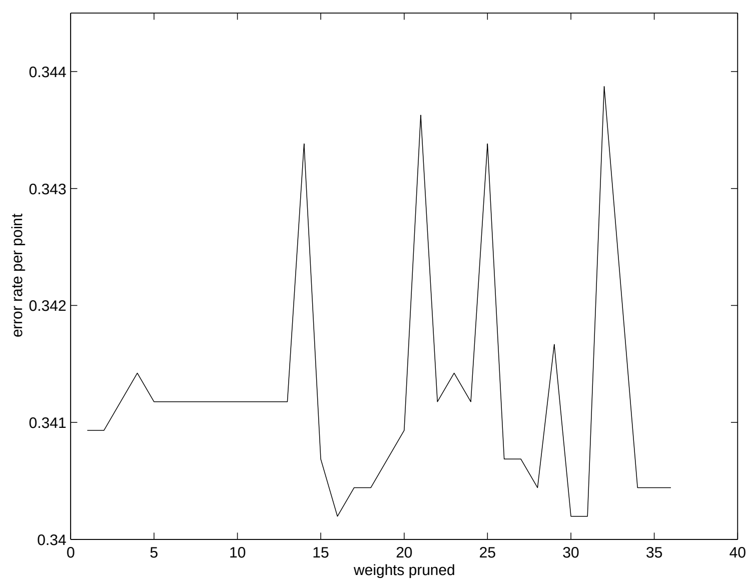

**Figure D.2.** Validation error as a function of number of weights pruned. Note the scale; the fluctuations are not large. Traffic noise, SNR 0.

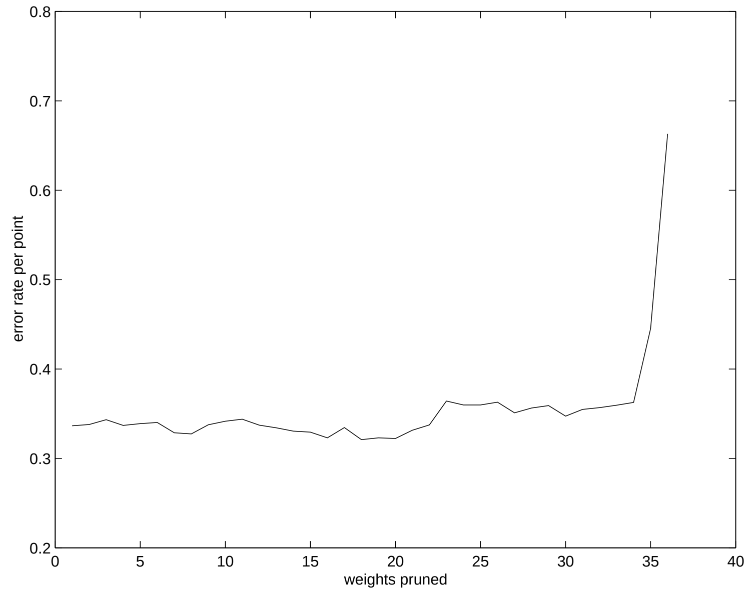

**Figure D.3.** Validation error as a function of number of weights pruned. Clicks noise, SNR 0.

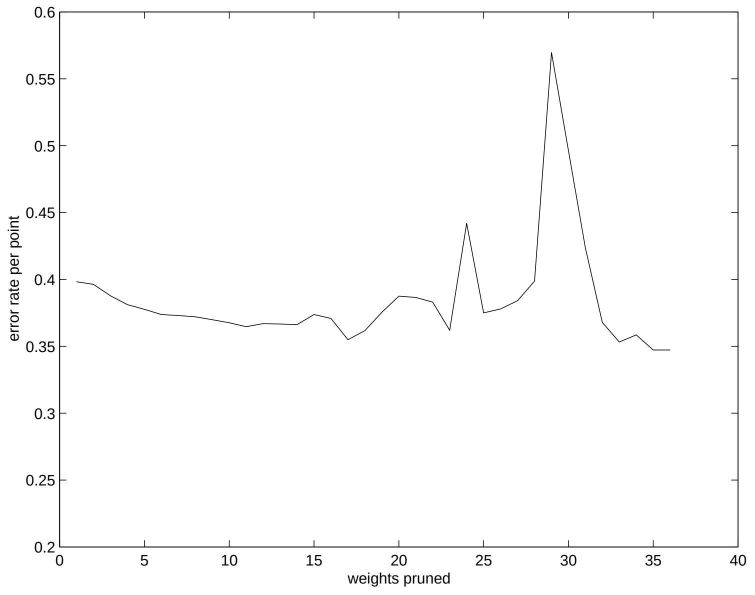

**Figure D.4.** Validation error as a function of number of weights pruned. Babble noise, SNR 0.

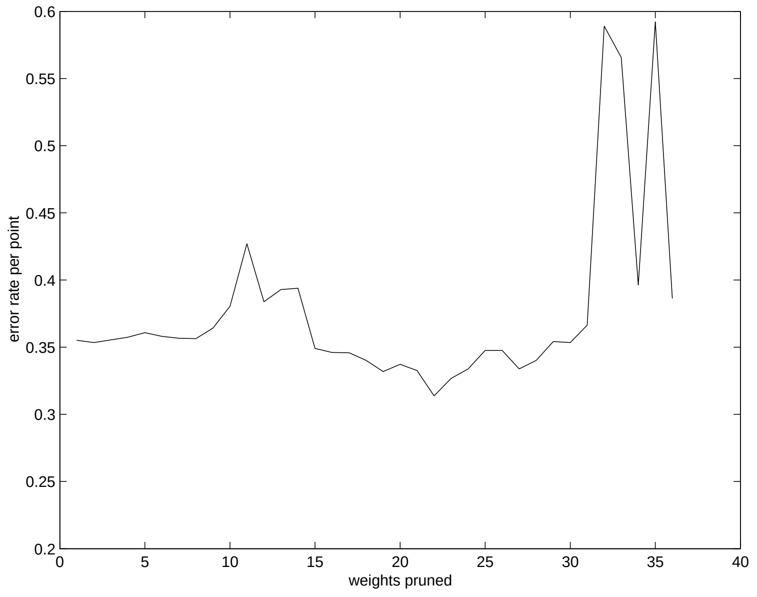

**Figure D.5.** Validation error as a function of number of weights pruned. Mix of all noise types, SNR 0.

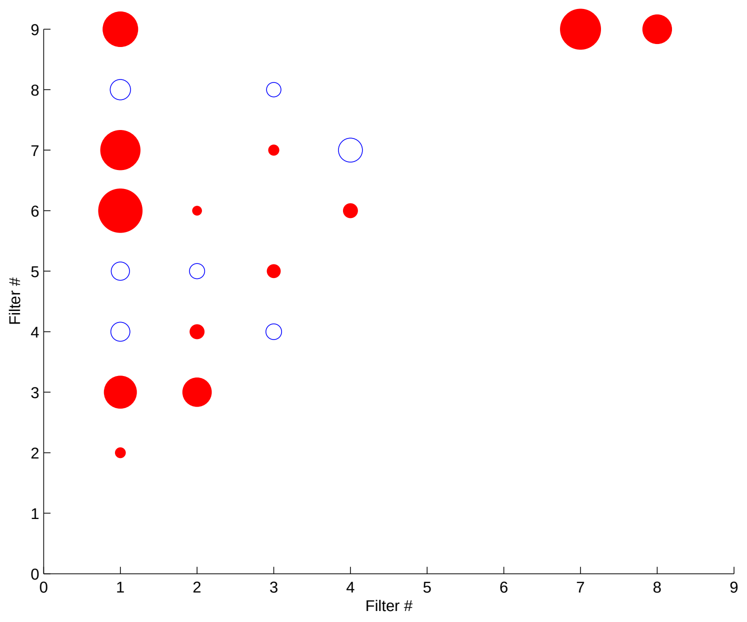

**Figure D.6.** Relative weighting by a network trained on traffic noise data at SNR 0. Note the degeneration; the performance of this network is very poor.

The following figures illustrate more examples of the variation found when pruning several times with different noise types and SNR's.

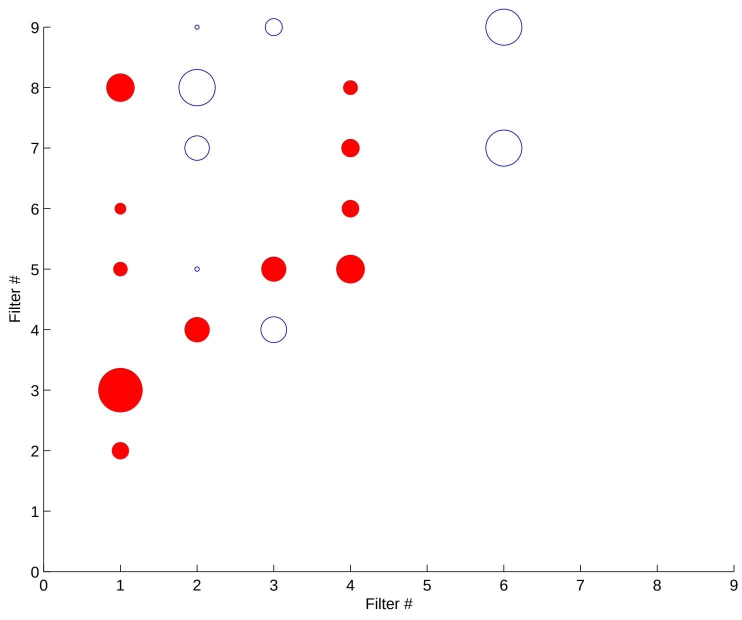

**Figure D.7.** Relative weighting by a network trained on clicks noise data at SNR 0. Again, performance of this network is poor.

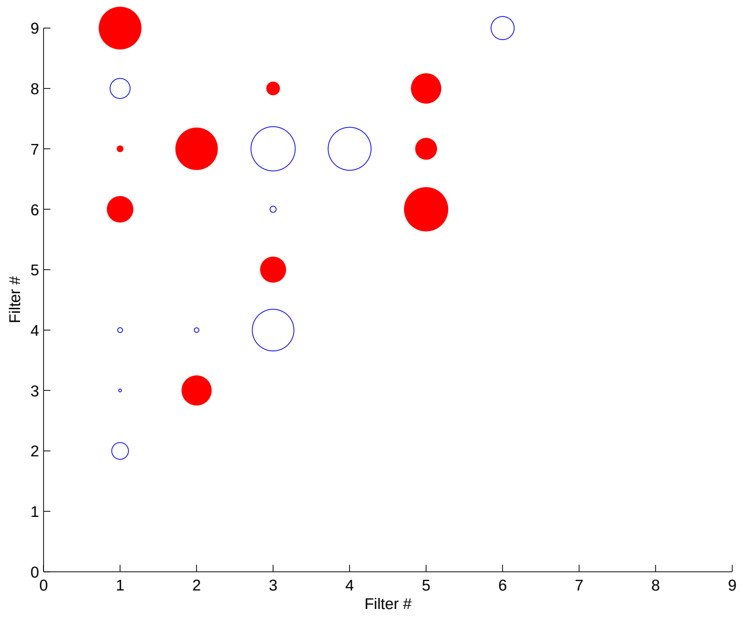

**Figure D.8.** Relative weighting by a network trained on babble noise data at SNR 10. Network has poor performance.

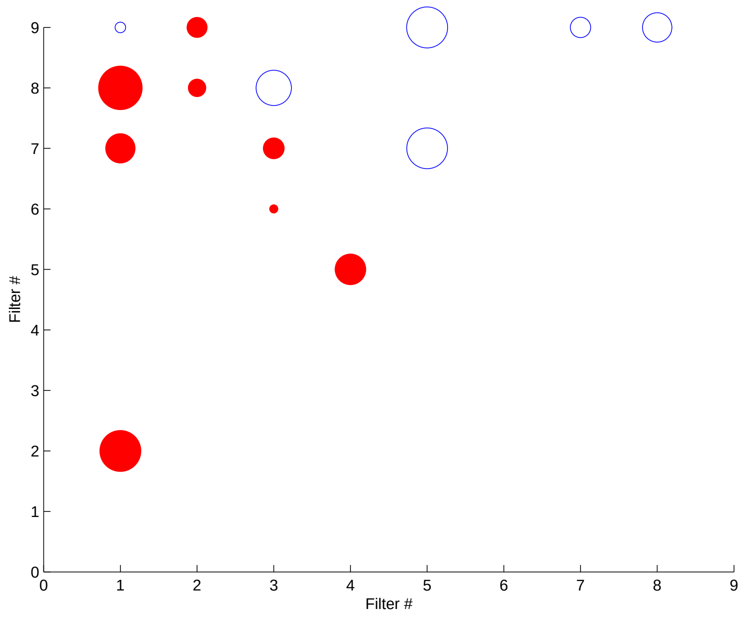

**Figure D.9.** Relative weighting by a network trained on an equal mix of all noise types at SNR 0. Obviously dominated by the presence of babble.

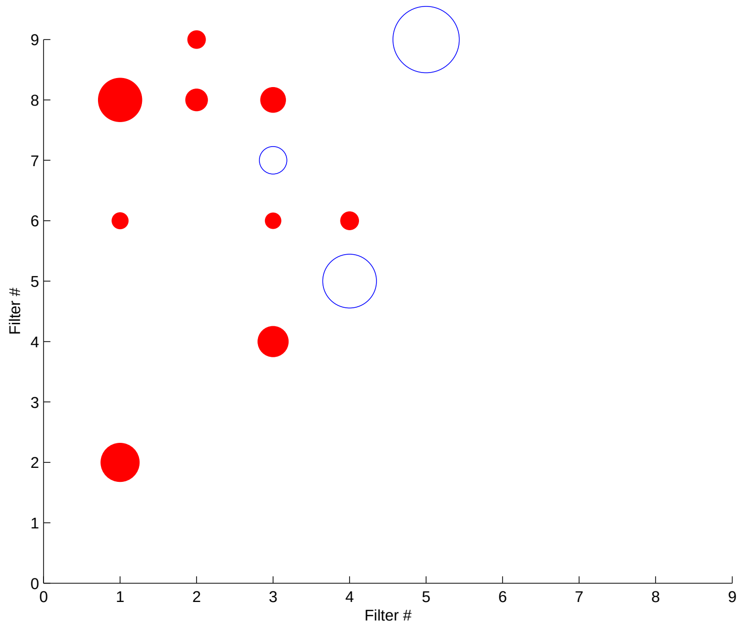

**Figure D.10.** Relative weighting by a network trained on an equal mix of all noise types at SNR 10.

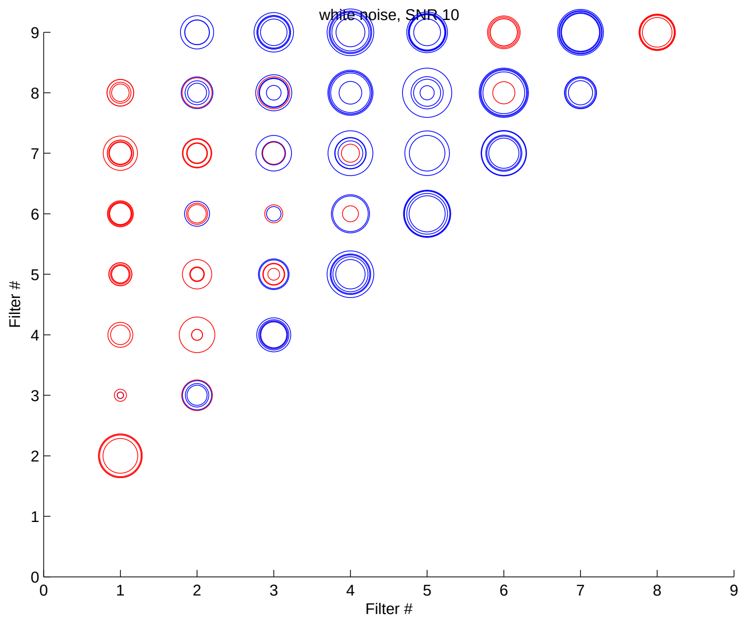

**Figure D.11.** Relative weighting by pruned networks with white noise, SNR 10.

In the following figures, 5 random pruning runs are shown to illustrate some typical results for the different noise types and SNR's. They are all for networks using all 36 cross-correlations as inputs.

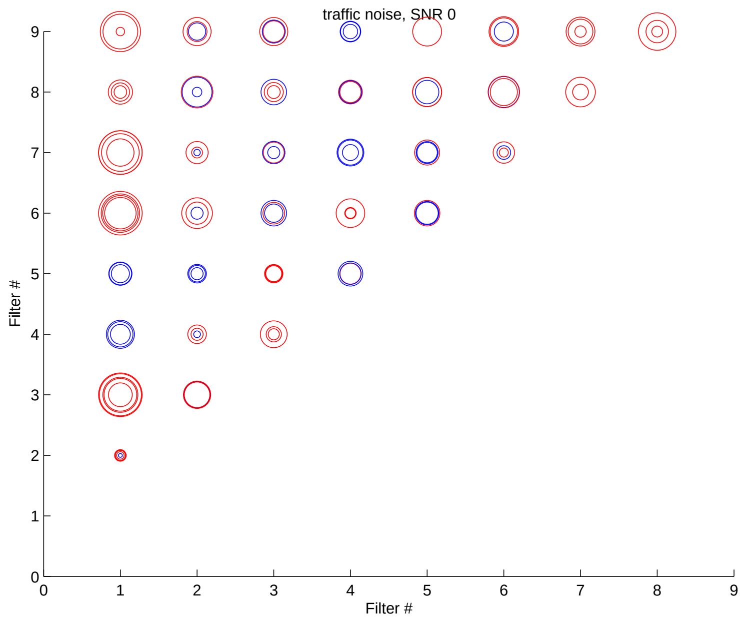

**Figure D.12.** Relative weighting by pruned networks with traffic noise, SNR 0.

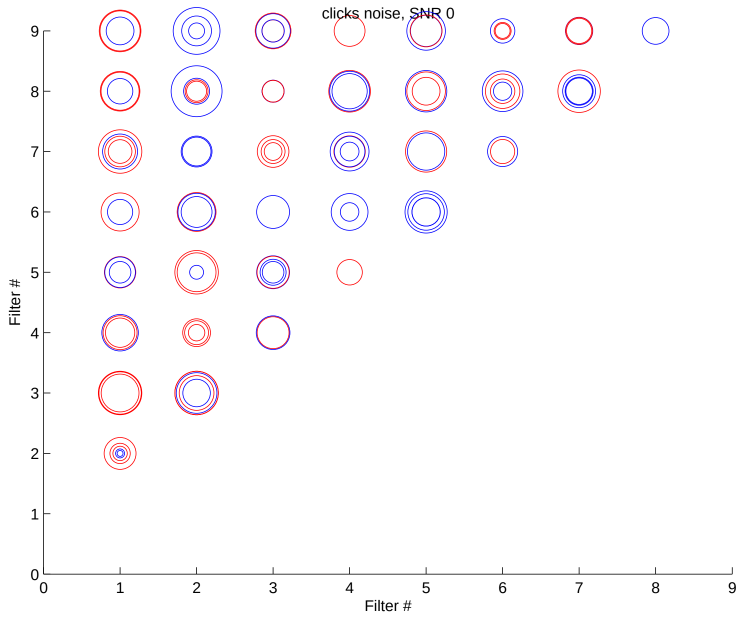

**Figure D.13.** Relative weighting by pruned networks with clicks noise, SNR 0.

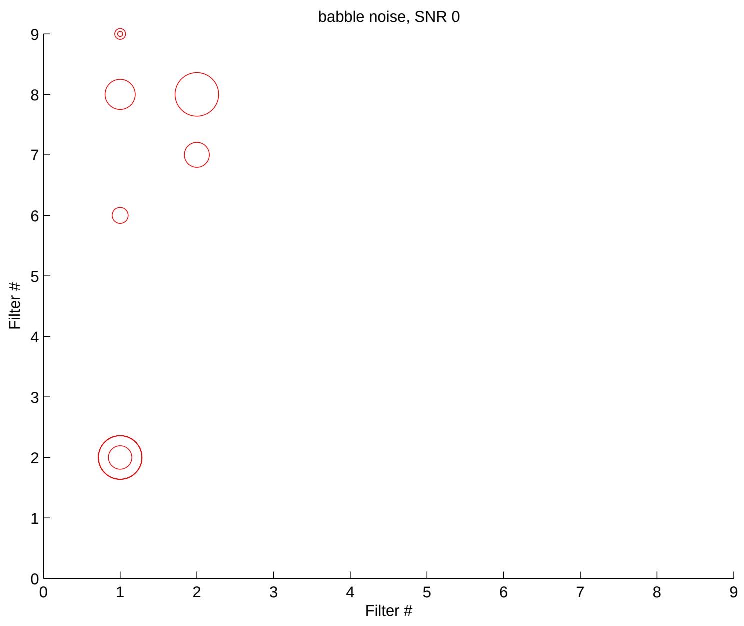

**Figure D.14.** Relative weighting by pruned networks with babble noise, SNR 0. Notice the degeneration; most networks are pruned down to 1 or 2 weights.

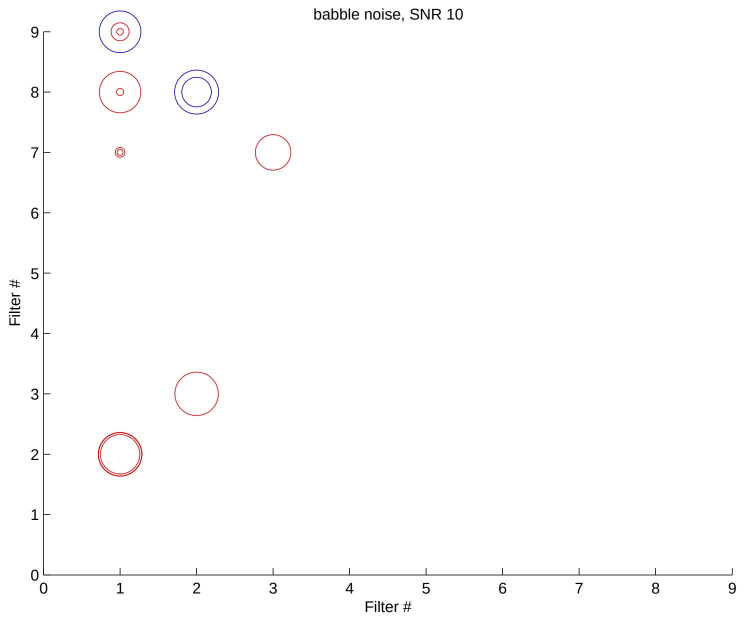

**Figure D.15.** Relative weighting by pruned networks with babble noise, SNR 10.

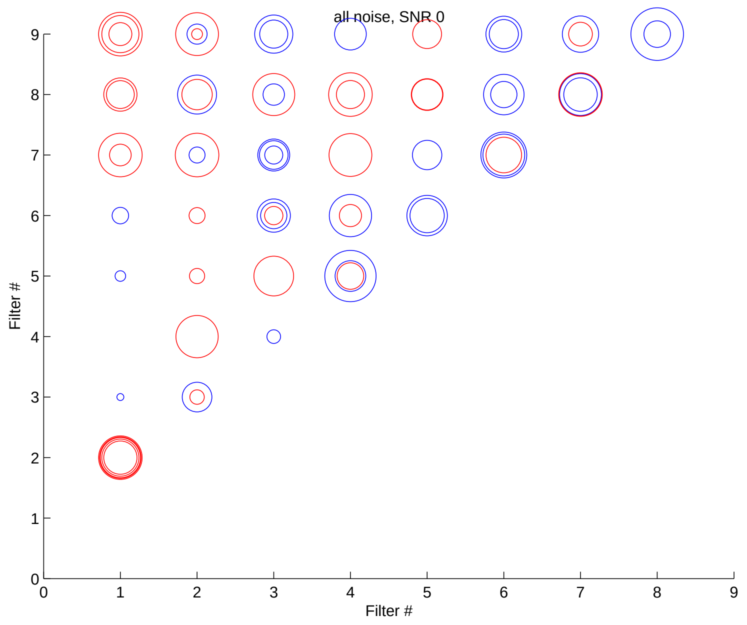

**Figure D.16.** Relative weighting by pruned networks with mixture of all noise types noise, SNR 0.

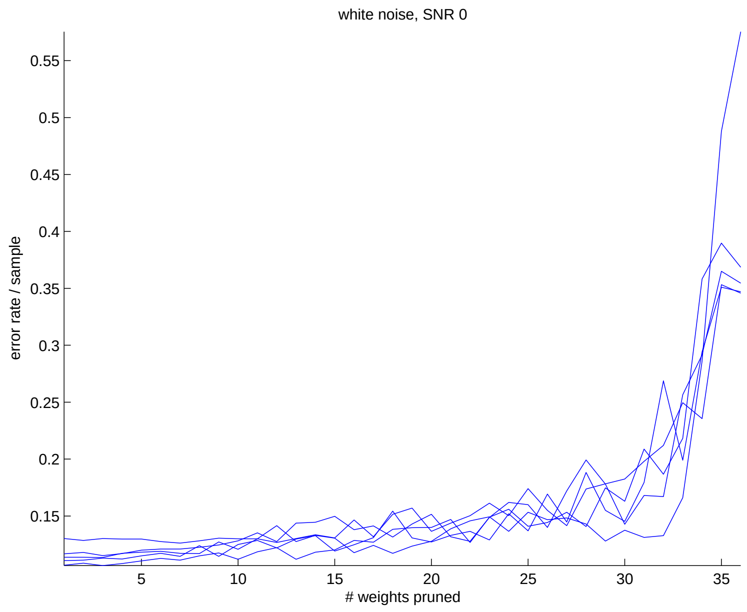

**Figure D.17.** For white noise, most networks perform well.

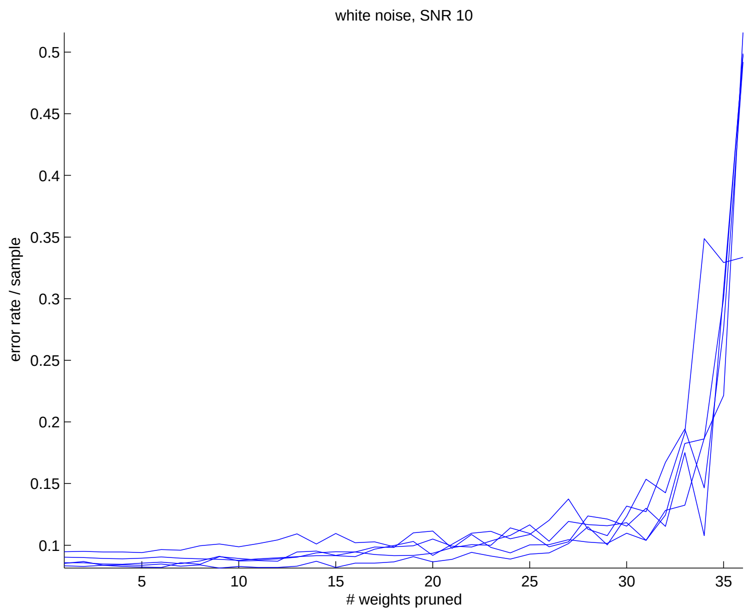

**Figure D.18.** White noise, SNR 10.

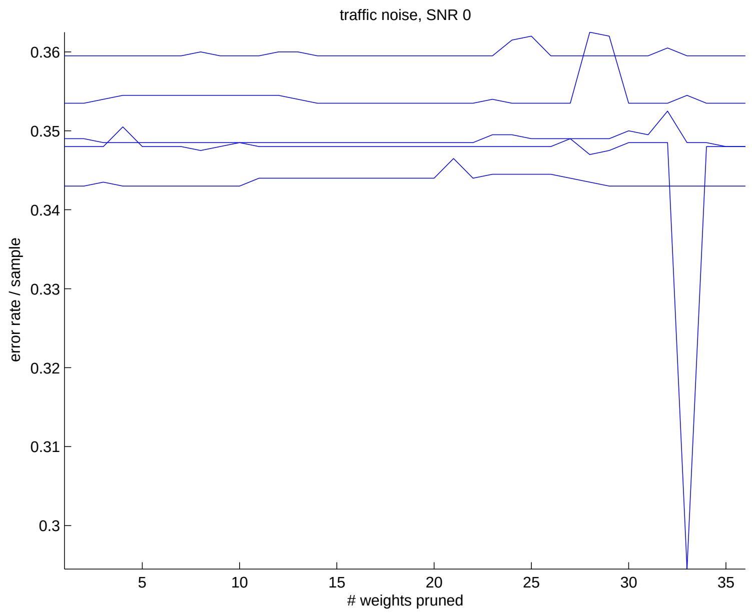

**Figure D.19.** For traffic, SNR 0, most networks perform poorly.

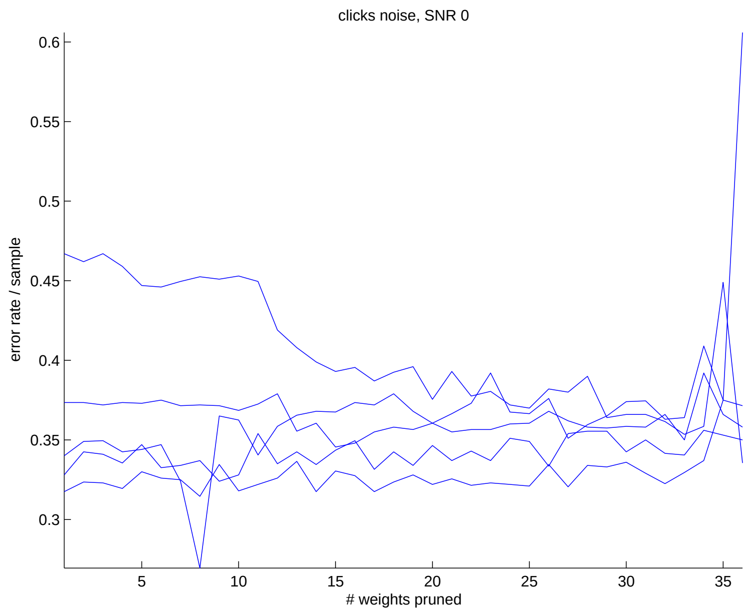

**Figure D.20.** In clicks noise at SNR 0, very few networks are able to learn anything.

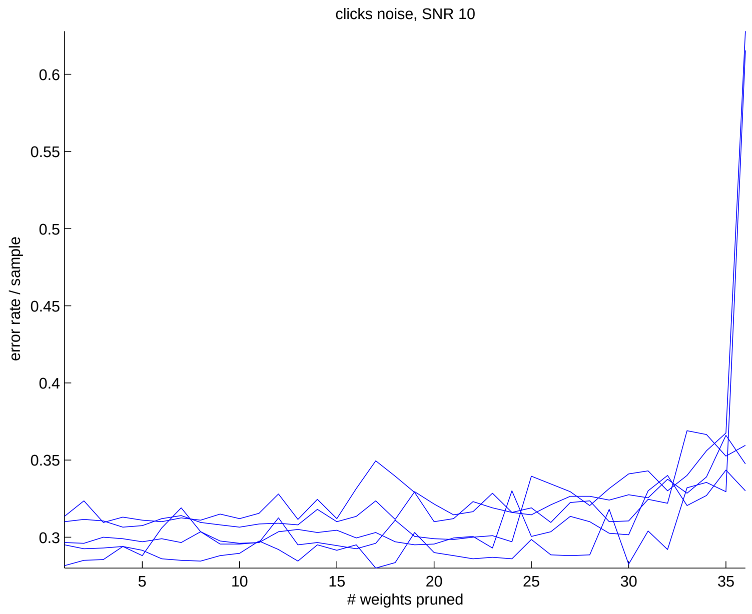

**Figure D.21.** At SNR 10, the results are better and more consistent.

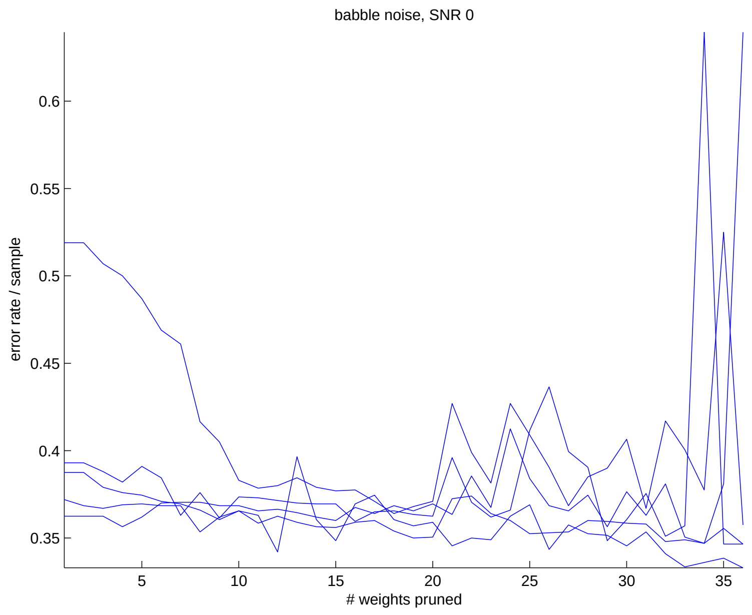

**Figure D.22.** Babble noise makes for impossible learning tasks! (SNR 0).

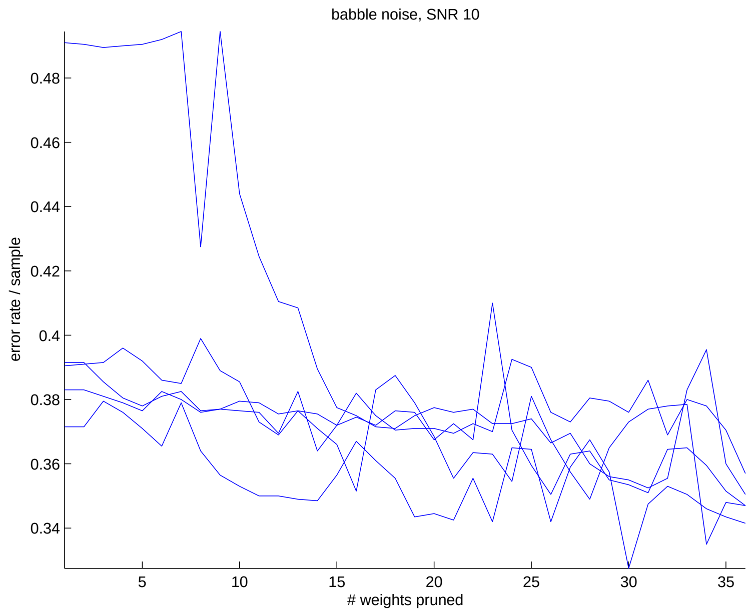

**Figure D.23.** See previous figures comments.

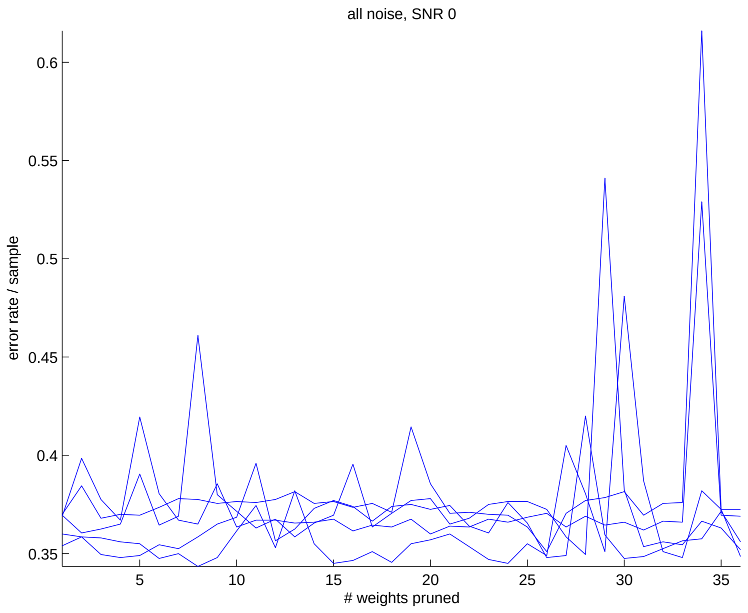

**Figure D.24.** Mixture of all noise types; at SNR 0, the presence of babble noise is probably responsible for the poor performances.

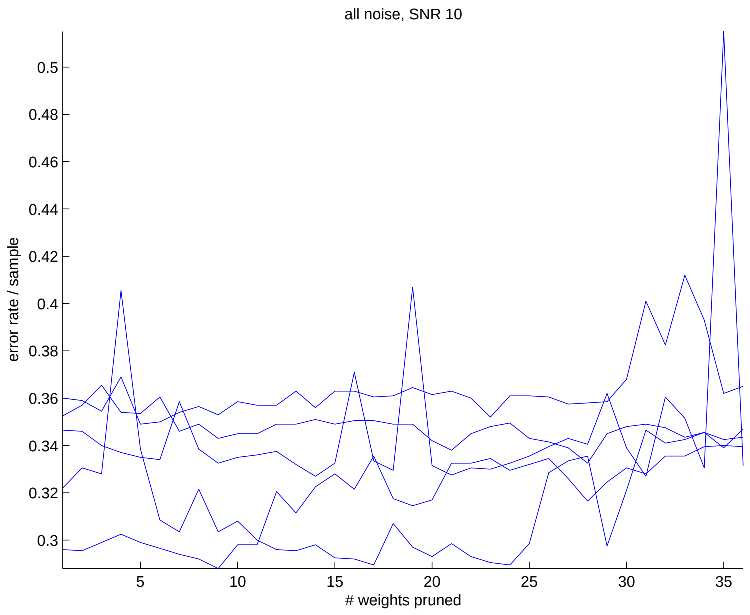

**Figure D.25.** At SNR 10, some networks are able to learn.

The following figures show mean and confidence intervals (assuming normal distributions) for the validation errors, based on 10 pruning runs.

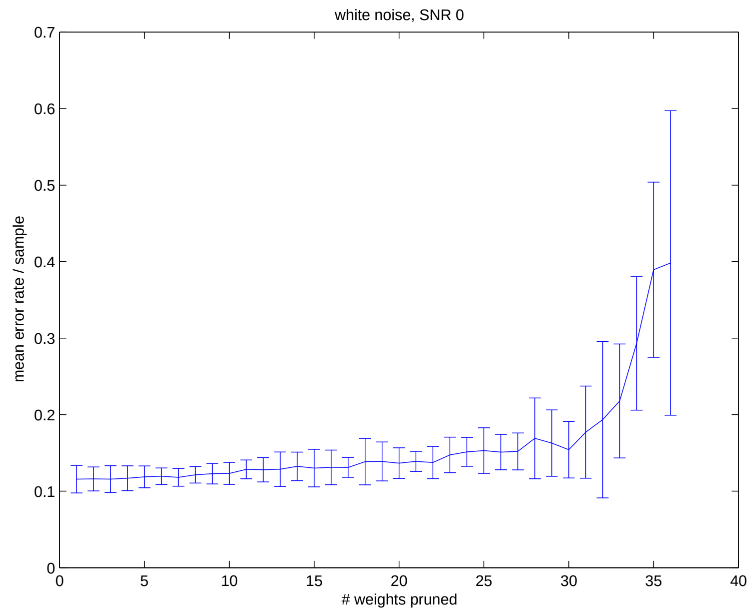

**Figure D.26.** Again, for white noise, performance is consistently good and therefore, variance is low.

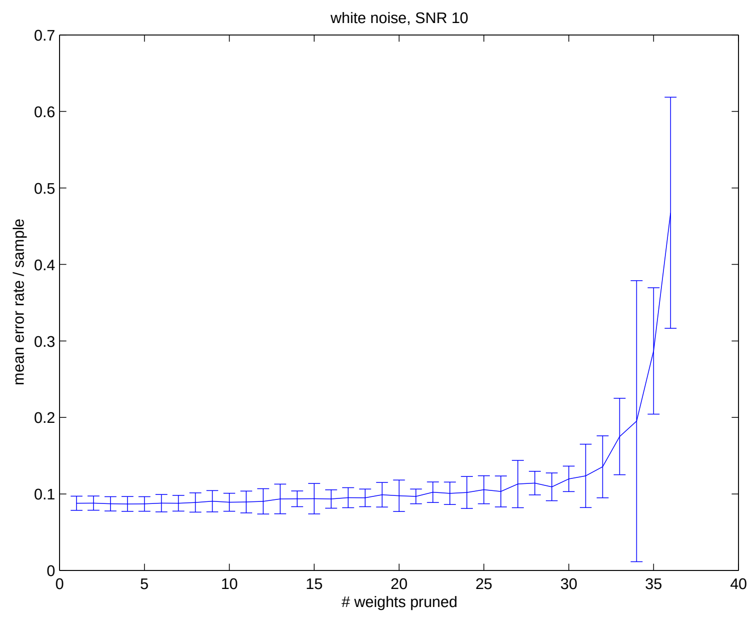

**Figure D.27.**

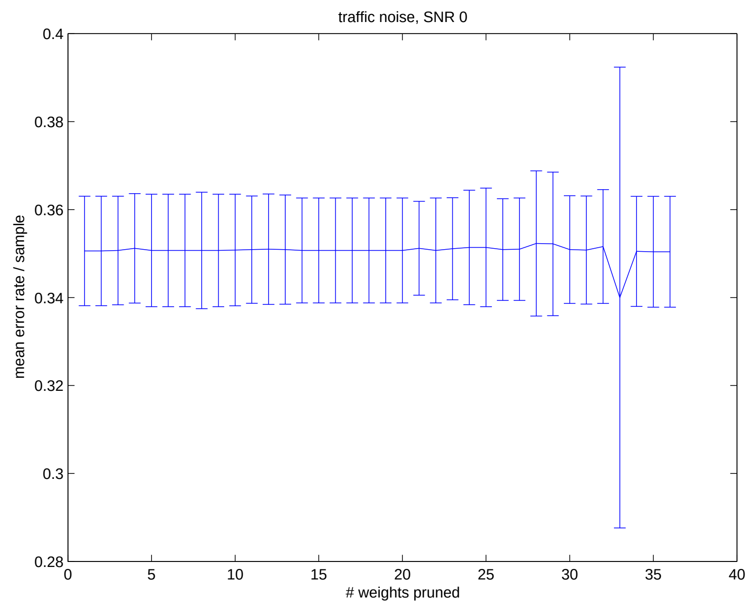

**Figure D.28.**

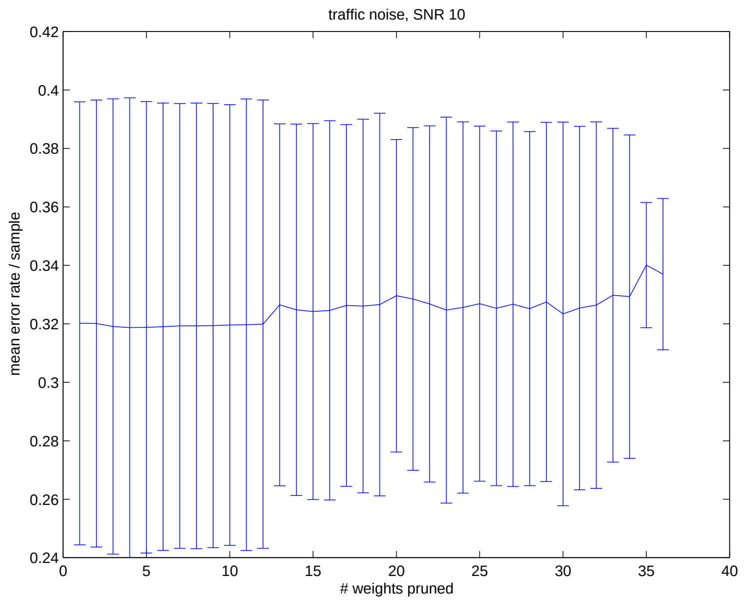

**Figure D.29.**

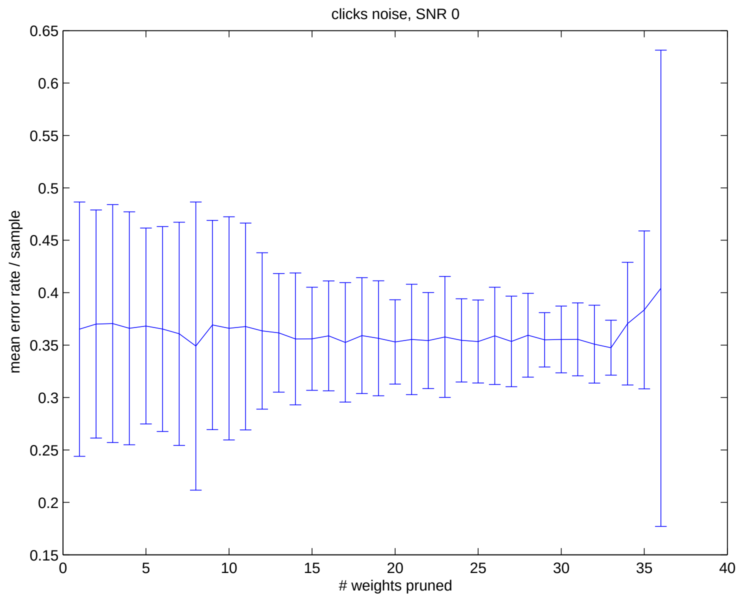

**Figure D.30.**

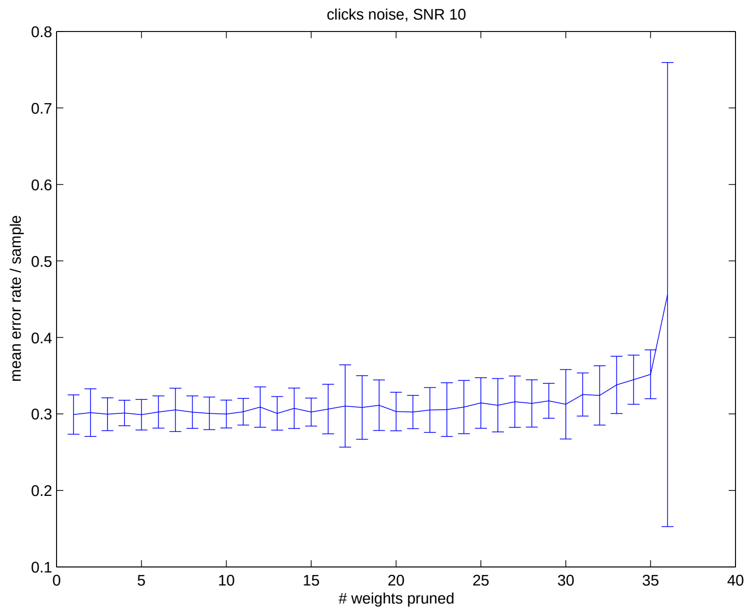

**Figure D.31.** For clicks at SNR 10, results are consistently quite good and the variance is low, meaning that most networks learn to the same degree.

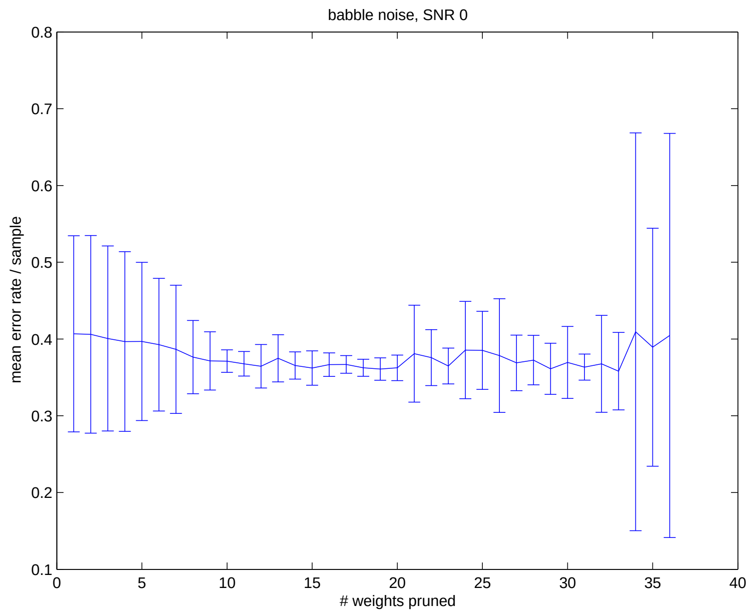

**Figure D.32.**

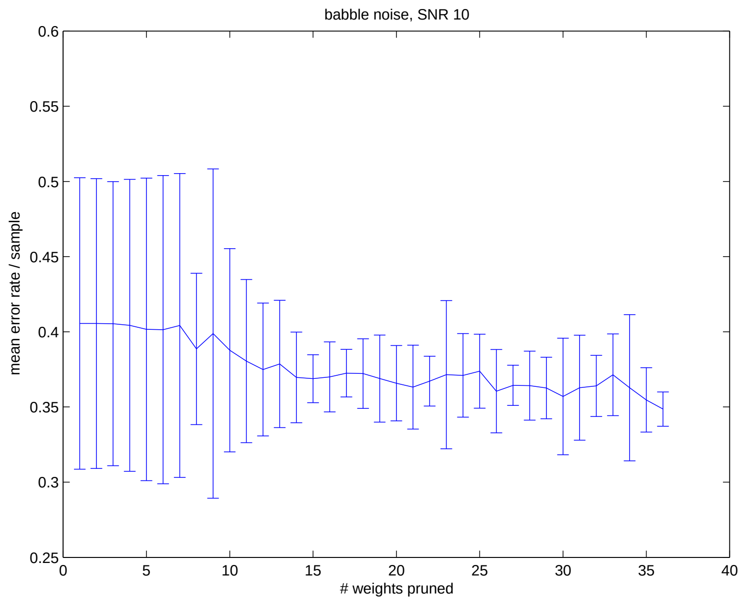

**Figure D.33.**

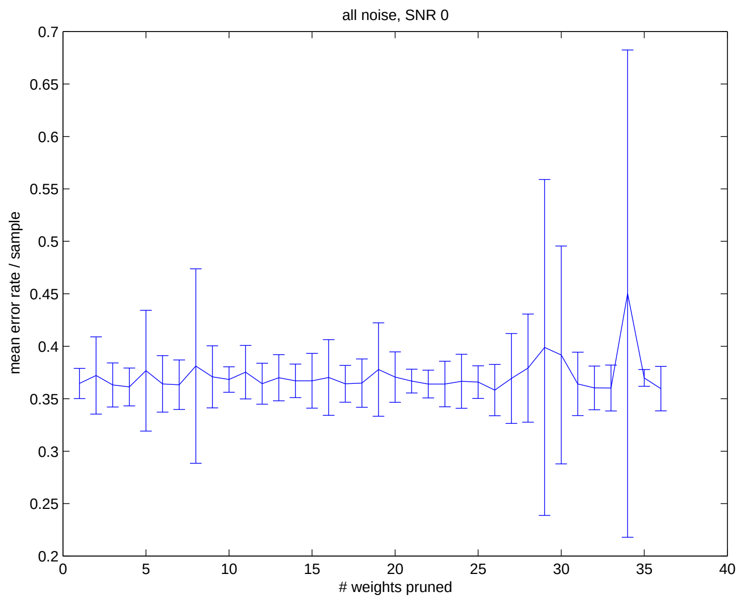

**Figure D.34.** For a mixture of all noise types at SNR 0, the results are probably 'ruined' by the presence of babble.

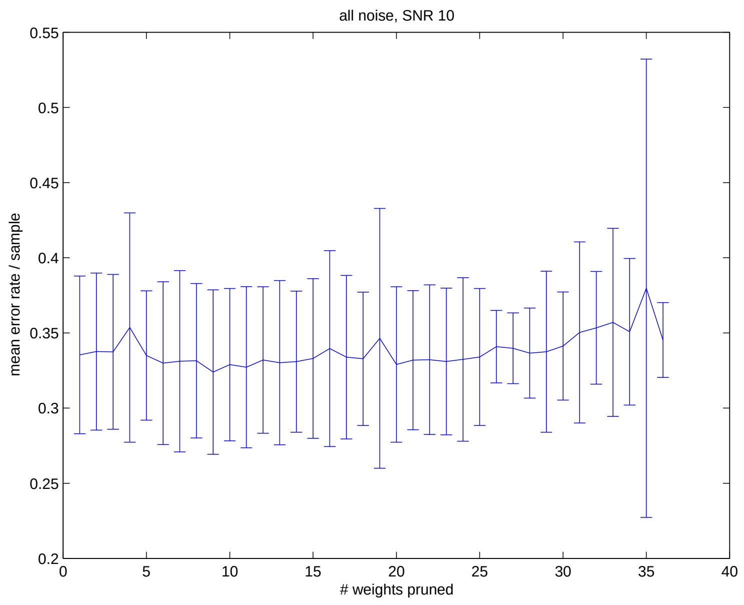

**Figure D.35.** At SNR 10, the variance is probably high due to the random presence of babble, sometimes leading to no learning, while other networks may 'pick up' learning from examples that are not babble.

# Appendix E

# Other features

This appendix gives a short review of some of the features that are often used in audio (and speech) processing.

## E.1 Time-domain features

### E.1.1 Zero-Crossing Rate (ZCR)

This is simply the number of times the time-domain signal crosses zero and is a classic feature for speech/non-speech discrimination. The major problem is that it cannot (alone) discriminate between speech and any other rhythmical source, such as music. The upside is that it is extremely cheap computationally. Used in e.g. [26].

### E.1.2 Energy

This is the average energy of a frame (usually called "Short-time energy" since the frames are short (say 20 ms)). Also used in [26].

## E.2 Spectral-like Features

### E.2.1 Fourier transformation (FT)

The frequency content of part of the signal, e.g. the previous N samples. It can be computed using the Fast Fourier Transform (FFT) so is relatively cheap in complexity ($n \log n$ where $n$ is the number of samples used).

### E.2.2 Spectrogram

This is simply an extension of the FT and contains information on frequency content over time of a part of the signal. Also called "Short-time Fourier transform" (STFT). It is computed by doing a FFT for each window in a series of (possibly overlapping) time-domain windows and thus gives "local" frequency information. The biggest problem is the selection of window length.

## E.3 Spectrally derived features

Many of these are used in [26] and are derived from the power spectrum of the input signal. The simples way to calculate this is using the periodogram:

$$P_x = \frac{1}{N}|X(f)|^2 \tag{E.1}$$

There are many better ways of estimating the power spectrum. This choice is a parameter in itself, and each of the more advanced methods have additional parameters.

### E.3.1 4Hz modulation energy

According to [26], speech has a characteristic energy modulation peak around the 4 Hz syllabic rate (i.e. the rate of phoneme production). This means that the energy in all frequency bands is modulated with time.

This can be measured by taking a series of short-time spectrums of the signal (e.g. columns of a spectrogram) and filtering each of these 'channels' with a bandpass filter centered at 4 Hz. The energy of this filtered signal is then a measure of this feature.

### E.3.2 Percentage of "Low-Energy" frames

This is the proportion of signal frames with power less than 50% of the mean power of the previous several frames. It expresses something about the distribution of power in the signal.

### E.3.3 Spectral roll-off point

This is the 95th (or similar) percentile of the power spectral distribution and is thus a measure of skewness. Voiced speech and music are similar with respect to this feature while unvoiced speech is different. The variance of this feature should then be a good feature.

### E.3.4 Spectral centroid

This is the center of mass for the power spectrum, defined as

$$C = \frac{\sum_i iA_i}{\sum_i A_i} \tag{E.2}$$

It also contains information about the shape of the power spectrum.

### E.3.5 Bandwith

For a spectrum, bandwidth can be calculated as ([20]:

$$BW = \sqrt{\frac{\sum_{n=0}^{N}(n - SC)^2 |X_i(n)|^2}{\sum_{n=0}^{N} |X_i(n)|^2}} \tag{E.3}$$

where SC is the spectral centroid. This says something further about the power spectrum shape.

### E.3.6 Spectral flux

This is the 2-norm of the frame-to-frame spectral amplitude modulation. It is generally higher for music than for speech.

### E.3.7 Rhythmicity

There are different ways of trying to capture rhythmicity in a signal, the idea being to be able to detect the presence of music (which could be made relevant in a VAD context). Ideally, the detection should not react to just any periodic beat (as this would detect for instance clock ticks) but find the sort of patterns associated with rhythmic music.

The 'Pulse metric' is a clever, but heuristic - and quite involved - detection of rhythmicity. It involves peak detection, which is not simple or easy to do robustly. See [26].

### E.3.8 Wavelets

A Wavelet transform (WT) produces a time-frequency 'image' of an audio signal. The difference between the STFT and a WT is that the former has a fixed absolute window length while the latter has a fixed relative (input-signal to transform-signal) window length. Proponents of the WT would argue that it avoids the fixed window length problem of the STFT. There is a 'fast' group of algorithms, called 'Fast Lifting Wavelet Transform' algorithms. It is unknown how fast this is compared to an FFT on the same signal. Probably it depends on the choice of the wavelet function.

### E.3.9 Advanced models of human audition

See for instance [19]. Again, the auditory toolbox contains many advanced models. The driving motivation is often the belief that the human auditory system somehow represents a powerful feature extraction (amongst other things) system. Still, it is questionable if models of it are directly useful for artificial classification systems.

### E.3.10 Cepstra

The cepstral coefficients (CC) are obtained by doing an IFFT (inverse Fourier transform) or a DCT (Discrete Cosine Transform) on the logarithm of the DTFT (Discrete-Time Fourier Transform) of a signal. It de-emphasizes high-frequency components of the DTFT. Usually, not all CC's are used, yielding a special form of compression of the DTFT.

It has the advantage of potentially separating spectrally multiplicative effects and may also be used to compress the Fourier transform.

It can be computed from LPC (Linear Predictive Coding) coefficients (then called 'LPCC'), but this yields slightly different results. LPC-based cepstra may be more robust to noise, but FFT-based cepstra are supposedly better suited for low-noise signals.

They can also be computed from a transformed FFT, e.g. on the mel-scale, see [24]. Using the DCT, this is done as:

$$c_p = \sum_{k=1}^{N} (\log S_k) \cos\Big(\frac{p(k-\frac{1}{2})\pi}{N}\Big) \tag{E.4}$$

where $S_k$ is the output power of each of the N mel-scale bandpass filters; [24] use $N = 20$. They also use only around the first 8 cepstral coefficients. Of course, this can be done using any filterbank.

Cepstral coefficients are highly uncorrelated, so they describe different 2nd-order characteristics of the signal.

Computational cost is of the same order as the FFT (2 FFT's and a logarithm).

#### Mel-frequency cepstral coefficients (MFCC)

These are obtained by doing a DCT (usually) on the log of the power outputs of a range of filterbanks (e.g. gammatone filterbanks). This de-emphasizes high-frequency components of the DTFT due to the frequency location and bandwidth of the filterbanks. Usually, not all CC's are used, yielding a special form of compression of the DTFT.

8 coefficients are used by [24] .

## E.4 Frequency-related features

### E.4.1 SAPVR

This stands for "Spectral Autocorrelation Peak Valley Ratio". It is used to detect structure in the spectral autocorrelation domain. See [22].

Specific to detecting signals with periodic spectrum structure.

### E.4.2 Spectral crest factor

This is the ratio of the larges spectral coefficient to the mean (in a frequency band). Much simpler to calculate than - and would be highly correlated to - the SAPVR.

### E.4.3 Spectral Flatness Measure

See [9]. Discriminates between "tonal" and "noisy" signals. Often done on several frequency bands.

### E.4.4 Spectral peak presence

If there is a (marked) peak in the spectrum of a signal, this indicates a tonal signal. See [9]. Used in the MPEG-1 model.

### E.4.5 Spectral predictability

Using some (simple) model (e.g. LPC) for the spectral coefficients, if prediction error is small, this indicates a tonal quality. Also used by MPEG-1.