الجمهورية الجزائرية الديمقراطية الشعبية
République Algérienne Démocratique et Populaire
وزارة التعليم العالي و البحث العلمي
Ministère de l'Enseignement Supérieur et de la Recherche Scientifique

École Nationale Polytechnique

المدرسة الوطنية المتعددة التقنيات
Ecole Nationale Polytechnique

Electronics Engineering Department

Final Year Project Dissertation
in partial fulfilment of the requirements for:
Electronics Engineer's Degree

Hybrid Deep Learning Based Speech Signal Separation

Authors:

BOUAOUNI Mohamed Yacine **AIT ALI YAHIA Rayane**

Presented and defended in July 11, 2021 before the members of jury:

President	Mourad	Adnane	Prof.	ENP, Algiers
Supervisor	Adel	Belouchrani	Prof.	ENP, Algiers
Examiner	Sid-Ahmed	Berrani	PhD.	ENP, Algiers

ENP 2021

École Nationale Polytechnique (ENP)
10, Avenue des Frères Oudek, Hassen Badi, BP. 182, 16200 El Harrach, Alger, Algérie
www.enp.edu.dz

الجمهورية الجزائرية الديمقراطية الشعبية
République Algérienne Démocratique et Populaire
وزارة التعليم العالي و البحث العلمي
Ministère de l'Enseignement Supérieur et de la Recherche Scientifique

École Nationale Polytechnique

المدرسة الوطنية المتعددة التقنيات
Ecole Nationale Polytechnique

Electronics Engineering Department

Final Year Project Dissertation
in partial fulfilment of the requirements for:
Electronics Engineer's Degree

Hybrid Deep Learning Based Speech Signal Separation

Authors:

BOUAOUNI Mohamed Yacine **AIT ALI YAHIA Rayane**

Presented and defended in July 11, 2021 before the members of jury:

President	Mourad	Adnane	Prof.	ENP, Algiers
Supervisor	Adel	Belouchrani	Prof.	ENP, Algiers
Examiner	Sid-Ahmed	Berrani	PhD.	ENP, Algiers

ENP 2021

École Nationale Polytechnique (ENP)
10, Avenue des Frères Oudek, Hassen Badi, BP. 182, 16200 El Harrach, Alger, Algérie
www.enp.edu.dz

الجمهورية الجزائرية الديمقراطية الشعبية
République Algérienne Démocratique et Populaire
وزارة التعليم العالي و البحث العلمي
Ministère de l'Enseignement Supérieur et de la Recherche Scientifique

École Nationale Polytechnique

المدرسة الوطنية المتعددة التقنيات
Ecole Nationale Polytechnique

Département d'Electronique

Mémoire de projet de fin d'études
pour l'obtention du diplôme:
Ingénieur d'état en Electronique

Apprentissage Profond Hybride Appliqué à la Séparation de Signaux Audio

Auteurs:

BOUAOUNI Mohamed Yacine **AIT ALI YAHIA Rayane**

Présenté et soutenu publiquement le 11/07/2021 auprès des membres du jury:

Président	Mourad	Adnane	Prof.	ENP, Alger
Promoteur	Adel	Belouchrani	Prof.	ENP, Alger
Examinateur	Sid-Ahmed	Berrani	PhD.	ENP, Alger

ENP 2021

ملخص يعد فصل مصدر الصوت مشكلة صعبة تتكون من تحديد المصادر المختلفة الموجودة في إشارة مختلطة ، إما باستخدام الأساليب التقليدية القائمة على النماذج أو باستخدام خوارزميات التعلم العميق. في هذا العمل ، نقترح نموذجين مختلفين للجمع بين الأساليب القائمة على النموذج (عامل المصفوفة غير السلبي) مع الشبكات العصبية للاستفادة من كليهما. يدمج الأسلوب الأول بين *NMF* والشبكة العصبية العميقة *DNN* في مكدس مرحلتين متتاليتين ، حيث يعزز *DNN* فصل الإشارات عن طريق تحديث مخططات الطيف / المكاسب التي تم تقديرها باستخدام *NMF* . يتم تقديم معمارتين تستندان إلى أجهزة التشفير التلقائية في هذه الأطروحة ، والتي تتعامل مع نوعين مختلفين من بيانات الإدخال. يعتمد النهج الثاني على نموذج يتكشف بعمق. وهو يتألف من فتح خوارزمية التحسين للطريقة القائمة على النموذج في طبقات من شبكة عميقة ، وتدريبها باستخدام تقنيات التعلم العميق.

كلمات مفتاحية ـ التشفن التلقائي، تعليم عميق، فتح الخوارزميات، *NMF* ، *DNN*

Résumé— La séparation de source audio est un problème complexe qui consiste à identifier les différentes sources présentes dans un signal mixte, soit en utilisant des méthodes de modèle traditionnelles, soit à l'aide d'algorithmes d'apprentissage profond. Dans ce travail, nous proposons deux paradigmes différents pour la combinaison de méthodes basées sur des modèles (factorisation de matrice non négative) avec des réseaux de neurones afin de tirer profit des deux. La première approche consiste à fusionnner la NMF et le réseau de neurones profond, dans une pile constituée de deux étapes séquentielles, où le DNN améliore la séparation des signaux en mettant à jour les spectrogrammes / gains estimés à l'aide de la NMF. Deux architectures basées sur des autoencodeurs sont présentées dans cette thèse, qui acceptent deux types de données d'entrée différents. La deuxième approche est basée sur le paradigme de déploiement profond, qui consiste à déployer l'algorithme d'optimisation de la NMF, sur les couches d'un réseau de neurone profond et à l'entraîner en utilisant des techniques de Deep Learning.

Mots-clés : Apprentissage profond, *NMF*, *DNN*, Autoencodeurs, Algorithme de Déploiement Profond.

Abstract— Audio source separation is a challenging problem which consists of identifying the different sources present in a mixed signal, either by using traditional model based methods or using deep learning algorithms. In this work, we propose two different paradigms for combining model based methods (non-negative matrix factorization) with neural networks to take advantage of both. The first approach fuses the NMF and a deep neural network (DNN) in a two sequential stages stack, where the DNN enhances the separation of the signals by updating the spectrograms/gains that were estimated using the NMF. Two architectures based on autoencoders are presented in this thesis, that handle two different kind of input data. The second approach is based on the deep unfolding paradigm. It consists of unrolling the optimization algorithm of the model based method into layers of a deep network, and train it using deep learning techniques.

Index-terms : Deep Learning, NMF, DNN, Autoencoders, Unfolding algorithm.

Acknowledgement

First and foremost, all our gratitude, and thanks go to Allah the whole powerful who gave us strength, patience, courage, and willingness to develop this work.

We thank our promoter Adel Belouchrani, Professor at the National Polytechnic School, world leader in signal processing, for having accepted to supervise us, and to have been present at all times for the performance of this work. We also thank him for his continued encouragement and motivating, for his moral support and his relevant remarks. We wish to express our sincere gratitude to him for his availability and his advice which were invaluable to us in order to carry out this work successfully.

We would also like to thank Mr. Mourad Adnane, Professor at the National Polytechnic School, having accepted to chair our jury, and Mr. Sid-Ahmed Berrani our examiner, for his interest in our work.

During these 3 years of dedication, Mr. Mourad Adnane and Mr. Sid-Ahmed Berrani have helped us enormously, by their knowledge, their teachings, their availability, and their way of being. They have always been attentive and have never hesitated to share their knowledge with disconcerting ease which demonstrates their excellence.

Dedication

We would like to dedicate our work to our parents, brothers and sisters, who always helped us throughout the entire educational pathway. They always stood alongside us during tough moments, offering us their wholehearted support and elating words of encouragement.

We offer this dissertation to all our professors at the Electronics Department at Ecole Nationale Polytechnique in particular Mr. Adel Belouchrani, Mr. Mourad Adnane, and Mr. Sid Ahmed Berrani, who have been always supportive and available.

We also dedicate this project to our beloved ones Lynda Zemirline and Meriem Zanoun, Yana Fernani, Malik Ait Ali Yahia, Omar Medjdoub, Younes Moussaoui, Youcef Aissaoui, Rabah Saadoun, Mechehed Mokrane, Djelloul Bouida, Houssem Otmani, Khennas Mohamed Ibrahim, Soulaimane Saadi. All those who have supported me throughout the process. We will always appreciate everything they have done.

Contents

List of Figures

Acronyms

AdaGrad Adaptive Gradient Algorithm.

Adam adaptive moment estimation.

AE Autoencoder.

AI Artificial Intelligence.

ANN Artificial Neural Network.

DFT Discrete Fourier Transform.

DL Deep Learning.

DNNs Deep Neural Networks.

FT Fourier Transform.

GMM Gaussian Mixture Model.

IS Itakura-Saito.

ISTA Iterative Shrinkage and Thresholding Algorithm.

ISTFT Inverse Short-Time Fourier Transform.

KL Kullback-Leibler.

LBFGS Limited-Memory Broyden–Fletcher–Goldfarb–Shanno.

LSTM Long Short Term Memory.

MAE Mean Absolute Error.

ML Machine Learning.

MLP Multi-Layer Perceptron.

MSE Mean Square Error.

MU Multiplicative Update.

NMF Non-Negative Matrix Factorization.

NN Neural Network.

NNDSVD Non Negative Double Singular Value Decomposition.

PMF Probability Mass Function.

RBM Restricted Boltzmann Machine.

RELU Rectified Linear Unit.

RMSprop Root Mean Square Propagation.

RNN Recurrent Neural Network.

SDR Signal-to-Distortion Ratio.

SE Square Error.

SGD Stochastic Gradient Descent.

SIR Signal-to-Interference Ratio.

SMR Signal-to-Music Ratio.

SNMF Sparse Non-Negative Matrix Factorization.

SNR Signal-to-Noise Ratio.

STFT Short-Time Fourier Transform.

SVD Singular Value Decomposition.

TF Time frequency.

WSISTA Warm Start Iterative Shrinkage and Thresholding Algorithm.

Introduction

Source Separation Source separation is a subject of utmost importance because of the challenges it presents, and the need for such analysis in today's world. Simply said, source separation aims to analyze a set of mixed signals, in order to recover each source signal from this mixture with little or no information about the mixing process or the sources.Among the areas in which source separation is applied, we will cite audio separation, with its applications to speech, music, and environmental audio. Audio source separation has been a resounding success since the technological development, the emergence of devices equipped with one or more microphones, such as smartphones, and hearing aids that require developing efficient audio processing algorithms to offer a unique user experience.

Audio Pollution In addition, noise pollution encourages scientists to work on solutions based on source separation. The term "noise pollution" refers to the effects caused by acoustic phenomena (noises) that have consequences on people's health, from temporary discomfort to more serious disorders. According to a WHO study [1], noise is an underestimated threat that can cause health problems both in the short and the long term, such as cardiovascular diseases, a drop in productivity, and hearing loss. In addition, 80% of workers who work in a noisy environment reported being nervously tired, which impacts business productivity and therefore potential economic loss. Moreover, most hearing aids available at affordable prices suffer from the problem of amplifying both the speech and the background noise, which makes the sounds distorted and creates a whistling or buzzing. Some solutions have been proposed using a transmitter-receiver device, but they limit the communication to only one single person at a time.

Deep learning Deep Learning (DL) is a branch of Artificial Intelligence (AI) that was inspired by the biological brain. It allows tackling tasks that are too difficult to solve with fixed programs designed by human beings. It has been widely recognized as the representative advances of machine learning or artificial intelligence in general nowadays [2]. This can be attributed to the recent development made by deep learning in a series of challenging applications. For instance, in computer vision, DL algorithms improve the accuracy of face recognition to be higher than 99%, beating the human level [3]. In speech recognition and machine translation, deep learning

is approaching the performance level of an interpreter [4]. In audio separation and speech enhancement, deep learning almost manages to mimic the behavior of the auditory cortex to ignore the surrounding noise. In the medical diagnosis, it has matched the level of senior professional physicians [5]. Until now, it has been hard to find areas in which DL was not successful.
From a scientific point of view, deep learning is interesting because it helps to understand the principles that underlie intelligence, such as the brain's primary visual cortex that was an inspiration for Convolutional Neural Network (CNN) .
Deep learning models are often called data-driven because they heavily rely on huge amounts of data to learn the rules that govern the inputs. These models aim to solve non-linear input/output mapping by providing a powerful framework for supervised, unsupervised, and semi-supervised learning. This efficiency can be attributed to the ability of DL algorithms to automatically extract salient features that represent the data and model functions of increasing complexity. In addition to its simple use and open source community, DL models provide unbeatable results with computationally efficient operations that make them convenient to work with. However, the dimensionality and their millions of parameters make it impossible to interpret the behavior of these models. That's why deep learning is considered a black box.

Our work In our thesis, we focused our solution on model-based and data-driven methods. We propose two different paradigms for combining the best of both methods while avoiding their drawbacks, to solve the audio source separation problem, more precisely on separating speech from music audio under different constraints.

In fact, real-world speech signals are often polluted by interfering speakers, environmental noise, music, or reverberation. These disturbances deteriorate speech quality and, in adverse scenarios, speech coherence, and clarity, but also automatic speech recognition performance. Source separation is therefore required in such scenarios. Speaking about our developed approaches in the subject, they demonstrated prompting results over the state of the art audio source separation techniques. Furthermore, our methods may be as well performed on images and tensors which may involve no time dimension. For example, in medical imaging that involves careful measurements, artifacts can significantly degrade the accuracy of the measurement, our algorithms can help remove undesired artifacts from the desired signal. Moreover, we can cite seismic monitoring that uses sensitive seismographs to determine the location, as well as other characteristics of earthquakes, which can also be improved using our approaches. Besides, radar, which is a multi-sensor system, can also be another application of our approaches for meteorology, air traffic control, surveillance, and astronautics.

The richness of our thesis is characterized by the different perspectives from which we ap-

proached the problem. Our solutions consist of two different methods that aim to solve the same problem of mixture separation. The main contributions of our work can be summarized as follows:

- Propose a Deep Neural Network architecture that enhances the magnitude spectrum (time-frequency domain) of each respective source estimated by the Sparse Non-Negative Matrix Factorization (SNMF). This approach is characterized by a pretrained stage using an UnderComplete Autoencoder which was used to initialize the DNN.
- Propose an architecture based on a Denoising Autoencoder (Denoising-SNMF) that aims to improve the gains/activations matrix (temporal domain) estimated by the Sparse Non-negative Matrix Factorization (SNMF). Time series are known for being noise prone, the denoising process of the Denoising Autoencoder helps undo this corruption, which improves the mixture separation.
- Propose an improvement to the Warm start Iterative Shrinkage and Thresholding Algorithm (WSISTA) used to solve the Non-negative Matrix Factorization (NMF) problem by introducing a memory term to provide more smooth changes in the estimations.
- Improve the Deep Recurrent NMF that was proposed in [6]. This method is based on the deep unfolding paradigm that encourages both the interpretability of the deep network and its performances.

Organization This thesis is organized as follows:

- **Chapter I** provides a background on the context of speech source separation, time-frequency processing, Deep Learning, and Deep Unfolding. It gives an overview of the basic use of time-frequency tools, as well as the mechanism of Deep learning and Unrolling Algorithms.
- **Chapter II** describes the dataset used, the preprocessing techniques, the evaluation criteria, and the software tools (libraries and environment).
- **Chapter III** presents a model-based method (NMF) widely used for single channel source separation, over which we build our solutions. This chapter gives detailed explanations about the mechanism that governs low-rank matrix decomposition, especially NMF, as well as its variants, its training and validation process. Finally, it shows the results of such techniques, and the impact of some factors on the separation process.
- **Chapter IV** presents the proposed methods based on Deep Learning, by explaining technical details related to both architectures, and the contributions we made over the paper we inspired the work from. Finally, we show the results and their interpretations.

- **Chapter V** is without any doubt, the most exciting chapter we have worked on. It introduces unfolding techniques that combine both advantages of the model-based and data-driven methods. This chapter demonstrates how we unrolled the Iterative Shrinkage and Thresholding Algorithm (ISTA) [7] into a network and trained it with back-propagation. We then present the many contributions and changes we made over the original paper [6] to adapt it to our problem. Finally, we conclude this study with the results obtained by this technique.

Chapter 1

Background

1.1 Source separation

Source separation attracted the attention of scientists and researchers by its complexity and the need of solving such a problem in a world where the signal is subject to different types of noise and distortions in multiple fields such as the medical field, radars, and seismology where accuracy and precision are required.

The source separation aims to analyze the mixture signal, in order to separate each source component from other components. This problem can be formulated as follows:

$$s(t) = \sum_{i=1}^{M} s_i(t) + n(t) \tag{1.1}$$

$s_i(t)$: i^{th} source signal of the overall mixture.
M : Total number of sources.
$n(t)$: Additive noise.
The approach to perform source separation follows a general processing scheme, described in subsection 1.1.1.

1.1.1 General Process of Source Separation

A vast majority of approaches that are interested in the separation of mixture signals follow the general process below:

- Convert the mixture signal $s(t)$ from the time domain to the time-frequency domain.
- Estimate the parameter relative to time-frequency characteristics of each source defined by their spectra, according to some criteria and constraint, for example, when using Non-Negative Matrix Factorization NMF, we assume the resulting matrix to be non-negative.
- Application of spatial/spectral filtering in order to extract the complex-valued time-frequency coefficients of the sources.
- Applying an inverse operation to switch from time-frequency domain to the temporal domain yielding source estimates $\tilde{s}_i(t)$

1.1.2 Categorization of source separation

Because source separation can be handled differently depending on the information available about the mixture signal as well as the clean signal of every source, source separation can be defined by category:

- **Blind**: When no prior information and knowledge are available about the mixture, and/or the source signals.
- **Semi-blind**: When general information is available about the mixture signal, and the mixture process.

1.1.3 Source Separation Approaches

Tackling source separation problem depends on the prior information we have about the mixture signal as described in 1.1.2, that's why different approaches exist, which depend on the category of the problem:

- **Learning-Free**: This kind of method does not rely on any kind of prior knowledge about the mixture signal; in other words: when no training data is available. All parameters are directly estimated from the signal mixture. For the algorithms that are Learning Free we can cite Independant Component Analysis (ICA) or Learning-Free NMF.
- **Supervised methods**: Consist of training a model for each source from clean signals, in order to separate the test mixture using the prior knowledge of each isolated source, such as Trained-NMF.
- **Separation based Training**: This method based on Neural Networks, reformulates the source separation problem into minimizing an objective function between the input and the desired target.

1.1.4 Single Channel vs Multi-Channel

Assume that the observed signal is recorded using one or more microphones. By single channel, we mean the output of one microphone, and it represented by a scalar $s(t)$, while multi-channel concerns the output of multiple microphones, represented by a $C \times 1$ vector $s(t)$ where C is the number of microphones.

1.1.5 Audio Source Separation

The audio source separation stems from the "Cocktail party problem" [8], where multiple people talk at the same time. In fact, sounds from different sources in the party mix in the air before arriving at the ear of the observant, constraining the auditory cortex [9] to focus on a single speaker while ignoring the rest.
An ideal solution would be to imitate the auditory cortex in order to take the sound consisting of the mixture of different sources, and somehow separate out the individual source signals. This is known as the audio source separation problem.

Source Separation vs Enhancement

Cocktail party problem implies multiples speakers, causing interference, distortions, echo and reverberations.
The source separation aims to extract one or more target source while trying to cancel signal contamination such as interference and distortion. On the other hand, enhancement is mostly used in the case where the target we want to isolate is speech, it also refers to the problem of extracting one or more sources while canceling all types of distortion including reverberations and echo that are not explicitly canceled by source separation.

Related Works

In term of the single-channel audio separation, many techniques were used to solve this problem. For approaches that use training data, authors of [10], [11] used probabilistic models such as the Gaussian Mixture Model (GMM), hidden Markov Model (HMM). Authors of [12]–[14] used low level matrix approximation approach based on Non-Negative Matrix Factorization (NMF), using non-negative training dictionaries, with different penalties and loss functions. A variant of NMF was proposed by [15] that learned temporal dependencies using convolution. Other models based on deep learning were proposed by [16] which approach the problem using a DNN to enhance the estimated spectrum of NMF, Recurrent Neural Networks (RNN) were also used as well as Long Short Term Memory (LSTM) [17], [18] for single-channel speech separation and speech enhancement respectively. Modern technique in the intersection between model-based and data-driven methods was also used, where authors of [6] proposed an unrolling algorithm based on NMF combined with Deep Learning.

1.2 Time Frequency Processing

Audio signal processing has three main representations of the signals: time domain, frequency domain, and time-frequency domain. Most of the audio processing algorithms are applied to the time-frequency representation of the signals and not the raw audio data (temporal representation).

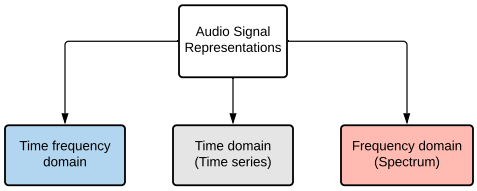


Figure 1.1: Audio signal representations.

- **Time domain** : Representation of the amplitude of the sound at each timestamp. It's the raw data that is collected with one or multiple microphones.
- **Frequency domain** : Representation of the signal as a function of frequency. The "spectrum" of frequency components is the frequency-domain representation of the signal. The conversion from the time domain to the frequency domain is done using transforms (e.g. Fourier transform). This representation does not capture the variations over time.
- **Time-frequency domain**: Rather than encoding the signal in temporal domain or in the frequency domain, this two-dimensional representation considers both the time and frequency characteristics of the signal.

Why we need a TF representation?

Time-frequency representations have been widely used to solve separation and enhancement problems.

The interest in this representation is due to the fact that most signals hold important features in both the time and frequency domain. Also, Fourier analysis assumes that signals are infinite in time or periodic. However, most signals change over time. Thus, the need for representing the variations of the frequency characteristics over time (non-stationary signals). Moreover, the natural sounds have a sparse distribution in the time-frequency domain, meaning there is less overlap between the different sounds, which makes the separation task easier compared to the time or frequency domain. Finally, different signals can have the same spectrum in the frequency domain (Figure 1.2) and this makes the task of identifying sources tricky.

1.2.1 Time Frequency Analysis and Synthesis

The conversion of the raw temporal audio data to the time-frequency domain requires applying some analysis methods. The most basic and commonly used method in audio processing is called the **Short Time Fourier Transform (STFT)**. At the end of the processing pipeline, reconstructing the audio signal from the TF domain is done using a synthesis method which is the inverse transform of the analysis method (e.g. Inverse STFT).

Short Time Fourier Transform

Short-time Fourier transform (STFT) is a sequence of Fourier transforms of a windowed signal. It provides the time-varying frequency information in local sections of the signal. This time-frequency representation is the most used in audio processing because of its simplicity and low computational complexity compared to other representations.

The STFT of a signal is computed as follows :

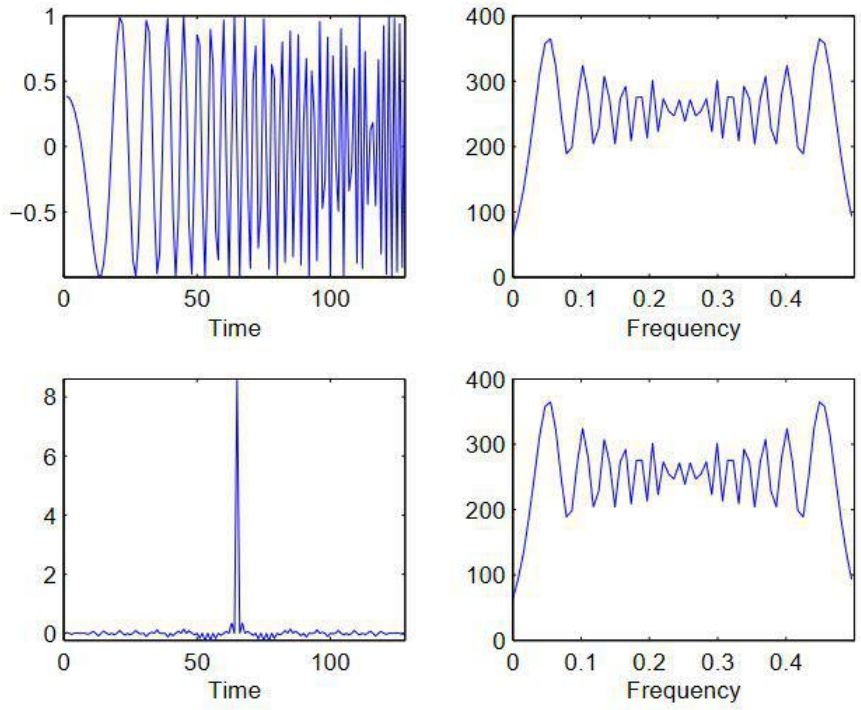


Figure 1.2: Two different signals in time domain that have the same representation in the frequency domain [19].

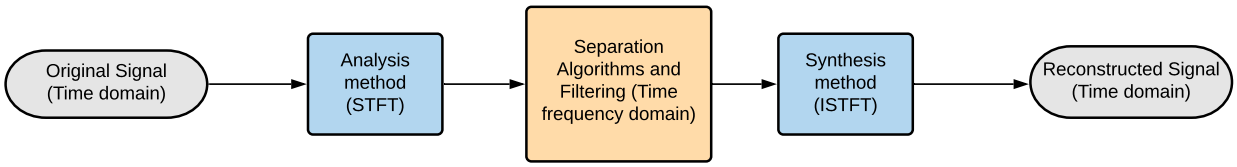


Figure 1.3: Pipeline of the conversion between time and time-frequency domain.

1. Segmentation of the signal into frames with a fixed-length T. Its typical values in audio processing fall between 10 ms to 120 ms. The different segments overlap to reduce the artifacts in the boundaries. The common overlap values are between 50% - 75%.

   Artifacts: Error or anomaly in the representation of a signal that is a result of digital signal processing.

2. Windowing the signal with an appropriate window function. The motivation behind windowing is to reduce the spectral leakage problem where the energy of one frequency component leaks to the adjacent frequency bins. This problem is a result of applying the Fourier transform on a short segment. The windowing in signal processing is a convolution operation of the signal spectrum with the discrete Fourier transform of the window, and a multiplication operation of the signal and the windows function which can be formulated

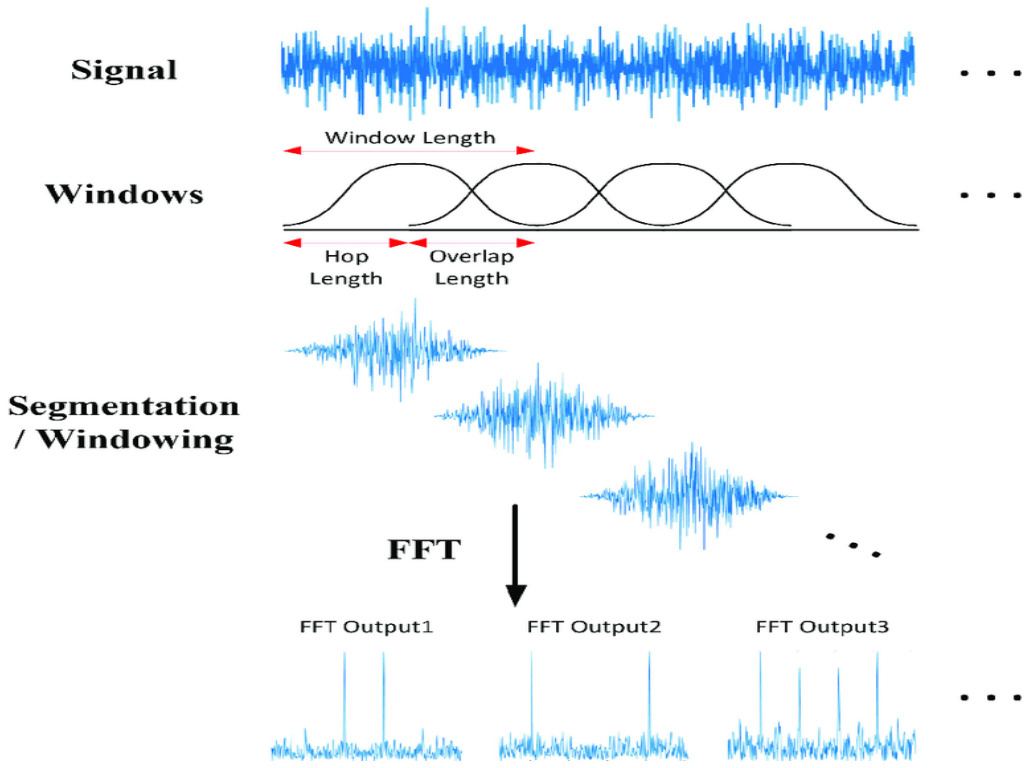


Figure 1.4: Short-time Fourier transform analysis [20].

as follow:

$$x(n,t) = x(t + t_0 + n \cdot M) \cdot h_a(t),\ t \in \{0,...,T-1\},\ n \in \{0,...,N-1\} \tag{1.2}$$

N : Number of time frames.
T : Number of samples in a frame.
t_0 : Position of the first sample of the first frame.
M : Number of samples between adjacent frames.
$h_a(t)$: Window function.

3. Applying the Discrete Fourier Transform (TFD) on each windowed frame resulting in complex values Short Time Fourier Transform coefficients.

$$x(n,f) = \sum_{t=0}^{T-1} x(n,t) \cdot e^{-j \cdot 2 \cdot \pi \cdot t \cdot f / F},\ f \in \{0,...,F-1\} \tag{1.3}$$

F : The number of frequency bins.
f : Discrete frequency bin.

In general , the number of frequency bins is equal to the number of samples in a frame $F = T$. Values of F larger than T can also be taken into account by applying zero padding on $x(n,t)$ as follows :

$$x(n,t) = 0,\ t \in \{T, ..., F-1\} \tag{1.4}$$

The frequencies $f \in \{0, ..., [\frac{F}{2}]\}$ denotes the positive frequencies, and $f \in \{[\frac{F}{2}]+1, ..., F-1\}$ the negative ones. It should be noted that the STFT coefficients of the positive frequencies are the complex conjugate of the coefficients of the negative frequencies. For this reason, most papers often disregard the negative frequencies in their work.

Inverse Short Time Fourier Transform

After performing separation or enhancement algorithms on the STFT of the signal, an inverse operation is needed to reconstruct the signal in the time domain from the modified spectrogram $\hat{S}(n,f)$. In the case of STFT, this operation is called the Inverse Short Time Fourier Transform (ISTFT).

Following is the synthesis process using the inverse STFT :

1. Convert the frames of the STFT to the time domain using the inverse discrete Fourier transform (IDFT).

$$\hat{S}(n,t) = \frac{1}{F} \cdot \sum_{f=0}^{F-1} \hat{S}(n,f) \cdot e^{j \cdot 2 \cdot \pi \cdot t \cdot f/F},\ t \in \{0, ..., T-1\}. \tag{1.5}$$

2. The filtering applied in the time-frequency domain to obtain the target signal can create some artifacts, mostly in the frame boundaries. Attenuating these artifacts is done using windowing with an appropriate window.

3. Sum the overlapped frames to get the estimated signal in the time domain $\hat{s}(t)$

$$\hat{s}(t) = \sum_{n=0}^{N-1} x(n, t - t_0 - n \cdot M) \cdot h_s(n, t - t_0 - n \cdot M) \tag{1.6}$$

h_s : Synthesis window function.

Figure 1.5 illustrates the different steps in the synthesis process.

The analysis and synthesis windows should satisfy the perfect reconstruction property. Meaning that for an unchanged STFT of the original signal $\hat{S}(n,f) = \hat{X}(n,f)$, the estimated signal after the synthesis should be equal to the original signal $\hat{s}(t) = x(t)$. This property can be achieved by taking the synthesis and analysis windows as follows :

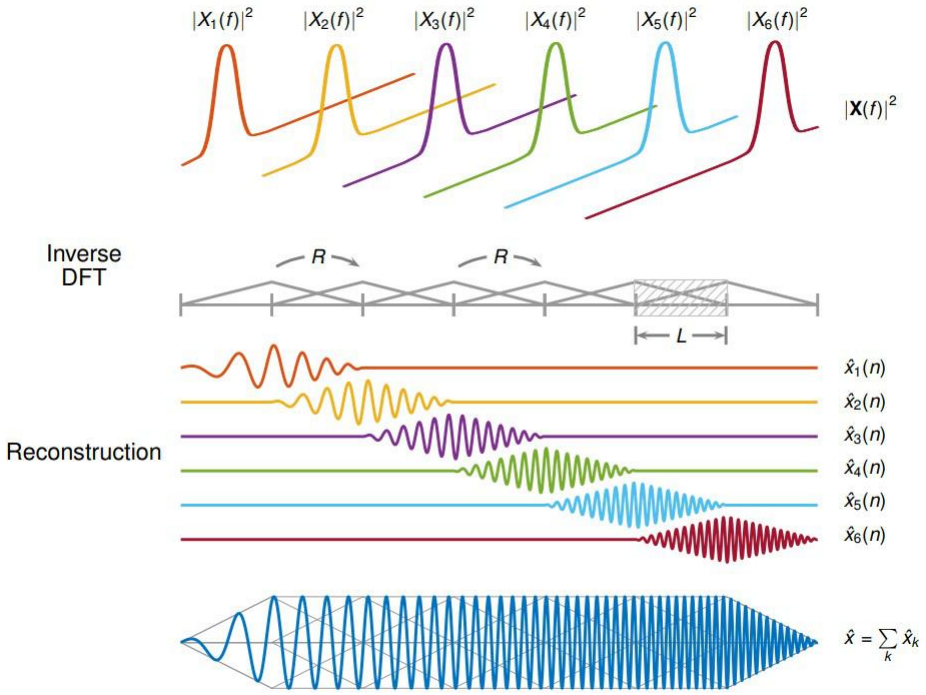


Figure 1.5: Synthesis process using inverse STFT.

$$\sum_{n=0}^{N-1} h_a(n, t - t_0 - n \cdot M) \cdot h_s(n, t - t_0 - n \cdot M) = 1 \tag{1.7}$$

Time and frequency resolution trade-off

One of the most important properties to consider is the time and frequency resolution of a representation. There is an uncertainty principle also called Heisenberg principle that holds for a function and its Fourier transform that can be written for a function that has a normalized energy as follows :

$$\underbrace{\left(\int_{-\infty}^{\infty} t^2 |f(t)|^2 dt\right)}_{(1)} \cdot \underbrace{\left(\int_{-\infty}^{\infty} u^2 |\hat{f}(u)|^2 du\right)}_{(2)} \geq \frac{1}{16\pi^2} \tag{1.8}$$

Term (1): Measures the spread of the signal across time.
Term (2): Measures the spread of the signal across frequencies.

This principle tells us that the signal cannot have a sharp representation in both time and frequency. This trade-off is controlled in the STFT by the choice of the window $h_a(t)$. A very narrow window size (small T) implies a high temporal resolution because the window is very localized. However, this means a very poor resolution in the frequency domain. The inverse can

be observed when taking a large window size T. Figure 1.6 illustrates the impact of the window size on the resolution.

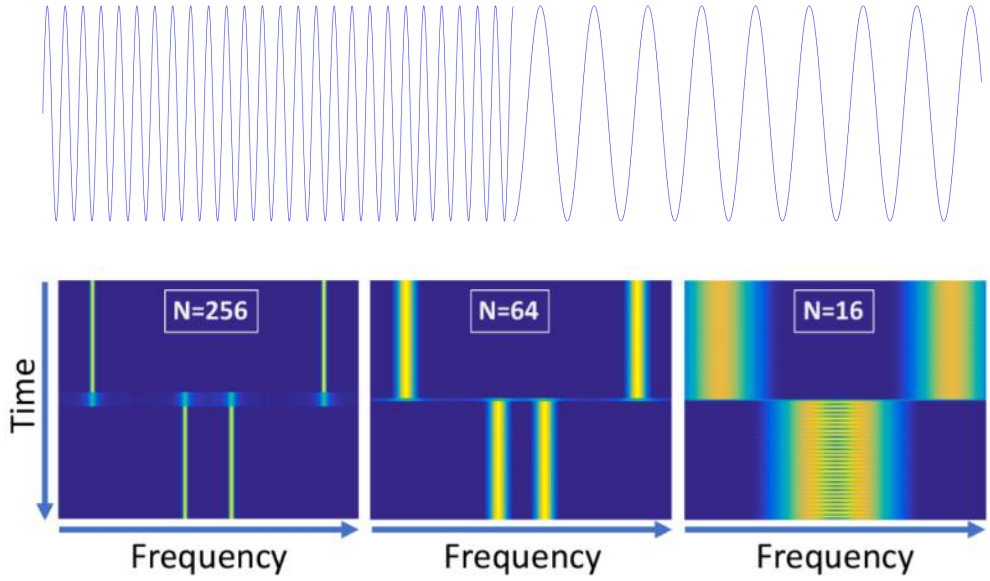


Figure 1.6: Impact of the window size T on the frequency and time resolution.

Another parameter that impacts the frequency resolution is the shape of the window. Bell-shaped windows are widely used in signal processing for their capacity of attenuating artifacts. However, they still have lower frequency resolution compared to rectangular windows. Figure 1.7 compares the frequency resolution of the STFT representation using two different windows: Hanning and a rectangular window. We notice that the rectangular window has a more sharp frequency representation (high resolution) but it suffers from artifacts (side lobes) that are due to the discontinuity in the shape of the window and the convolution with *Sinc*. On the right side, a Hanning window was used as an analysis window and we notice a strong attenuation of the side lobes (artifacts) thanks to the bell smooth shape of Hanning. But, the frequency bands became larger, meaning the frequency resolution is worsening.

1.2.2 Problem Formulation in Time Frequency Domain

The source separation problem can be formulated in the time frequency domain as follows:

Let $Y(t, f)$ be the STFT of the mixture signal $y(t)$ and (t,f) are the frame index and the frequency index respectively.

$$Y(t, f) = S_1(t, f) + S_2(t, f) \tag{1.9}$$

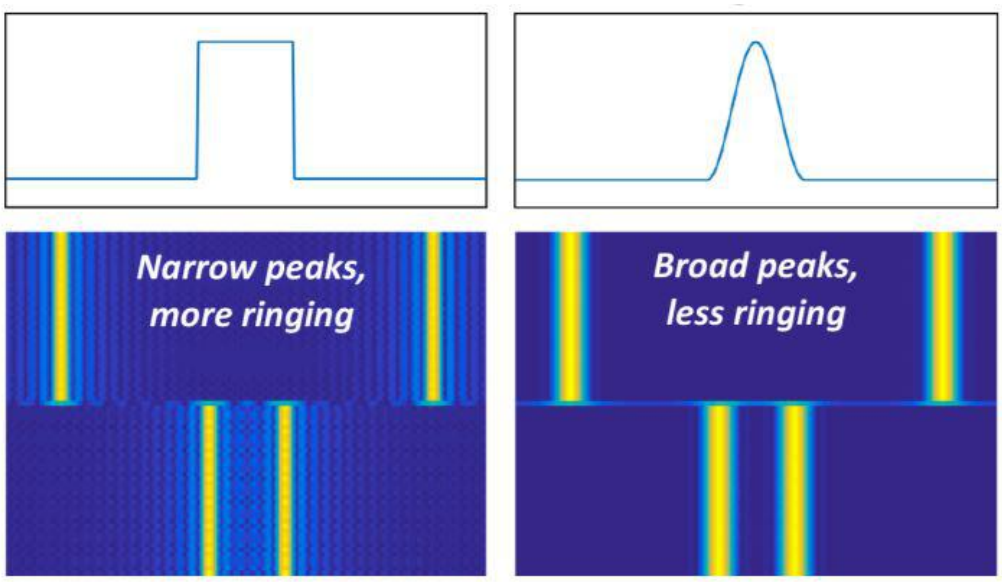


Figure 1.7: Impact of the window shape on the frequency resolution and the artifacts.

where $S_1(t,f)$ and $S_2(t,f)$ are the unknwon STFT of the two sources: speech and music.
The separation algorithms that will be presented in the next chapters are applied in the magnitude spectrum domain. Therefore, the phase of the STFTs are ignored during the separation phase and it will be considered at the end to reconstruct the audio signal. Hence, the magnitude spectrum of the mixture signal can be written as a sum of the magnitude spectrums of the different sources present in the mixture.

$$|Y(t,f)| \approx |S_1(t,f)| + |S_2(t,f)| \tag{1.10}$$

The purpose of our separation methods is to estimate the unknown magnitude spectrums $|S_1(t,f)|$ and $|S_2(t,f)|$ using their mixture and the training data.

1.3 Model based and Data Driven approaches

1.3.1 Model-Based Approach

Model-driven AI, attempts to capture knowledge and derive decisions through explicit representation and rules. These traditional algorithms are usually highly interpretable because they are developed via modeling the physical processes underlying the problem and/or capturing prior domain knowledge.
Moreover, the identification of the limitations, thresholds and potential failure cases of designed systems, is crucial in areas such as medical applications and autonomous driving where interpretability plays a fundamental role.

Furthermore, Model-driven approaches are powerful because they rely on a deep understanding of the system or process, and can benefit from scientifically established relationships. Models can't accommodate infinite complexity and generally must be simplified.

Minimal Example

For example, in a system recognition using the model-based method, a cat would be explicitly represented as a four-legged animal, with two eyes, a nose and a mouth, with a relatively small height. The model would look at the image, deconstruct it into lines and shapes and colours and compare that against the set of rules we've supplied about how lines and shapes and colours combine in the world to give us different animals.
To approach this recognition task, we could use the *Scale-Invariant feature Transform* (SIFT) [21], which computes SIFT descriptors, that will characterize the visual content of the target image. For example, two images of the same content would have a high probability of having the same SIFT descriptors.

Advantages

- System-oriented approach is deterministic, which means that it does not include elements of randomness, because it provides the necessary theoretical basis for examining the relative importance of various factors that influence the output target.
- Highly interpretable.
- Problem domain-knowledge that results from the analytical expressions that describe the phenomena and the different assumptions.

Drawbacks

- High cost of implementation.
- Difficult to apply on complex systems.

1.3.2 Data-Driven Approach

Current, tech-industry interest in artificial intelligence is almost entirely focused on data-driven AI. The reasons are easy to understand

- Cheap data storage.
- Powerful computational resources (e.g. GPU).
- Advancements in neural net algorithms.

- Open source libraries and frameworks (e.g. Tensorflow, Pytorch).
- Availability of open and large datasets.

These three main causes have made it possible to extract huge value out of data. We can build systems that can predict what will happen next based on what they've seen so far, very efficiently. Their performances are even better than human being one's, and model-based methods.
As cited before, the use of data as an alternative to physical knowledge has attracted more and more interest throughout the years, since it is often cheaper and less time-consuming.
Deep neural networks provide surprising performance gains in many real-world problems in signal and image processing. Despite these performances, the black-box nature of deep neural networks is a hindrance to this kind of method.
This kind of approach attempts to automatically discover model information and incorporate the data by optimizing model parameters that are gleaned from real-world training samples, learning the rule that maps input data to target data. Neural Networks generally adopt a hierarchical architecture composed of many layers including a large number of parameters capable of learning complicated mappings, which are difficult to design explicitly. When training data is largely available, this sufficient amount often enables deep networks to overcome model shortcomings, especially when the underlying physical scenario is difficult to accurately define. They are implemented very quickly, for example, convolutional neural networks that use convolution make the process computationally efficient. In addition, the number of layers in a deep network is usually much less than the number of iterations required in an iterative algorithm. Therefore, deep learning methods have emerged to offer desirable computational benefits over state-of-the-art approaches in many areas of signal processing, imaging, and vision.

It is therefore difficult to discover what is learned within networks by examining network parameters, which are usually high dimensional, and what are the roles of individual parameters. In other words, general deep networks are difficult to interpret. It is well known that the practical success of deep learning sometimes depends excessively on the quantity and quality of training data available. In scenarios where training samples are not available, such as in medical imaging, the performance of deep networks may deteriorate significantly. This deterioration can be explained by the overfitting phenomena **overffiting**, where a data-driven model became super-specialized, and therefore cannot generalize, consequently requiring a large amount of data to get decent results.

Minimal Example

The question a data-driven is asking itself in a binary classification problem is:

" Given these data input, should my weights send a stronger signal to the cat or the dog?"

But whenever the data-driven method wants to adjust its parameter during the back-propagation [22] in order to learn the distribution of a specific class, the model asked itself:

"Given a cat, which distribution of data should I expect?"

Advantages

- Simplicity of implementation.
- Low cost.
- Open source deep learning libraries and frameworks (e.g. Tensorflow, Pytorch) that enable using hardware accelerators and parallel processing.

Drawbacks

- Brute Force technique making its parameters difficult to interpret, consequently, it's strained to define the failure of the model. Even though many research studies attempt to interpret these methods such as interpreting the convolutional layers of Alexnet in [23], these methods are still far from being interpretable especially with their architecture that is going more and more complex and deep.
- Needs a lot of data over a lot of iterations, to learn the rule that maps inputs data to targets.

1.4 Deep Learning

1.4.1 Biological Neurons

Before digging into neural networks, it is important to consider the authors inspirations who originated the idea. Authors inspired their studies from the nervous system. The nervous system is a net of neurons, where each neuron having a soma (buble containing the cell nucleus) and an axon as illustrated in figure 1.8. Their synapses are between the neuron and the soma of another. At every timestamp t, a neuron has some threshold, which excitation must exceed to initiate an impulse. This, except for the fact and the time of its occurence, is determined by the neuron and its neighbors, not by the excitation, meaning that neurons excitations depends upon structural properties of the net. Authors of [24] presented a simplified computational model of how

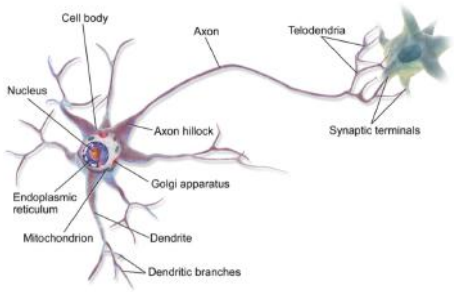


Figure 1.8: Biological neuron.

biological neurons might work to perform complex computations. Thus individual biological neurons seem to behave in a rather simple way, they are organized in a vast network of billions, with each neuron typically connected to thousands of other neurons their models 'neuron' has two states, on or off, it sums activation from other neurons, and it enters the on-state when the sum of its inputs exceeds a threshold. Networks composed of such abstract "neurons" can compute any finite logical expression. The architecture of biological neural networks is still the subject of active research, but some parts of the brain have been mapped, and seem that neurons are often organized in consecutive layers as shown in Figure 1.9.

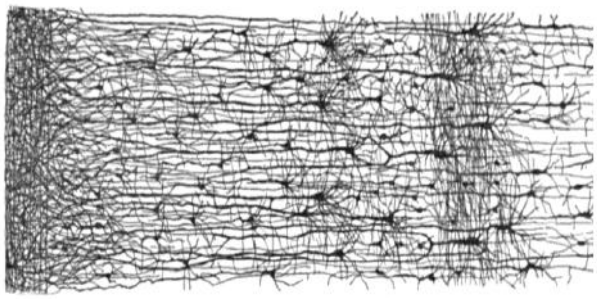


Figure 1.9: Organization of neurons in the brain.

1.4.2 Perceptron

The perceptron is one of the simplest artificial neural network architectures [25], primarily used for binary and multioutput classification. The perceptron consists of 4 parts:

- Input values or input layer.
- Weights and Bias.
- Net sum.
- Activation Function.

In summary, the perceptron is based on a threshold logic unit TLU, each input connection is associated with a weight that will be used as a ponderation, this latter expressed the strength of the particular node. In the other hand, the bias b allows to shift off the TLU curve up or down.

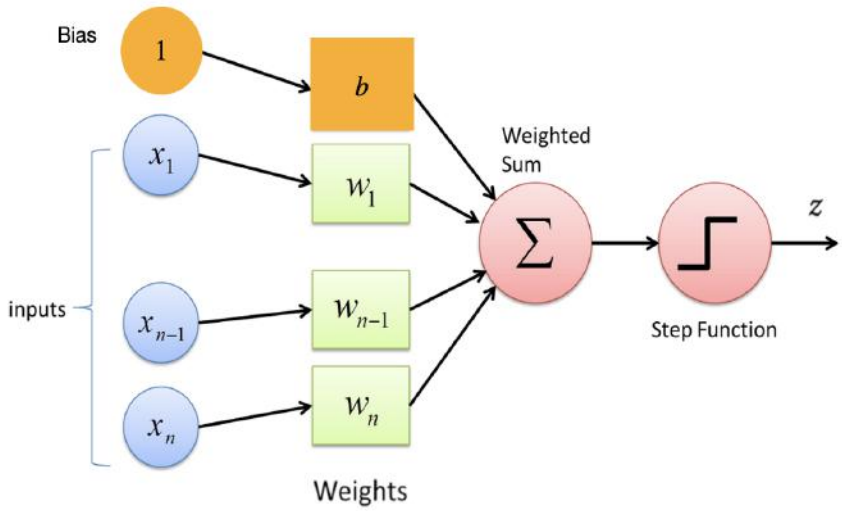


Figure 1.10: Perceptron's diagram

The most common step function used in Perceptron is the heaviside function.

$$\text{heaviside}\ (z) = \begin{cases} 0 \text{ if } z < 0 \\ 1 \text{ if } z \geq 0 \end{cases} \tag{1.11}$$

The TLU computes a weighted sum of its inputs then applies a step function,as shown below:

$$\mathbf{z}_{W,b}(\mathbf{X}) = \phi(w_1 x_1 + w_1 x_1 + \cdots + w_n x_n + b) = \phi(\mathbf{XW} + \mathbf{b}) \tag{1.12}$$

X: input data feeded to the perceptron.
W: weight matrix of the perceptron.
b: bias of the model.
ϕ: TLU's function.

A perceptron can also be composed of multiples TLUs, with each TLU connected to all the inputs, making a fully connected layer between the input nodes and the output layer. allowing the perceptron to behave for example like a multi-output classifier. However, Perceptron networks have several limitations:

- The output values of a perceptron can take on only one of two values (0 or 1) due to the hard-limit transfer function, We can overcome this problem by introducing a new type of artificial neuron called a sigmoid neuron [26], or using a non-linear activation function like ReLU [27] which is defined latter in subsection 1.4.5.
- Perceptrons can only classify linearly separable sets of vectors [28]. If a straight line or a plane can be drawn to separate the input vectors into their correct categories, the input vectors are linearly separable. If the vectors are not linearly separable, learning will never reach a point where all vectors are classified properly.

1.4.3 Multi Layer Perceptron

A Multilayer Perceptron is a finite acyclic graph, its nodes are neurons activated using the logistic activation. The Multi-Layer Perceptron that was initially proposed by [29], is an improvement over the Perceptron, composed of an input layer, one or more intermediate layers called hidden layers, and an output layer. The connections that hop over several layers are called a shortcut, all neurons of one layer are connected to all neurons of the next layer, while neurons of the same layer are independents.

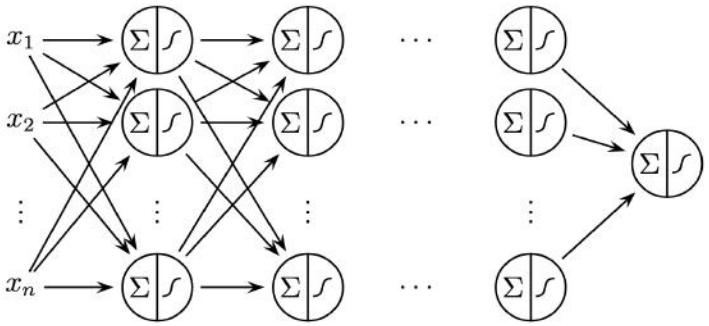


Figure 1.11: Architecture of a MLP

As we saw earlier, the Perceptron, calculates a discontinuous function which is a step function.

The Multi-Layer Perceptron, computes a smoothed variant of this using the logistic function often called the sigmoid activation [30] illustrated in Figure 1.12, which has the characteristics below:

- Monotonically increasing.
- $\lim_{z \to \infty} = 1$
- $\lim_{z \to -\infty} = 0$
- $sigmoid(z) = 1 - sigmoid(-z)$
- Continuous, and differentiable.

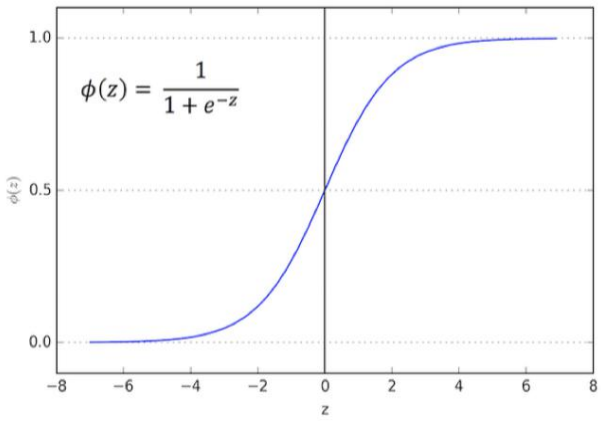


Figure 1.12: Logistic function

The (Heaviside) step function is typically only useful within single-layer Perceptron, in cases where the input data is linearly separable. However, MLP stands out from perceptron because they use function approximators and it can distinguish data that is not linearly separable thanks to the logistic function. It is also important to note that MLP is trained using backpropagation (which we'll see in the next subsection), which requires the activation function to be differentiable. That's because backpropagation is used to update the network weights using the gradient of the error. That said, The Heaviside step function is non-differentiable at $x = 0$ and its derivative is 0 elsewhere, meaning that it contains only a flat segment. It means that backpropagation will fail to update weights because there will be no gradient to backpropagate (Gradient Descent cannot move on a flat surface). The sigmoid or logistic function does not have this shortcoming and this explains its usefulness as an activation function.

Effect of hidden Layer

The hidden layers extract data from one set of neurons (input layer) and provide the output to another set of neurons (output layer). An Multi-Layer Perceptron with just one hidden layer can

theoretically model even the most complex functions, provided it has enough neurons. But for complex problems, deep networks have a much higher parameter efficiency than shallow ones: they can model complex functions using exponentially fewer neurons than shallow nets, allowing them to reach much better performance with the same amount of training data. It also helps the model to converge faster to a good solution, but it also improves their ability to generalize to new datasets Simply said, the hidden layers, as they go deeper, capture all the minute details. This results in discovering various relationships between different inputs. So, in theory, deeper networks (more hidden layers) are more complex as they develop a more granular/detailed representation of the data input.

1.4.4 Backpropagation

The backpropagation algorithm is probably the most fundamental building block in a neural network. Authors of [22] proposed this algorithm to effectively train a neural network through a method called chain rule. In simple terms, after each forward pass through a network, backpropagation performs a backward pass while adjusting the model's parameters represented by its weights and biases. The method repeatedly adjusts the weights of the connections in the network to minimize a measure of the difference between the actual output vector of the net and the desired output vector. In simple words, backpropagation aims to minimize a cost function, this can be as simple as MSE (mean squared error) or more complex like the cross-entropy, by adjusting the network's weights and biases. The level of adjustment is determined by the gradients of the cost function concerning those parameters. In other words, it can find out how each connection weight and each bias term should be tweaked to reduce the error. Once it has these gradients, it just performs a regular Gradient Descent step, and the whole process is repeated until the network converges toward a solution. Let's run through this algorithm in a bit more detail:

- It handles one mini/stochastic/ batch at a time, and it goes through the full training set multiple times. Each pass is called an epoch.
- The network computes the output of all the neurons for every layer for each batch. The result is passed on to the next layer, and so on, until we get the output of the last layer, ie: the output layer. This is the forward pass.
- Next, the algorithm measures the network's output error by comparing the desired output and the actual output of the network and returns some measure of the error.
- Then, it computes how much each output connection contributed to the error.
- Finally, the algorithm performs a Gradient Descent step to tweak all the connection weights in the network, using the error gradients it just computed.

Minimal Example of Algorithm

In order to explain the mathematics behind the backpropagation algorithm, let's study a minimal example of a MLP with two inputs, two hidden neurons, two output neurons. Additionally, the hidden and output neurons will include a bias, the architecture is illustrated in Figure 1.13

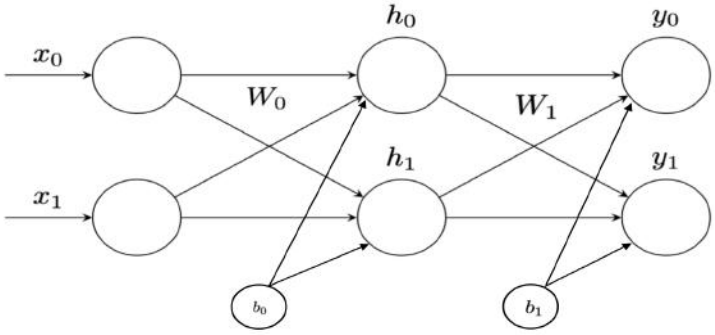


Figure 1.13: Architecture of an MLP with two inputs neurons, two hidden neurons, and two output neurons

As we said, before backpropagation aims to optimize weights and bias of the architecture, let's write down the equation (Eq. 1.13) which governs the model behavior.

Forward Pass

$$\mathbf{H}_{W_0,b_1} = \sigma(\mathbf{X}^T\mathbf{W}_0 + \mathbf{b}_1) = \sigma(\begin{bmatrix} h_1 \\ h_2 \end{bmatrix} = \begin{bmatrix} w_1 & w_2 \\ w_3 & w_4 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} + b_1) \tag{1.13a}$$

Where σ represents the logistic function we defined earlier in subsection 1.4.3, we repeat this process for the output layer neurons, using the output from the hidden layer neurons as inputs.

$$\mathbf{Y}_{W_1,b_2} = \sigma(\mathbf{H}^T\mathbf{W}_1 + \mathbf{b}_2) = \sigma(\begin{bmatrix} y_1 \\ y_2 \end{bmatrix} = \begin{bmatrix} w_5 & w_6 \\ w_7 & w_8 \end{bmatrix} \begin{bmatrix} h_1 \\ h_2 \end{bmatrix} + b_2) \tag{1.13b}$$

Calculating the Total Error

The next step, is to calculate the error that each neuron contributes using the mean squared error for example , and sum them up to get the total error.

$$E_{total} = E_{y_1} + E_{y_2} = \frac{1}{2}\left(y_{true1} - y_1\right)^2 + \frac{1}{2}\left(y_{true_2} - y_2\right)^2 \tag{1.13c}$$

The $\frac{1}{2}$ is included so that exponent is cancelled when we differentiate later on.

The backward Pass

Our goal now, is to propagate the error contribution on each neuron of the model by updating its weights and eventually the bias of the architecture.

Output Layer

$$\frac{\partial E_{\text{total}}}{\partial W_{1,i}} = \frac{\partial E_{\text{total}}}{\partial y_i} \frac{\partial y_i}{\partial(\sigma y_i)} \frac{\partial(\sigma y_i)}{\partial W_{1,i}} \tag{1.13d}$$

To decrease the error, we then subtract this value from the current weight (optionally multiplied by some learning rate η), and apply this for every weight of the matrix $\mathbf{W_1}$

$$W_{1,i}^{+} = W_{1,i} - \eta \cdot \frac{\partial E_{\text{total}}}{\partial W_{1,i}} \tag{1.13e}$$

Hidden Layer

Next, we'll continue the backwards pass by calculating new values for weights of the matrix $\mathbf{W_0}$

$$\frac{\partial E_{\text{total}}}{\partial W_{0,i}} = \frac{\partial E_{\text{total}}}{\partial h_i} \frac{\partial h_i}{\partial(\sigma h_i)} \frac{\partial(\sigma h_i)}{\partial W_{0,i}} \tag{1.13f}$$

To decrease the mean squared error, we repeat the process using the gradient of the error

$$W_{0,i}^{+} = W_{0,i} - \eta \cdot \frac{\partial E_{\text{total}}}{\partial W_{0,i}} \tag{1.13g}$$

In summary, the backpropagation algorithm first makes a prediction (forward pass) and measures the error, then goes through each layer in reverse to measure the error contribution from each connection (reverse pass), and finally, tweaks the connection weights to reduce the error (Gradient Descent step).

1.4.5 Rectified Linear Units

One of the insights in the 2010 wrote by *Glorot and Bengio* [31] was that the problems with unstable gradients were in part due to a poor choice of activation function. Until then most people had assumed that if Mother Nature had chosen to use roughly sigmoid activation functions in biological neurons, they must be an excellent choice. But it turns out that other activation func-

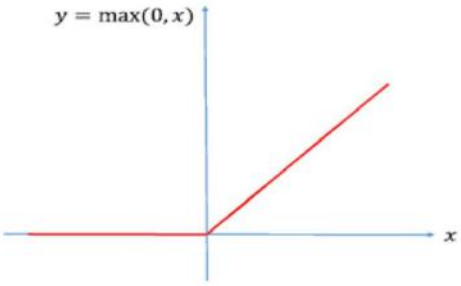
tions behave much better in deep neural networks—in particular, the ReLU activation function, because it does not saturate for positive values and because it is fast to compute. ReLU is an activation function introduced in [27]. In 2011 , it was demonstrated to improve training of deep neural networks. This non-linear function works by thresholding values at 0.

$$y = \max(0, x). \tag{1.14}$$

Simply put, it outputs 0 when $x < 0$, and conversely, it outputs a linear function when $x \geq 0$ as shown in Figure 1.14.

Note

It is important to initialize all the hidden layers connection weights randomly or using initialization strategy, or training will fail. For example, if you initialize all weights and biases to zero, then all neurons in a given layer will be perfectly identical, and thus backpropagation will affect them in the same way. If instead we randomly initialize the weights, we break the symmetry and allow backpropagation to train a diverse team of neurons.



x

Figure 1.14: Rectified Linear Unit

1.4.6 Dropout

Deep neural networks contain multiple non-linear hidden layers and this makes them very expressive models that can learn very complicated relationships, requiring some sort of regularization, which includes for example:

- Early stopping: stopping the training as soon as performance on a validation set starts to get worse.
- L1 or L2 penalties to regularize model weights, by adding “absolute value of magnitude” and “squared magnitude” respectively to the weights parameters of the model.

However, techniques exist such as Model Combination and Bagging that improve the performance of machine learning methods. While the first works as multiple predictors that behaves as a single predictor, Bagging trains multiple models on multiple subsets of the training data Thus some training examples are not shown to a given model. With large neural networks, those techniques might be expensive. The combination of models is efficient and practical only when models learn different features from the data; extract different information from the input, so these models should either be trained on different data or having different architectures to express something different. Moreover, large networks require large amounts of training data, making the training of multiple large networks difficult and computationally inefficient. Even if one was able to train many different large networks, using them all at test time is infeasible in a real-time application where quick response is more important than the application itself.
Dropout is a technique that addresses both these issues as cited in [32] which proposed a technique "by training all sub-networks of a model that can be formed by removing non-output units from an underlying base network." Dropout refers to the probability p of a random neuron of the model (excluding outputs neurons), being temporarily ignored during every training step. The hyperparameter p is called the dropout rate and is typically set between 0.1 and 0.5. Dropout prevents overfitting and provides a way of approximately combining exponentially many different neural network architectures efficiently. It's important to note, that dropout is used only in the training stage of the neural network, and absent in the validation and test stage.

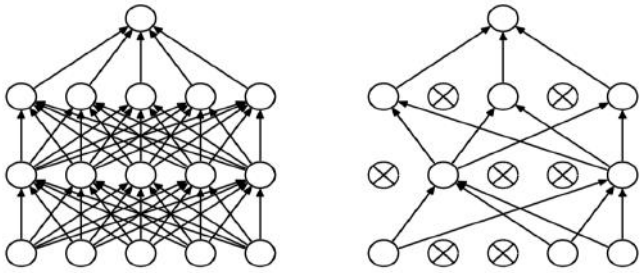


Figure 1.15: Schema representing the dropout

Probability mass Function

Probability mass function (PMF) is the probability distribution of a discrete random variable, and provides the possible values and their associated probabilities that must be positive and sum up to 1.

$$\sum p_X(x_i) = 1, \text{with } \; p(x_i) > 0 \tag{1.15}$$

Where X denotes the random variable, $p(x_i)$ denotes the probability of the event x_i to occur.

Bernoulli Random Variables

A Bernoulli random variable can take only two values, 1 and 0. It takes on a 1 if an experiment with probability p resulted in success and a 0 otherwise. If X is a Bernoulli random variable, denoted $X \sim \text{Bernoulli}(p)$:
Probability mass function of X:

$$P(X = x) = p^x(1-p)^{1-x} \tag{1.16}$$

A random variable is then called a Bernoulli random variable if it has a PMF for p between 0 and 1.

Minimal Example of Dropout

In order to be concise, lets describe how the dropout operation occurs in neural network model. Consider a neural network of L hidden layers with $l \in \{1...L\}$, let x denotes the vector of inputs of layer l, and $z_i^{[l+1]}$ denotes the neuron i of the $l+1$ layer, $y_i^{[l+1]}$ denoting the activation of the latter neuron.$w_i^{[l+1]}$ and $b_i^{[l+1]}$ denotes respectively the weight matrix and the bias vector at the $l+1$ layer as shown in Figure 1.16.

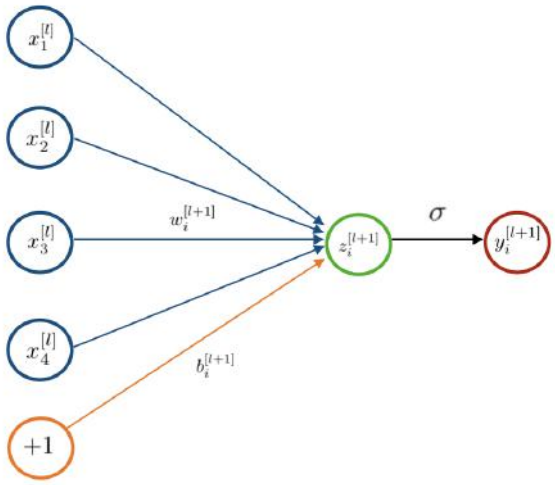


Figure 1.16: Diagram of a neural network model without dropout.

$$z_i^{(l+1)} = \mathbf{w}_i^{(l+1)}\mathbf{y}^l + b_i^{(l+1)},$$
$$y_i^{(l+1)} = \sigma\left(z_i^{(l+1)}\right)$$

where σ is any activation function.

With dropout, the feed-forward operation becomes (Figure 1.17):

$$\begin{aligned}
\gamma_j^{[l]} &\sim \text{Bernoulli}(p) \\
\widetilde{\mathbf{x}}^{[l]} &= \gamma^{[l]} \odot \mathbf{x}^{[l]} \\
z_i^{[l+1]} &= \mathbf{w}_i^{[l+1]} \widetilde{\mathbf{x}}^{l} + b_i^{[l+1]} \\
y_i^{[l+1]} &= \sigma\left(z_i^{[l+1]}\right)
\end{aligned}$$

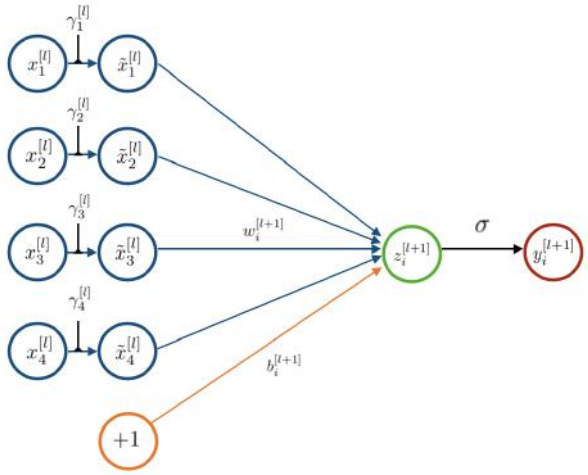


Figure 1.17: Dropout applied on the input of the neural network before the sum and activation.

Here $\odot$ denotes an element-wise product. For any layer l, $\gamma^{(l)}$ is a vector of independent Bernoulli random variables each of which has probability p of being 1 or 0. Dropout uses independent Bernoulli so that the dropout of a random neuron will not affect the other neuron. In other words, it will not affect the probability of another event happening.

$$P\{drop(\mathbf{x}_i) \mid drop(\mathbf{x}_j)\} = P\{drop(\mathbf{x}_i)\}$$

$\gamma^{(l)}$ is then sampled and multiplied element-wise with the outputs of that layer, $\mathbf{x}^{(l)}$, to create the masked outputs $\widetilde{\mathbf{x}}^{(i)}$, which will be then used as input to the next layer. This process is applied at each layer. For learning during the training stage, the derivatives of the loss function are back-propagated only through the neurons that were not dropped that constituted the sub-network. Finally, The resulting neural network is used without dropout for the inference.

1.4.7 Optimizers

Deep learning algorithms require optimization to find the set of learnable parameters Θ that minimizes the cost function $\mathbf{J}(\mathbf{\Theta})$ that represents the performances of the model in the training phase.
The purpose of optimization in machine learning and deep learning, in particular, is different from traditional optimization in mathematics where minimizing the cost function J is a goal in and of itself. In Machine learning, we intend to improve a performance measure also called the metrics M indirectly by training the model to reduce the cost function J.

Various research studies have been conducted to find algorithms and techniques that solve many challenges in optimizing Deep Neural Networks such as vanishing and exploding gradients, local minima, and other flat region points such as saddle points.

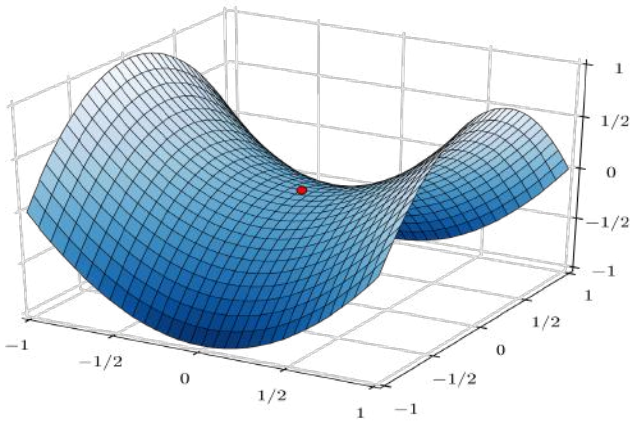


Figure 1.18: Saddle point where the derivatives in two orthogonal directions are equal to zero but the point is neither a local maximum nor a local minimum.

Stochastic, Batch and Mini batch

The cost function can be decomposed as a sum over the training samples. This results in 3 different approaches to estimate the cost function :

- **Batch algorithms** : Such as batch gradient descent use all the training data simultaneously to estimate the cost function.
- **Stochastic algorithms** : In contrast to batch algorithms, stochastic algorithms process each training sample at a time like the stochastic gradient descent.
- **Mini batch algorithms** : The most widely used optimization methods in Deep learning use a number of samples more than one (Stochastic) and less than the training data size (Batch). This type of methods combined the advantages of both stochastic and batch algorithms.

The choice of the batch size (number of samples in a mini batch) depends on some theoretical factors and hardware setup. The following are the most significant ones :

1. Increasing the batch size gives a better estimate of the gradients because more training samples are considered.
2. Small batch size have a regularization effect that can be attributed to the noise introduced in the training process. This implies a better generalization ability. In [33], it has been proved using empirical results that the stochastic gradient descent is expected to be at least as accurate as well as faster than batch training. However, the stochastic methods are the more likely to diverge, therefore, they require a small learning rate to maintain their stability which will often result in a slow convergence.
3. Reducing the batch size implies more frequent updates of the parameters that will make the training process slower.
4. Some specialized hardware devices have better results with a batch size equal to a power of 2 (e.g. 16,32,64). In practice, the widely used batch size falls between 16 and 256.
5. In parallel processing, the samples of each batch will be processed in parallel. Therefore, the size of the memory used will increase with the batch size. This point is a limiting factor for choosing a batch size.
6. The batch size is highly dependent on the optimization algorithm. Some algorithms require more samples because they are not robust to sampling errors. This is due to either the difficulty of the approximation they try to perform or the inverse relationship between the sampling error and the number of samples. As instance, second order methods that make use of the second-order derivative Hessian matrix $\mathbf{H}$ to find the optima of an objective function (e.g. Broyden, Fletcher, Goldfarb, and Shanno, or BFGS Algorithm). Such methods estimates H (or its inverse) and computes updates such as $\mathbf{H}^{-1} \cdot \mathbf{g}$ where g is the gradient. Even a very accurate estimate of the hessian matrix, if combined with poor errors of the gradients can increase dramatically the errors in the term $\mathbf{H}^{-1} \cdot \mathbf{g}$. In contrast, first order methods like gradient descent do not suffer from this problem and are robust to sampling errors, this means that even low batch sizes can give good estimates of the gradients.
7. It should be noted that in many practical situations the batch size will be a hyperparameter to tune and its value will be determined empirically.

Shuffling

Another important technique to consider is shuffling the data before feeding it to the Deep learning model. The idea behind this technique is that in estimation theory combining independent

models can lead to better performances, an example of this is Bagging (bootstrap aggregation) that reduces the variance of an estimate by averaging together multiple estimates. In optimizing Deep models, we want unbiased estimates of the gradients, this requires the samples in the same mini batch to be independent and randomly selected. However, shuffling the data is not allowed in some cases when working with time series or text data because it will break the temporal (structure) dependency.

Mini batch Gradient Descent

Mini batch is the version of Gradient descent algorithm that works with mini batch sampling. It is known to have a faster convergence than classical gradient descent and a better estimate of the gradient compared to stochastic gradient descent. Figure 1.19 illustrates the convergence of the three algorithms. It should be noted that the random noise in the SGD behavior helps avoiding local minimums.

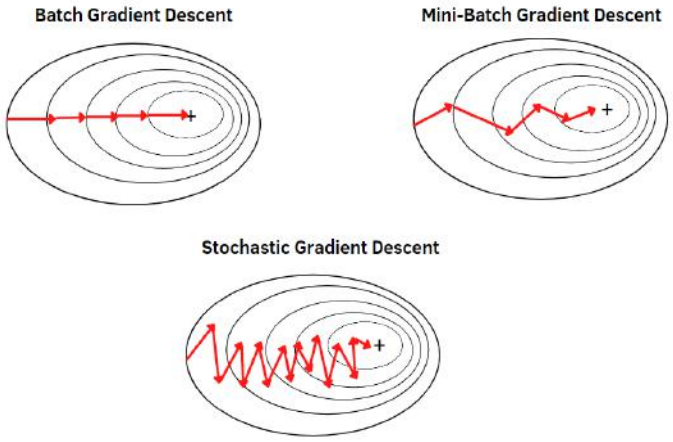


Figure 1.19: Comparison between batch, mini batch and stochastic gradient descent.

Algorithm 1: Mini batch gradient descent update.

Input: Learning rate μ, $\boldsymbol{x}$ is the input, $\boldsymbol{y}$ is the target,$\mathbf{L}$ is the cost value, $\boldsymbol{\theta}$ are the model parameters, and m is the mini-batch size.

for *Sample* (x_i, y_i) *from mini batch where* x_i *is the input and* y_i *is the target* **do**

Gradient Estimate : $\hat{g} \leftarrow \frac{1}{m} \nabla_{\theta} \sum_i \mathbf{L} \left(f \left(\boldsymbol{x}^{(i)}; \boldsymbol{\theta} \right), \boldsymbol{y}^{(i)} \right)$

Update : $\theta_0^+ \leftarrow \theta_0 - \mu \cdot \hat{g}$

end

The implementation of this algorithm is available in popular frameworks such as Pytorch and Tensorflow. It can be used as follows:

```
from torch import optim
optim.SGD(model.parameters(), lr= 0.01)
# lr : Learning rate.
# model.parameters() : Parameters of the Deep Learning model.
```

Figure 1.20: SGD code using Pytorch.

Momentum

Momentum is a technique that makes use of exponentially weighed moving averages in order to provide a better estimate of the gradients and reduce the noisy behavior of the stochastic methods. It helps accelerating the stochastic gradient descent in the right direction by reducing the noisy oscillations. Figure 1.21 illustrated the effect of momentum on the SGD algorithm.

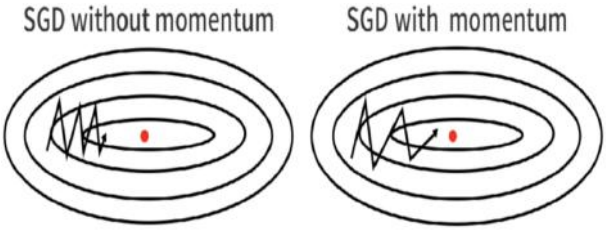


Figure 1.21: The effect of momentum on the SGD.

The update of SGD with momentum is presented in equations 1.17. Where Eq. 1.17a is the moving average of the gradients and Eq. 1.17b is the parameters' update.

$$V_t = \beta V_{t-1} + (1-\beta)\nabla_w L(W, X, y). \tag{1.17a}$$

$$\Theta = \Theta - \mu \cdot V_t \tag{1.17b}$$

The hyperparameter β can take values between 0 and 1 but it is usually picked in the range [0.9-0.99]. High values of β means giving more importance to the memory term, in other words, taking more terms in the moving average.

Adaptative algorithms

The learning rate is one of the most important hyperparameters that can have a significant impact on the performances of deep learning models. The momentum techniques attempts to improve the traditional algorithms. But another problem raises : How can we fix the learning rate when the objective function is more sensitive in some dimensions more than others. Some optimization algorithms have been proposed as a solution to the problem of fixing the same learning rate for all the parameters such as Adaptive Gradient Algorithm [34] and Root Mean Square Propagation [35].

- **Adaptive Gradient Algorithm (Adagrad)** : Adapts the learning rate according to the inverse of the sum of old gradient values. Parameters with high derivatives will have their learning rate decreased more than parameters with low derivatives. However this algorithm is limited because of summing all the gradients from the beginning of the training results often in vanishing the learning rate values.

- **Root Mean Square Propagation (RMSprop)** : Fixes the limitations of Adagrad by using an adaptive learning rate that is adjusted according to the square of the gradients' exponential decaying moving average.

$$\boxed{\begin{aligned} E\left[g^2\right]_t &= \beta \cdot E\left[g^2\right]_{t-1} + (1-\beta) g_t^2 \quad (1.18a) \\ \Theta_{t+1} &= \Theta_t - \frac{\eta}{\sqrt{E\left[g^2\right]_t + \epsilon}} g_t \quad (1.18b) \end{aligned}}$$

  The hyperparameter β controlls the history terms of the gradients and avoid taking into account the values from the begining of the training as it is done in Adagrad. It is suggested to have a default value of 0.9 .

1.4.8 Network's Initialization

He Initialization

In section 1.4.4, we saw that the back-propagation algorithm proceeds in two phases, one forward from the input layer to the output layer, and a back pass which propagates the gradient error in the network layers to update the model parameters of each layer.
The problem with this process constituted of two phases, is that while the algorithm progresses towards lower layers, gradients becomes lower and lower, consequently the update of the weights will not happen because the gradient will vanish making the convergence of the model towards a good solution difficult, if not impossible, this phenomena is called **gradient vanishing**. The opposite can happen, making the gradient extremely large which makes the model diverges, but this situation is often observed on recurrent networks, called gradient explode. In summary, deep neural network suffer from gradient instability making the learning very hard, which was the main reason why deep learning was abandoned.
Authors of [36] proposed a solution to solve this problem, by allowing the gradient flow to propagate correctly through all layers to enhance the learning as well as allowing a faster convergence. It also seems that the He initialization is specific to RELU activation. The idea is to bring the variance of every output layers to approximately one. So Authors of [36] claims that the best weight initialization strategy to a model that use RELU activation which is non-differentiable at

$x = 0$, is to initialize the weights randomly with a specific variance, such as:

$$Var(w_l) = \frac{2}{N} \tag{1.19}$$

The central idea of [36] is then to investigate the variance of the responses in each layer:

$$\mathbf{y}_l = \mathrm{W}_l \mathbf{x}_l + \mathbf{b}_l \tag{1.20}$$

Where W_l, $\mathbf{x}_l$, and $\mathbf{b}_l$ represents respectively the weights, the inputs and the bias of the layer l. It is also important to note that $\mathbf{x}_l = f(\mathbf{y}_{l-1})$ where f represents the ReLU activation.
Authors begin with some assumptions:

- Elements in $\mathbf{W}_l$ are mutually independents and share the same distribution.
- As cited in [37], authors assume that elements of $\mathbf{x_l}$ are also mutually independent with the same distribution.
- $\mathbf{x}_l$ and $\mathbf{W}_l$ are independent of each other such $\mathbf{x}_l \perp \mathbf{W}_l$, meaning that the occurence of $\mathbf{W}_l$ does not affect the probability of $\mathbf{x}_l$ and vice versa.

From these assumptions, we can write:

$$\begin{aligned} Var[y_l] &= n_l Var[\mathbf{W}_l \mathbf{x}_l] && \text{(1.21a)} \\ &= n_l \mathbb{E}\{\mathbf{W}_l^2 \mathbf{x}_l^2\} - \mathbb{E}\{\mathbf{W}_l \mathbf{x}_l\}^2 && \text{(1.21b)} \\ &= n_l \mathbb{E}\{\mathbf{W}_l^2 \mathbf{x}_l^2\} - \mathbb{E}\{\mathbf{W}_l\}^2 \mathbb{E}\{\mathbf{x}_l\}^2 && \text{(1.21c)} \end{aligned}$$

Because we let $\mathbf{W}_l$ having zero mean:

$$\mathbb{E}\{\mathbf{W}_l\} = 0, \quad \text{and} \quad Var[\mathbf{W_l}] = \mathbb{E}\{\mathbf{W_l^2}\} \tag{1.21d}$$

We can then write:

$$Var[y_l] = n_l \mathbb{E}\{\mathbf{W}_l \mathbf{x}_l\} \tag{1.21e}$$

Where n_l denotes the number of connections in the layer l

Because $\mathbf{x}_l = max(0, \mathbf{y}_{l-1})$, the expectation $\mathbb{E}\{\mathbf{x}_l\}$ is not equal to zero making $\mathbb{E}\left[x_l^2\right] \neq \mathrm{Var}\left[x_l\right]$. The expectation of $\mathbf{x}^2$ is then defined as follow in terms of integrals:

$$\mathbb{E}\{\mathbf{x}_l^2\} = \int_{-\infty}^{+\infty} \max(0, \mathbf{y}_l)^2 p(y_l) dy_l \tag{1.21f}$$

The negative part of y does not contribute to the integral:

$$\mathbb{E}\{\mathbf{x}_l^2\} = \int_0^{+\infty} \mathbf{y}_l^2 p(y_l) dy_l \tag{1.21g}$$

$\mathbf{y}^2$ is symmetric around 0, and $p(y)$ is assumed to be symmetric around 0 :

$$\mathbb{E}\{\mathbf{x_l}^2\} = \frac{1}{2} \int_{-\infty}^{+\infty} \mathbf{y}_l^2 p(y_l) dy_l \tag{1.21h}$$

If we let w_{l-1} having a symmetric distribution around zero and $b_{l-1} = 0$, then y_{l-1} has a symmetric distribution around zero which implies $\mathbb{E}\{\mathbf{y}_{l-1}\} = 0$, so we can write:

$$E\left[y_l\right] = E\left[w_l x_l\right] = E\left[x_l\right] E\left[w_l\right] = 0 \tag{1.21i}$$

$$\mathbb{E}\{\mathbf{x}_l^2\} = \frac{1}{2} \int_{-\infty}^{+\infty} (\mathbf{y}_l - \mathbb{E}\{\mathbf{y}_l\})^2 p(y_l) dy_l = \frac{1}{2}(y_l - \mathbb{E}\{\mathbf{y}_l\})^2 \tag{1.21j}$$

$$= \frac{1}{2} Var[\mathbf{y}_l] \tag{1.21k}$$

From Eq.(1.21k), Eq.(1.21e) becomes:

$$Var[\mathbf{y}_l] = \frac{n_l}{2} Var[\mathbf{W}_l] Var[\mathbf{y}_{l-1}] \tag{1.21l}$$

By Combining layer 1 to layer L:

$$\operatorname{Var}\left(\mathbf{y}_L\right) = Var\left(\mathbf{y}_1\right) \left(\prod_{l=2}^{L} \frac{n_l}{2} Var\left(\mathbf{W}_l\right) \right) \tag{1.21m}$$

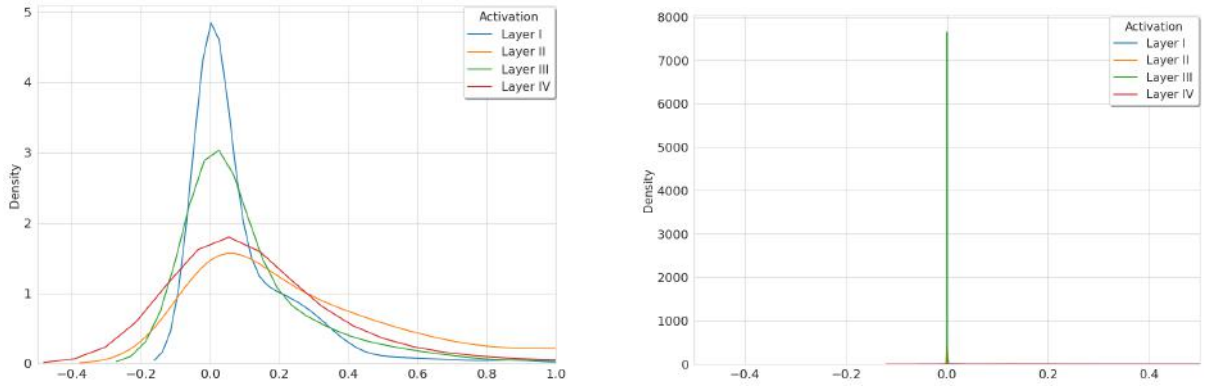
To prevent exploding or vanishing gradients problem, we expect variance at the input to be equal to the variance at the output. It will happen only if each term inside the product $= 1$. ie:

$$\frac{n_l}{2} \operatorname{Var}\left(\mathbf{W}_l\right) = 1, \quad \forall l. \tag{1.21n}$$

Hence, we reach the previously mentioned formula, ie:

$$\mathbf{W} \sim \mathcal{N}\left(0, \frac{2}{n_l}\right) \tag{1.21o}$$

This leads to a zero-mean Gaussian distribution whose standard deviation σ^2 is $\frac{\sqrt{2}}{n_l}$ and b is initialized with 0.



(a) With He Initialization **(b)** Without Initialization

Figure 1.22: Activation value of each layer

From Figure 1.22, we can clearly see that an uninitialized model suffers from gradient vanishing and saturation, preventing the model from learning salient and diverse features for each layer. On the other hand, by limiting the variance of weights distribution to 1, *He* Initialization allows the model to learn noteworthy features across its layers by computing something interesting, and prevent from excessive saturation of activation functions on one hand by letting gradients propagate well, and avoid overly linear units.

Restricted Boltzmann Machine

Restricted Boltzmann Machine are artificial neural network that can learn a probability distribution over its set of inputs. This architecture was introduced in [38] by Geoffrey Hilton, and was used for:

- Dimensionality reduction.
- Classification.
- Topic modeling.

RBM are shallow, two-layer neural nets, the first layer of the architecture is called the visible/input layer, while the second layer is called the hidden layer like it is illustrated in Figure 1.23. Meanwhile, the hidden layer is responsible to reconstruct the input data as close as possible by tuning the connection weights and biases repeatedly.

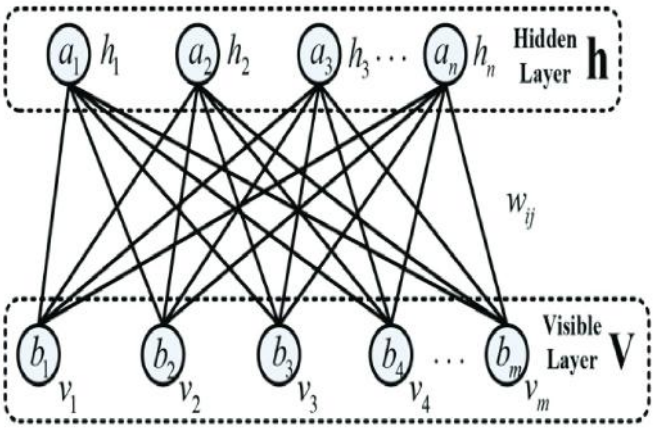


Figure 1.23: Basic Structure of a Restricted Boltzmann Machine

Each unit in layers are called neuron-like or node. The nodes are connected to each other across layers, but no two nodes of the same layer are linked, we can say that there is no intra-layer communication; the neurons from the same layer are disconnected. Each visible node takes a low-level feature from an item in the dataset to be learned. On the other hand, The hidden layer is responsible to reconstruct the input data as close as possible by tuning the connection weights and biases repeatedly (for example in an magnitude spectrum of 257 frequencies, we'll have 257 input nodes on the visible layer). At each node of the hidden layer, each v feed to the visible layer is multiplied by a dedicated weight, the products are summed, added to a bias, and activated. Each visible neuron represents a frequency feature with hypothetically Gaussian distribution. The energy function of joint configuration for the two layers is then defined as:

$$E(\boldsymbol{v}, \boldsymbol{h}) = -\sum_{i=1}^{m} b_i v_i - \sum_{j=1}^{n} a_j h_j - \sum_{i=1}^{m}\sum_{j=1}^{n} v_i h_j w_{ij} \tag{1.22}$$

where v_i and h_j are the binary states at the visible neuron i and hidden neuron j respectively. b_i and a_j are the corresponding biases of neurons, w_{ij} is the connection weight between them. Based on the Boltzmann distribution and energy function, a joint probability for pair of the visible and hidden layer is determined by:

$$p(\mathbf{v}, \mathbf{h}) = \frac{1}{Z} e^{-E(\mathbf{v},\mathbf{h})} \tag{1.23}$$

Where $Z = \sum_{v,h} e^{-E(v,h)}$ denotes the normalization term.
Considering that the hidden neurons are conditional independents due to no connections between them, given visible vector v, the conditional probability of neuron h_j being 1 can be obtained as follows:

$$p\left(h_j = 1 \mid \mathbf{v}\right) = \sigma\left(a_j + \sum_i v_i w_{ij}\right) \tag{1.24}$$

Similarly, given hidden vector h, the conditional probability of the visible neuron v_i being 1 can be determined by

$$p\left(v_i = 1 \mid \mathbf{h}\right) = \sigma\left(b_i + \sum_j h_j w_{ij}\right) \tag{1.25}$$

where $\sigma(\bullet)$ denotes the logistic sigmoid function.
Given the training dataset $S = \left\{s^1, s^2, \ldots, s^{n_s}\right\}$, where n_s is the number of training samples, the parameters of RBM are trained to fit the training samples by maximizing a log-likelihood function, including connection weights w, biases a and b.

Training of the RBM

RBM learns to reconstruct data by themselves in an unsupervised fashion, making several forward and backward passes between the visible layer and hidden layer. In the reconstruction phase, the activation of hidden layer becomes the input in the backward pass. The activation of hidden layer are multiplied by the same weights. The sum of those products is added to a visible-layer bias at each visible node, and the output of those operations is a reconstruction which approximates the original input. On the forward pass, an RBM uses inputs to make predictions about node activation, or the probability of a given input v : $p(h|v;w)$ but on its backward pass, RBM estimate the probability of input v given activation h, which are weighted with the same coefficients used in the forward pass : $p(v|h,w)$, where w represents the weights of the architecture.
The goal is then, to maximize the joint probability $p(x,h)$ in order to maximize the probability of event h occurring at the same time that event v occurs:

$$L_S = \sum_{i=1}^{n_s} \log p(\mathbf{v}, \mathbf{h}) \tag{1.26a}$$

Simply said, we can define the parameters necessary for the gradient computation:

$$h = \sigma(W^T v + b) \tag{1.26b}$$

$$\tilde{v} = \sigma(Wh + a) \tag{1.26c}$$

$$\tilde{h} = \sigma(W^T \tilde{v} + b) \tag{1.26d}$$

the update rule of the RBM weights and biases of the architecture are usually described as follow:

$$w_{i,j} \leftarrow w_{i,j} + \eta\left(\mathbf{v}\mathbf{h}^\top - \tilde{\mathbf{v}}\tilde{\mathbf{h}}^\top\right) \tag{1.26e}$$

$$b_i \leftarrow b_i + \eta\left(\mathbf{v} - \tilde{\mathbf{v}}\right) \tag{1.26f}$$

$$a_j \leftarrow a_j + \eta\left(\mathbf{h} - \tilde{\mathbf{h}}\right) \tag{1.26g}$$

1.5 Deep Unfolding

The past decade has witnessed an unprecedented success of deep learning in many real-world signal and image processing applications. Neural networks became the state of the art in pattern recognition, computer vision, and speech processing. Despite their proven success and high performances, deep learning algorithms suffer from the lack of interpretability provoked by their black-box nature. In contrast, another set of methods that have been tremendously successful in machine learning by incorporating problem domain-knowledge are the model-based methods, but, at the expense of complexity.

Deep unfolding or algorithm unrolling is an emerging technique that was introduced in 2014 [39]. This direction knew increasing popularity in the last few years and it combines the advantages of both deep learning and model-based methods. In this section, we explain the idea and motivation behind deep unfolding, its advantages, and domain applications.

1.5.1 Unfolding principles

Deep unfolding is a paradigm that fuses a model-based method that relies on an iterative inference algorithm and deep learning tools to solve a wide range of tasks in signal and image processing with both high performances and better interpretability. In authors words [39], deep unfolding can be defined as follows :

" [...] given a model-based approach that requires an iterative inference method, we unfold the iterations into a layer-wise structure analogous to a neural network. "

This means that for an iterative algorithm that has a fixed number of iterations K, the algorithm will be unrolled into a deep network with K layers and the parameters of the model at each iteration will be considered as the parameters of that layer.

Algorithm 2: Iterative algorithm example.

Input: Data sample $\mathbf{x}$.
Input: Initial parameters Φ_0.
for $i = 1{:}K$ **do**
  Update $\Phi^{(i)}$ using $\Phi^{(i-1)}$ and $\mathbf{x}$.
end

This iterative algorithm can be unrolled into a neural network by considering the parameters $\Phi^{(i)}$ as the activation of each layer k and convert some of the parameters of the model Θ into trainable parameters that are trained using deep learning techniques with a suitable objective function and an optimizer (e.g Stochastic gradient descent) to solve the optimization problem.

Note: The parameters $\{\Theta^{(1)}, ..., \Theta^{(K)}\}$ are present in the original iterative algorithm as a single vector of parameters and they are duplicated in the unfolded network into K vector of param-

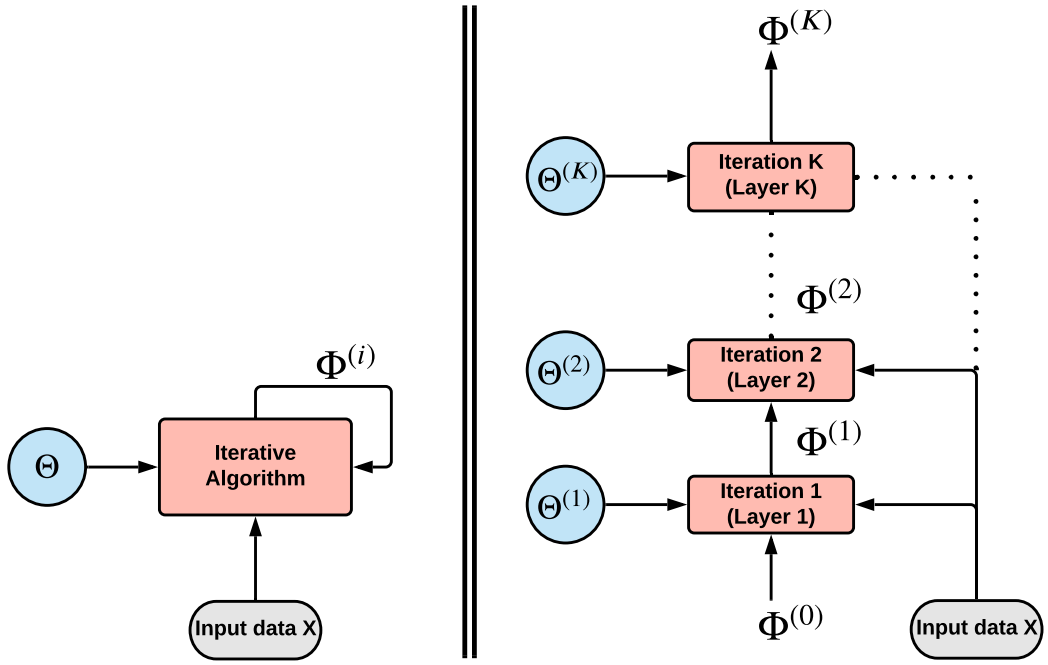


Figure 1.24: Unfolding an iterative algorithm into a K layers deep network.

eters. For example, the Gaussian mixture model is a probabilistic model based method that assumes the data is generated from a mixture of a finite number of gaussians with unknown mean μ, covariance matrix Σ and weights of each Gaussian π. The estimation of these parameters is done using an iterative algorithm called the expectation maximization algorithm (EM). The iterations of EM can be unrolled and the learnable parameters $\Theta^{(i)}$ are $\{\mu^{(1)}_{n=1:N}, ..., \mu^{(K)}_{n=1:N}, \Sigma^{(1)}_{n=1:N}, ..., \Sigma^{(K)}_{n=1:N}\}$ where N is the number of gaussian mixtures and the activations of the unfolded network Φ are the mixture weights of each iteration for each component n $\{\pi^{(1)}_{n=1:N}, ..., \pi^{(K)}_{n=1:N}\}$.

Another model-based method that was unfolded in [39] and [6] is the non-negative matrix factorization (NMF). The original NMF method is covered in in Chapter 3 and its unfolded version in Chapter 5 of our report.

Finally, we summarize the steps of unfolding the inference algorithm of a model-based method in the following :

1. Define a model (e.g. Gaussian Mixture Model) and its parameters ($\mu_{n=1:N}$,$\Sigma_{n=1:N}$,$\pi_{n=1:N}$, N is the number of mixtures).

2. Derive an iterative inference algorithm (e.g Expectation maximization algorithm) with a fixed number of iterations K.

3. Unroll the iterations of the iterative algorithm into K layer network.
4. Define the parameters of the model that should be trained with backpropagation and duplicate them across layers.
5. Choose an appropriate loss function and optimizer.
6. Train the unrolled network like a neural network.

1.5.2 Advantages of Unrolling Algorithms

The motivation behind model-driven neural networks is to fuse principled algorithms that have prior knowledge of the problem using handcrafted features (model based methods) with the techniques used in deep learning. The purpose of this paradigm is to combine the best of both worlds while avoiding their drawbacks.

Deep unfolding is a successful paradigm of model-driven neural networks that has the following advantages :

- Incorporating problem domain knowledge present in the logical and hand-crafted analytical expressions of the models into the neural networks design, instead of focusing on intensive learning from the data. Consequently, unrolled algorithms have better generalization performances because they already have prior information about the problem.
- The learnable parameters of the unfolded network corresponds to the model parameters and regularization coefficients which makes it easier to interpret the network and overcome the "Black-Box" problem that conventional neural networks suffer from. Besides, a forward pass through the network is nothing but running the iterative algorithm for a fixed number of times.
- Classical signal and image processing algorithms have in general significantly fewer parameters than neural networks (NNs). Hence, their unfolded version is less complex than NNs and requires less training data.
- Most iterative algorithms in signal and image processing have already an optimized hardware implementation making it easy to derive an implementation for their unfolded network. Moreover, these algorithms have low memory requirements compared to NNs. Deep unfolding takes advantage of this which makes this paradigm the best candidate for edge devices.
- Iterative methods suffer often from being too slow for a large number of applications. Unfolding the iterations of these algorithms makes them much faster to be considered for real-time applications.

Chapter 2

Materials and Methods

2.1 Dataset

The data we used is a collection of speech and music audio. The speech signal was issued from a business conversation between men and women, without background noise or music. On the other hand, the music audio provides piano pieces of multiple types (classical, pop ..) of multiples artists.
In fact, having male as well as female speeches make the audio signal much more complex, because of the variety of tones. Indeed, women speak at a higher pitch; about an octave (an octave is simply a measure of distance between one note and another note that is double its frequency) higher than men, while men's voices are generally deeper. In addition, we considered that having different types of songs from different artists would give us a robust algorithm by its ability of generalization across multiples pianists, whatever their musical genre. This choice also made the problem solving much more complex, but made it with more interesting and flexible perspectives.
Figure 2.1 above, describes how we split the data between the training and validation set, as well as the duration of each data used at each stage.

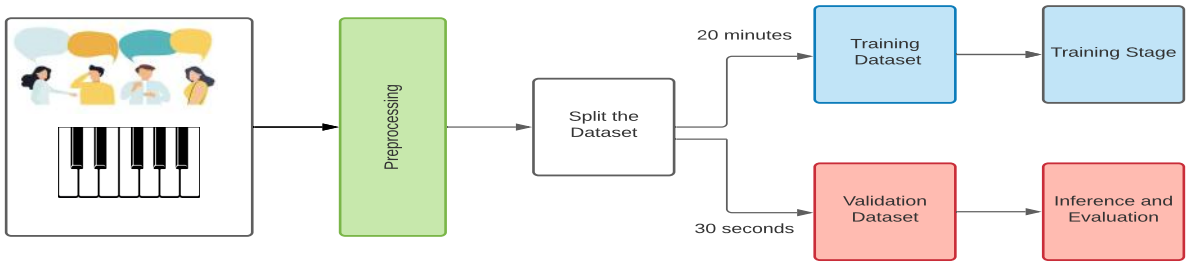


Figure 2.1: Data preparation and preprocessing

2.2 Software tools

In this section, we describe the environment we use to develop our solutions among model-based as well as the data-based method.

2.2.1 Programming language

For the implementation of the code solution, we use Python v3.7. In addition to be a high-level, interpreted, and object-oriented language, Python is a general-purpose coding language, its main strength lies in its readability, making the implementation session really easy, allowing students and researchers to focus much more on the problem and its solution than the language syntax

and memory management.

2.2.2 Libraries and Frameworks

Python is suitable for data science and Machine Learning. In 2018, 66% of data scientists reported using Python daily, making it the number one language for analytics professionals. In fact, Python has a great support and an extensive selection of libraries and frameworks for visualization, mathematical computing, and AI's modeling, we can cite:

NumPy [40] NumPy is one of the most powerful Python libraries. The library is open-source and mainly used for array computing, allowing to perform a number of mathematical operations on arrays such as trigonometric, statistical, and algebraic routines. Furthermore, because of its C implementation, NumPy arrays allow computing operations extremely fast, while keeping the syntax clear ,simple to write, and to read.

SciPy [41] SciPy is an open source Python-based library, dedicated to scientific computing, and mathematics. The basic data structure used by SciPy is NumPy arrays. We personally use this library for its signal processing support, such as frequency re-sampling, filtering and time-frequency representations such as *Short Time Fourier Transform* (STFT).

Matplotlib [42] Matplotlib is also an open-source Python library. It is the reference library for visualization, it provides many different kinds of 2D and 3D plots that are very useful for data analysis and machine learning tasks. Matplotlib offers flexibility and allows users to design complex and explicit figures. All the plots in this project have been created using this library.

Scikit-learn [43] Sklearn is the reference tool for machine learning in Python. It is built on the library NumPy for high-performance linear algebra and array operations. This library is used to build machine learning models related to clustering, decomposition, and classification. We personally used this library to use the NMF to separate our mixture signal. It should be noted that Scikit-learn should not be used for reading the data, manipulating and summarizing it, as there is better tool to do that like NumPy and Pandas.

Pytorch [44] The deep learning framework that was used to build models is Pytorch V1.9.0. Pytorch is a Python package based on Torch, which is an open-source machine learning package based on the programming language Lua. PyTorch has three main features:

- Tensor computation (like NumPy) with strong GPU acceleration, and an ease to use.
- Automatic differentiation for building and training neural networks.

- Build dynamic graph.

As a reminder, Python is convenient because it is easy to read and understand. PyTorch emphasizes flexibility and allows deep learning models to be expressed in idiomatic Python. Plus, Pytorch is written in such an intuitive way that you can learn in seconds. In other words, think about Numpy, but with strong GPU acceleration. In fact, most researchers in Machine Learning and Deep Learning prefer using Pytorch instead of Tensorflow, because it allows quick and efficient implementation of the solution proposed, allowing researchers to focus much more on the problem and its solution than in its software development. Moreover, PyTorch offers modularity, which enhances the ability to debug or see within the network, by supporting dynamic computation graphs that allow changing how the network behaves on the fly.

2.2.3 Google Colaboratory

Google Colaboratory is a free cloud service that offers Jupyter Notebooks via remote servers. This solution allows to write and run Python code in the browser. It offers three main advantages:

- No configuration required
- Access to GPUs and TPUs.
- Free.

In order to be precise, Google Colaboratory is a *Platform as a Service* (PAAS), that serves as a platform for the development of the code to create cloud-based applications, allowing customers to provision, instantiate, run, a computing platform and one or more applications, without the complexity of building and maintaining the infrastructure typically associated with developing and launching the application(s), allowing developers to create, develop, and package without worrying about the environment or the hardware.
Personally, We use this service because Google's Colaboratory (Colab) provides an enhanced Jupyter notebook that saves the work automatically in Google Drive. Next, because we wanted to completely focus on the problem solving. Moreover, we went for "Colab" because it allowed us to work on the same file, making it much easier to collaborate. Finally, whenever we had to upload heavy libraries and files (audio files), the process was very quick because input/output operations are very fast thanks to the *PAAS* type of Colab.

2.3 Data Preprocessing

Data preprocessing, broadly speaking, is anything did to data prior to inputting it into specific machine learning model.

2.3.1 Normalization

In order to get all the data on the same scale, allowing the model to implicitly consider all features equally in their representations, we choose to transform variables by scaling each of them to a range of 0 to 1, usually known as min-max scaler.

$$x_{f,t} = \frac{x_{f,t} - x_j^{min}}{x_f^{max} - x_f^{min}}, \tag{2.1}$$

$x_{f,t}$: the scaled input point at the timestamp t of the f^{th} frequency.
x_f^{max} : the maximum value of the feature column x_f.
x_f^{min} : the minimum value of the feature column x_f.
One of the main reasons of choosing this particular scaler is that it guarantees that scaled value will always be non negative, which respects the non-negativity constraint of the NMF.

2.3.2 Sampling

Shannon NyQuist Sampling Theorem

Suppose you have a signal that is oscillating representing continuous data, the Shannon Nyquist Sampling Theorem answer the famous question:

> "How fast have we to measure that signal to perfectly represent and reconstruct it, in order word: to go from the continuous domain to the discrete domain without losing information."

The theorem states that if you have a function and you want to ideally represent that function you want to perfectly resolve all of its frequency content, meaning that you have to sample that function at twice its highest frequency.

$$f_{max} < \frac{f_s}{2} \tag{2.2}$$

Where f_{max} represents the highest frequency of the signal which can be represented by the bandwidth B, and f_s is the sample rate frequency.
If the condition of Eq. 2.2 is not respected, aliasing phenomena occurs, consequently, it will affect the signal reconstruction.

Aliasing Phenomena

To get an insight of this phenomena, we illustrate it on Figure 2.2 above.

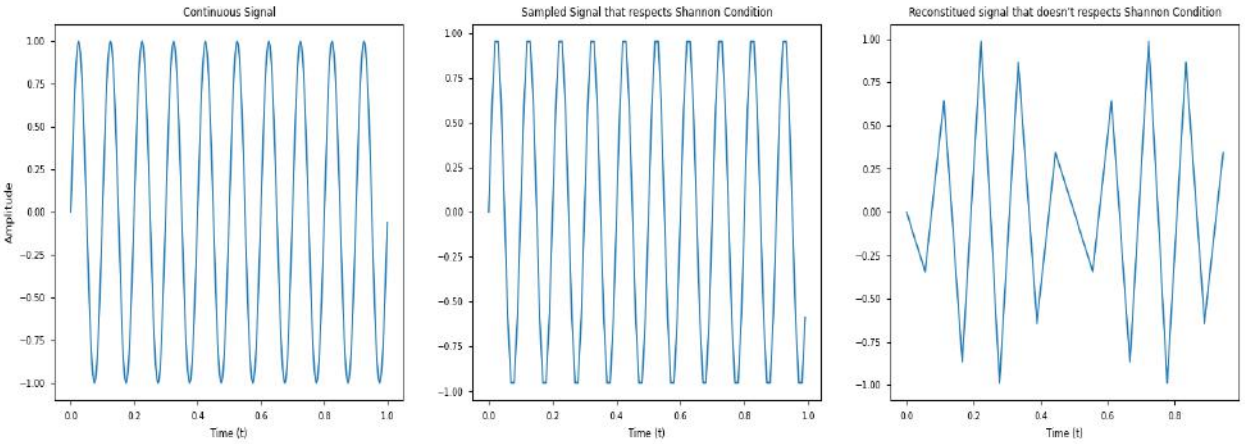


Figure 2.2: Effect of sampling on the signal

We can also study the Fourier transform of the sampled discrete signal, and compare it to the Fourier Transform of the continuous signal.
In order to get the discrete signal, we multiply the continuous signal by a Dirac comb $s(t)$:

$$x_s(t) = x_a(t) \cdot s(t) = x_a(t) \cdot \sum_n \delta(t - nT) \tag{2.3a}$$

Since multiplication in the time domain is equivalent to convolution in the frequency domain, we get:

$$X_s(j\Omega) = \frac{1}{2\pi} X_a(j\Omega) * S(j\Omega) \tag{2.3b}$$

Where Ω is the highest frequency component of $x_a(t)$, knowing that the TF of $s(t)$, is a periodic impulse in the frequency domain:

$$S(j\Omega) = \frac{2\pi}{T} \sum_k \delta(\Omega - \frac{k2\pi}{T}) = \frac{2\pi}{T} \sum_k \delta(\Omega - k\Omega_s) \tag{2.3c}$$

We ends up with:

$$X_s(j\Omega) = \frac{1}{T} \sum_k X_a(j\Omega - kj\Omega_s) \tag{2.3d}$$

From Eq. 2.3d, the TF of the sampled signal is a superposition of infinitely many shifted copies of the TF of the analog signal $X_a(j\Omega)$ resulting in repeated copies of this latter in integer multiple of Ω_s, in both direction of the frequency axis. From this conclusion, we can say that if the sampling rate is not greater than twice the Nyquist Frequency, the copies, when superimposed, will overlap and higher frequencies will be aliased into the lower frequency range; consequently,

we wouldn't be able to reconstruct the original signal from its samples by the use of a filter.

The wav audio we used is encoded at $f_s = 44Khz$ that is because humans can hear up to about $f_{max} = 20Khz$, so by taking the highest frequency and double that, we get perfect fidelity reconstruction, that is why audio signals are sampled at $f_s = 44Khz$.

Down-Sampling

Down-sampling allows to make a digital audio signal smaller by lowering its sampling rate or sample size (bits per sample). Down-sampling is used to decrease the bit rate when transmitting over a limited bandwidth or to convert to a limited audio format which is the main cause of our use of down-sampling. Furthermore, we used that technique in order to compare our methods to what it was already done in the research field, where authors generally down-sampled their audio signals to $f = 16Khz$ to allow the model to extract general features from the signal in order to make patterns much more obvious.
The figure 2.3 illustrates the process of down-sampling.

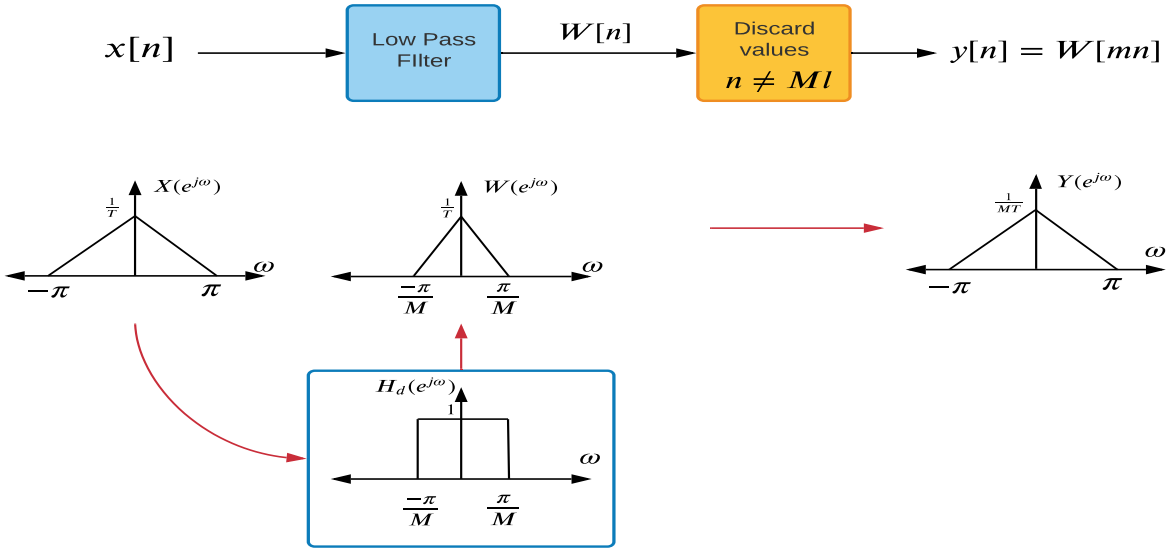


Figure 2.3: Down-Sampling Steps

- First, when a digital signal is downsampled, it is necessary to apply a low pass filter to the signal to reduce the signal's bandwidth to less than the Nyquist rate of the new sample rate; otherwise, aliasing will result, in other words, the low pass filter will help to get rid of high frequency components that could alias.
- The discard block is an operation that will throw away all the values except integer multiple

of M, that produces the desired signal, it is like scaling the time axis by a factor of M.

In the frequency domain, the Short Fourier Transform (SFT) of the signal, goes from $-\pi$ to π, with an amplitude of $\frac{1}{T}$. If we don't use an anti-aliasing filter, frequency higher than the frequency desired would alias back into the band of interest. Furthermore, the discarded block when throwing away all the values that are not a multiple of M, will stretch the frequency band of the signal, while compressing the signal in the time domain. Instead of ending the signal at $\frac{\pi}{M}$, it will extend all the way out to π, this property can be illustrated in the form of an equation.

$$\text{if } y[n] = W[mn] \text{ , then } Y\left(e^{j\omega}\right) = \frac{1}{M}\sum_{m=0}^{M-1} W\left(e^{j(\omega - m2\pi)/M}\right) \tag{2.4}$$

2.3.3 Filtering

Filters are one of the most essential operation used in signal processing. A filter in our case, is a process that removes unwanted components or features from a transmitted signal. Most often, this means removing some frequencies and not others to suppress interfering signals and reduce background noise. Ideally, a filter will not add new frequencies to the input signal, nor will it changes the component frequencies of that signal, but it will change the relative amplitudes of the various frequency components and/or their phase relationships.

Butterworth Filter

Butterworth Filter is a low pass filter, which is defined by an amplitude response with the equation below:

$$|H_n(j\omega)| = \frac{1}{\sqrt{1 + \left(\frac{\omega}{\omega_c}\right)^{2n}}} \tag{2.5}$$

Where ω_c represents the cut off frequency of the filter, and the factor quantity $\frac{\omega}{\omega_x}$ is raised to the exponent $2n$, where n is called the filter order. If you have a high filter order, it become more like a step function, while having a low filter order, this rolls off kind of smoothly as frequency goes up, as it is shown in Figure 2.4:

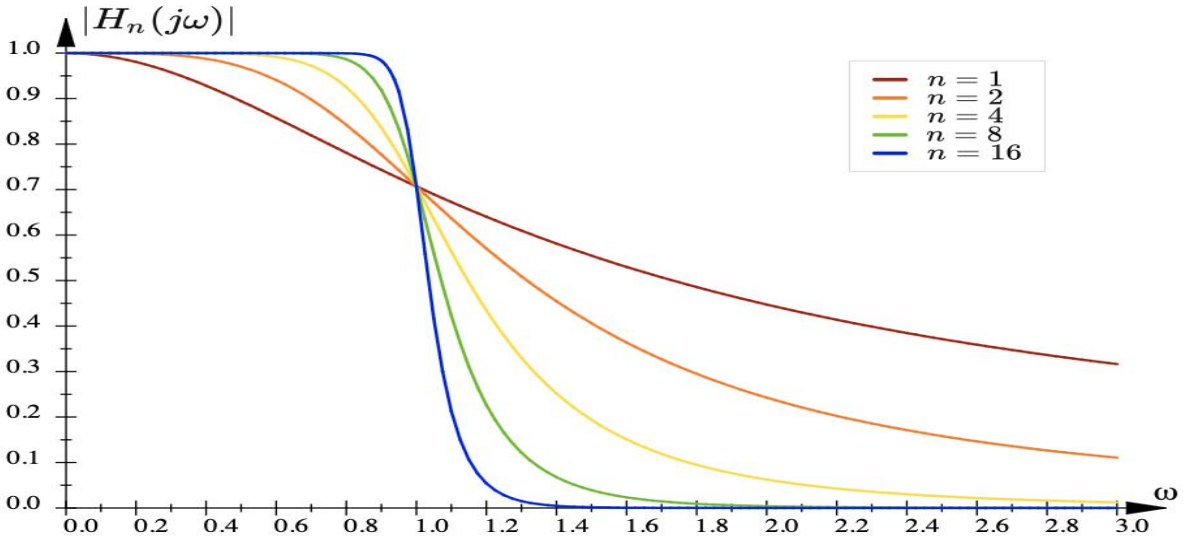


Figure 2.4: Butterworth filter Amplitude Response

We apply the Butterworth Filter after the down-sampling in order to apply a denoising process to the signal a second time. We set the cut off frequency at $f_c = 5Khz$, because we notice that frequencies equal or highest to f_c represented less than 1% of frequencies presented in the magnitude spectra of each source, this filtering in Data Science is a feature engineering that aims to get rid of outliers because removing high-frequency noise allows the signal of interest to be more compactly represented and enables more accurate analysis.

2.4 Performance Criteria

Measuring the results of a source separation approach is a challenging problem. Generally, there are two main categories for evaluating the outputs of a source separation approach:

- **Objective**: objective measures rate separation quality by performing a set of calculations that compare the output signals of a separation system to the ground truth isolated sources.
- **Subjective**: subjective measures involve having humans to give scores for the source separation system's output.

Objective and subjective measures both have benefits and drawbacks. Objective measures struggle because there are many aspects of human perception that are extremely difficult to capture by computational means alone. However, compared to subjective measures, they are much faster and cheaper to obtain. On the other hand, subjective measures are expensive, time-consuming,

and subject to the variability of human raters, but they can be more reliable than objective measures because actual human listeners are involved in the evaluation process. In our study, we choose the two approaches, with a focused on the objective measures using the Signal to Distortion Ratio (SDR).
An estimate of a Source $\hat{s}_i$ is assumed to actually be composed of four separate components expressed by Equation 2.6.

$$\hat{s}_i = s_{\text{target}} + e_{\text{interf}} + e_{\text{noise}} + e_{\text{artif}} \tag{2.6}$$

where s_{target} is the original source, and e_{interf} , e_{noise} , and e_{artif} are error terms for interference, noise, and added artifacts, respectively. The actual calculations of these terms is quite complex, so we refer the reader to the original paper for their exact calculation [45].

2.4.1 Signal-to-Distortion Ratio

The SDR is a widely used method for evaluating a source separation system's output, and usually considered to be an overall measure of how good a source sounds, the equation of this measures is illustrated by Equation 2.7.

$$\text{SDR} = 10\log_{10}\left(\frac{\|s_{\text{target}}\|^2}{\|e_{\text{interf}} + e_{\text{noise}} + e_{\text{artif}}\|^2}\right) \tag{2.7}$$

SDR was shown to be valid as a global performance measure in case we want to consider interference, noise, and artifact at the same time, which is equivalent to give importance to the Signal-to-Interference ratio (SIR), the Signal-to-Noise Ratio (SNR), as well as the Signal-to-Artifacts Ratio (SAR) at the same time.

Chapter 3

Non Negative Matrix Factorization

3.1 Non Negative Matrix Factorization

Non negative Matrix Factorization is a Constrained Low-Rank Matrix Approximation algorithm, that is used to decompose any non-negative matrix $\mathbf{X} \in D^{F \times N}$ into a non-negative basis vectors matrix $\mathbf{W} \in D^{F \times K}$ and a non-negative gain matrix $\mathbf{H} \in D^{K \times N}$

$$\mathbf{X} \approx \tilde{\mathbf{X}} = \mathbf{WH} \tag{3.1}$$

F: the total number of frequency of the spectrum.
N: the total number of sample.
The Non Negative Matrix Factorization is subject to the constraint that the approximating matrix has a reduced rank, so that it enables to extract pertinent information from large data sets such as:

$$\text{rank}(\tilde{\mathbf{X}}) \leq \text{rank}(\mathbf{X}) \tag{3.2}$$

In most cases, as cited before, k is usually chosen such that $k \ll \min(m, n)$ to respect the low ranked approximation aspect of the algorithm that aims to obtain a pair factor $(\mathbf{W}, \mathbf{H})$ that can be seen as a compressed form of the data in $\mathbf{X}$. Another key characteristic of NMF is that it doesn't allow negative entries in $\mathbf{W}$ and $\mathbf{H}$, to enable a non-subtractive combination of parts to form a whole. Therefore, NMF faces some important challenges affecting the numerical minimization of the problem, which includes the existence of local minimas due to the non-convexity of $f(\mathbf{W}, \mathbf{H})$, where f is the Frobenius Norm (see subsection 3.1.2), because the Hessian of the function is not positive semidefinite for $\mathbf{W}$, $\mathbf{H}$, and $\mathbf{X}$, resulting in the lack of a unique solution which can be easily seen by considering $\mathbf{WDD}^{-1}\mathbf{H}$ where $\mathbf{D}$ is an arbitrary diagonal matrix, for any nonnegative invertible matrix.

3.1.1 Beta-Divergence Loss

In this section, we discuss the objective function used for finding $\mathbf{W}$ and $\mathbf{H}$ as an approximation for the matrix $\mathbf{X}$. Density Power Divergence is indexed by a single parameter β which controls the trade-off between robustness and efficiency.

The β-Divergence was introduced by [46] who proposed a similarity loss for robust parameter estimation. Also known as the contrast function, the β-Divergence is a function that establishes the "distance" of one probability distribution to one another.

$$D_\beta(x \mid y) = \begin{cases} \frac{1}{\beta(\beta-1)} \left(x^\beta + (\beta-1)y^\beta - \beta x y^{\beta-1} \right) & \beta \in \mathbb{R} \backslash \{0, 1\} \\ x \log \frac{x}{y} - x + y & \beta = 1 \\ \frac{x}{y} - \log \frac{x}{y} - 1 & \beta = 0 \end{cases} \tag{3.3}$$

Reference [46] assumes $\beta \geq 1$, but the definition domain can be extended to $\beta \in \mathcal{N}$, as suggested

by [47].The limit cases $\beta = 0$ $\beta = 1$, correspond to the Itakura-Saito (IS) and Kullback-Leibler (KL) divergences, respectively.

Kullback-Leibler Divergence

The KL divergence tells us how well the probability distribution $\mathbf{Q}$ approximates the probability distribution $\mathbf{P}$ defined on the same probability space $\mathcal{X}$ by calculating the cross-entropy minus the entropy.

$$D_{\mathrm{KL}}(\mathbf{P}\|\mathbf{Q}) = \sum_{x\in\mathcal{X}} \mathbf{P}(x) \log\left(\frac{\mathbf{P}(x)}{\mathbf{Q}(x)}\right) \tag{3.4}$$

In other words, it is the expectation of the logarithmic difference between the probabilities $\mathbf{P}$ and $\mathbf{Q}$, where the expectation is taken using the probabilities $\mathbf{P}$. Intuitively, you can think of that as the statistical measure of how one distribution differs from another.

Itakura Saito Divergence

In Non-negative matrix factorization, the Itakura-Saito divergence can be used as a measure of the quality of the factorization; this implies a meaningful statistical model of the components and can be solved through an iterative method. Because it belongs to the β-Divergence family, the IS divergence measures the difference between an original spectrum $\mathbf{X}$ and an approximation $\tilde{\mathbf{X}}$ of that spectrum that represents the reconstruction of the spectrum using NMF.

$$D_{IS}(\boldsymbol{X}\|\tilde{\mathbf{X}}) = \sum_{f,n}\left(\frac{\boldsymbol{X}_{f,n}}{\tilde{\mathbf{X}}_{f,n}} - \log\frac{\boldsymbol{X}_{f,n}}{\tilde{\mathbf{X}}_{f,n}} - 1\right) \tag{3.5}$$

One of the unique properties of IS divergence is that it is scale invariant, meaning that low energy components of X bear the same relative importance as high energy ones. This is relevant to situations in which the coefficients of $\mathbf{X}$ have a large dynamic range.

$$D_{IS}(\lambda\boldsymbol{X}|\lambda\tilde{\mathbf{X}}) = D_{IS}(\boldsymbol{X}\|\tilde{\mathbf{X}}) \tag{3.6}$$

The scale invariance of the IS divergence is relevant to the decomposition of audio spectrums, which typically exhibit exponential power decrease along with frequency f and also usually comprise low-power components such as note attacks performed on piano for example, together with higher-power components such as tonal parts of sustained notes, making IS-divergence suitable for speech-music separation.

3.1.2 Frobenius Norm

The Frobenius norm, also called the Euclidean norm is the matrix norm of an $m \times n$ matrix X defined as the square root of the sum of the absolute squares of its elements, also equals to the square root of the matrix trace.

$$\|\mathbf{E}\|_F \equiv \|\mathbf{X} - \mathbf{WH}\|_F \equiv \sqrt{\sum_{i=1}^{m}\sum_{j=1}^{n}\left|e_{ij}\right|^2} \equiv \sqrt{\mathrm{Tr}\left(\mathbf{E}\mathbf{E}^{\mathrm{H}}\right)}.$$

Where $\mathbf{E}^{\mathbf{H}}$ is the conjugate transpose of $\mathbf{E}$. Then, Non-Negative Matrix Factorization (NMF) aims to factorize the non-negative $n \times m$ matrix X into two non-negative factor matrices $W(n \times k)$ and $H(k \times m)$ such that :

$$\min_{\mathbf{W},\mathbf{H}} \|\mathbf{X} - \mathbf{WH}\|_F$$

3.1.3 Singular Value Decomposition

Singular Value Decomposition is a data reduction algorithm that helps reduce the data into keys features to analyze and understand this data. The algorithm is based on a simple and interpretable linear algebra, and it is also scalable; meaning that it can be applied in small as well as large and massive datasets. Consider X a collection of column vectors:

$$X = \begin{bmatrix} \vdots & \vdots & & \vdots \\ x_1 & x_2 & \cdots & x_m \\ \vdots & \vdots & & \vdots \end{bmatrix} \tag{3.7}$$

SVD allows to take the matrix X and decompose it or represents it as the product of three other matrices:

$$X = \begin{bmatrix} \vdots & \vdots & & \vdots \\ u_1 & u_2 & \cdots & u_n \\ \vdots & \vdots & & \vdots \\ \vdots & \vdots & & \vdots \end{bmatrix} \begin{bmatrix} \sigma_1 & & & \\ & \sigma_2 & & \\ & & \ddots & \\ & & & \sigma_m \\ \hline & & & \\ & & 0 & \end{bmatrix} \begin{bmatrix} \vdots & \vdots & & \vdots \\ v_1 & v_2 & \cdots & v_m \\ \vdots & \vdots & & \vdots \end{bmatrix}^{\top} = U\Sigma V^{\top} \tag{3.8}$$

$$(n \times m) = \quad (n \times n) \quad (n \times m) \quad (m \times m) \quad = (m \times m)$$

U and V matrix are called unitary / orthogonal matrix, and Σ diagonal matrix. With $X \in \mathbb{R}^{n \times m}$ the matrix we want to approximate, $U \in \mathbb{R}^{n \times n}$ the left-singular matrix of X, $\Sigma \in \mathbb{R}^{n \times m}$ the matrix of singular value, and $V \in \mathbb{R}^{m \times m}$ the right-singular vectors of X . Columns of U are

hierarchically arranged so that u_i is somehow more important than u_{i+1} and so on and so forth in terms of their ability to describe the variance in the columns of X. Its also important to note that U and V are unitary, which means that:

$$U^T.U = V.V^T = I, \tag{3.9a}$$

$$V^T.V = V.V^T = I, \tag{3.9b}$$

U columns are orthonormals means that they are orthogonals and unit length providing a complete basis for all of our n dimensional vector space from the columns of the data. The same is for V.

$$|| \sum_{j=0}^{m-1} u_{i,j}|| = 1 \tag{3.10a}$$

$$u_i \cdot u_k = 0, \quad \text{with} \quad i \neq k \tag{3.10b}$$

Σ is a diagonal matrix, non negative and hierarchically ordered, which means diagonal values are ordered in a decreasing magnitude such as $\sigma_1 > \sigma_2 > \cdots > \sigma_m$, this ordering carries with it the ordering of the columns U and V as well.In summary, The relative importance of columns of U and V are given by the corresponding singular value. For example, if some σ are small, we're going to be able to ignore them to chop them off and approximate the matrix X only in terms of the first few dominant columns in U and columns of V and the dominant singular values σ. SVD Decomposition is guaranted to exist, and its unique.Furthermore, we can say that U contains information about the column space of X and V contains information about the row space of X, and Σ is a diagonal matrix that tells you how important the various columns of U and V are, hierarchically arranged.
In term of matrix approximation, there are only m linearly independent columns in X, meaning that there is only m columns of U that are important in representing the data. The expansion of the equation 3.8 is:

$$X = \sigma_1 u_1 \odot v_1^\top + \cdots + \sigma_m u_m \odot v_m^\top \tag{3.11}$$

$\odot$: outter product.
u_m : column of the matrix U.
v_m^T : line of the matrix $V^\top$.

In summary, we decompose X into orthogonal basis U and V by writing this as a sum of rank-one matrices that increasingly improve the approximation of X.In addition, even if U has n columns, there are only m non-singular values.So we select only the first m columns of U, the $m \times m$ block in Σ, this is often called the **Economy SVD**, assuming $n >> m$, meaning having many more entries in each column than having columns.
Its also worth to add, that the low rank approximation problem is solved by **SVD**. The Eckart–Young–Mirsky

theorem states that the best approximation for X of rank m is given by the first r truncated singular value of the **SVD** decomposition such as $r < m$.

$$\text{argmin}||X - \tilde{X}|| = \tilde{U}\tilde{\Sigma}\tilde{V}^{\top} \text{ such as } rank(\tilde{X}) = r \text{ and } r < m \tag{3.12}$$

3.1.4 Nonnegative Double Singular Value Decomposition

NNDSVD is a method that is used to initialize the nonnegative matrix factorization, that contains no randomization which enhances the interpretability and the control over the initialization stage. Due to the iterative nature of NMF algorithms, the initialization of the basis vectors and the masks W, H respectively is an important component in the convergence of the optimization algorithm. Its also important to state that traditional NMF uses random nonnegative initialization for its pair factor W, H; which requires several instances of the algorithm using different random initializations to select the best solution. It is also important to note that NMF suffers from slow convergence that can make it quite expensive. NNDSVD is based on two SVD processes, containing no randomization making the algorithm converge to the same solution every time. Before going any further, we should underline what authors mean by calling NNDSVD a good initialization strategy, they mean one that leads to rapid error reduction and faster convergence, and one that leads to better overall error at convergence. Because NMF is a constrained low-rank matrix approximation as cited in [48], the initialization strategy needs a low-rank factorization scheme. NNDSVD depends on two properties concerning the behavior of unit rank matrices. NNDSVD start from the basic property of the truncated matrix with rank r, that has the optimal rank r approximation of $\mathbf{X}$ with respect to the Frobenius norm:

$$X = \sum_{j=1}^{r} \sigma_j u_j v_j', \tag{3.13}$$

where $\sigma_1 \cdots \sigma_r > 0$ are the nonzero singular values of X and u and v are the corresponding left and right singular vectors respectively. Every pair factor σ_j, u_j, v_j that constitues a unit rank matrix $C^{(j)}$ is approximated by its nonnegative part $C_+^{(j)}$. We show below the Lemma that is essential for the elaboration of the solution:

Lemma 3.1.1 *Lemma 1. Consider any matrix* $C \in \mathbb{R}^{m \times n}$ *such that* $\text{rank}(C) = 1$, *and write* $C = C_+ - C_-$. *Then* $\text{rank}\,(C_+)$, *rank* $(C_-) \leq 2$.

The result tells us that if we zero out all negative values of a unit rank matrix, the resulting matrix will have rank 2 at most, making the approximation of C^j possible with its nonnegative section C_+^j. Furthermore, each matrix C^j can be written as the sum of two non-negative components:

$$C^{(j)} = C_+^{(j)} - C_-^{(j)}$$

$C_+^{(j)}$, $C_-^{(j)}$ being the positive and negative part of $C^{(j)}$ respectively.
Plus, the Perron-Frobenius theory that states that the leading eigenvalue is positive, guarantee that the pair (W, H) is initialized using the non-negative left and right singular vectors corresponding to the maximum singular value of the decomposition, meaning that the dominant singular triplets u_1, σ_1, v_1 can be used as initial vectors and rows to (W, H).
In summary, the main phases of this algorithm are as follow:

Step 1: Compute the largest r singular triplets of X (first SVD process).

Step 2: Initialize the first column and row vectors in W and H as the nonnegative dominant singular vectors of X weighted by $\sqrt{\sigma_i}$

Step 3: Compute the positive section of each $C^{(i)}$ using the "set to zero with small rank increment" property.

Step 4: Compute the largest r singular triplets of $C_+^{(j)}$(second SVD process).

Step 5: Initialize the k^{th} columns and rows in W and H, for $k = 2, \cdots, r$ as the singular dominant vectors of each $C_+^{(j)}$, weighted by the $\sqrt{\sigma_i\left(C_+^{(j)}\right)}$ and normalized.

Finally, the process of the $NNDSVD$ can be processed as below:

Algorithm 3: NNDSVD Algorithm steps

Data:

Input: magnitude spectrum $X \in \mathcal{R}_{+}^{m \times n}$, number of components k of the NMF, such as $k < min(m, n)$

Output: Rank-k nonnegative doublets $W \in \mathcal{R}_{+}^{m \times k}$, and $H \in \mathcal{R}_{+}^{k \times n}$

Compute the largest k singular triplets of X using Singular Value Decomposition:

$[U,\Sigma,V] = SVD(X,k) \approx \sum_{i=0}^{k-1} \sigma_i C_i \qquad \text{with } C_i = u_i v_i^{\top}$

Initialize the first column and row vectors in W and H as the nonnegative dominant singular vectors of X weighted by $\sqrt{\sigma_i}$:

$W[:,0] = \sqrt{\sigma_1} \cdot U[:,0]$

$H[0,:] = \sqrt{\sigma_1} \cdot V[:,0]^{\top}$

Compute the positive section of each C_i:

$C_i^{+} = \text{ReLU}(C_i, 0) = \sum_{i=0}^{k-1} u_i^{+} v_i^{+} + u_i^{-} v_i^{-}$

Compute the largest k singular triplets of C_i^{+} with a second Singular Value Decomposition Process:

$[U^{'},,V^{'}] = SVD(\sum_{0}^{k-1} C_i^{+}, k)$

Initialize the j^{th} columns and rows in W and H, for $j = 1, \ldots, k-1$ as the singular dominant vectors knowing the *Perron–Frobenius* theory, of each C_i^{+}, weighted by the $\sqrt{\sigma_i}$ and normalized.

Loop: *for i from 1 to $k-1$:*

W[:,i] $\leftarrow$ *normalized*$(\sqrt{\sigma_i} U_i^{'}[:,0])$,

H[i,:] $\leftarrow$ *normalized*$(\sqrt{\sigma_i} V_i^{'\top}[:,0])$,

3.1.5 Multiplicative-Update Algorithm

To solve the equation 3.1 , the solution to find **W** and **H** require the minimization of the Itakura-Saito (IS) Divergence cost function 3.1.1 that we used for our solution.

$$\min_{\boldsymbol{W},\boldsymbol{H}} D_{IS}(\boldsymbol{V} \| \boldsymbol{W}\boldsymbol{H}) \tag{3.14}$$

This update approach exploits the fact that multiplying any two non-negative values produces another non-negative value. This implies that if an element of either factor is assigned the value zero, during the update it remains at zero. One interesting characteristics of this technique is

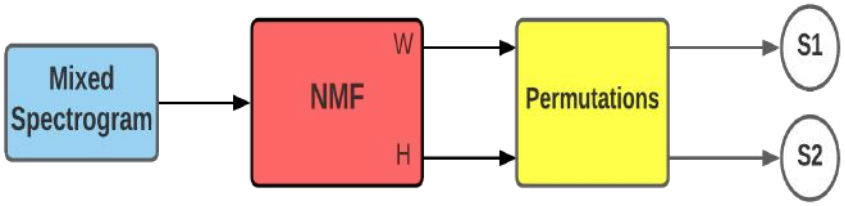


Figure 3.1: Pipeline of learning free NMF.

that it can be extended to a large number of cost functions.

$$H \leftarrow \mathbf{H} \odot \frac{\mathbf{W}^T \left(\frac{\mathbf{V}}{(\mathbf{WH})^{\beta+2}} \right)}{\mathbf{W}^T \left(\frac{\mathbf{1}}{(\mathbf{WH})^{\beta+1}} \right)}$$
$$W \leftarrow \mathbf{W} \odot \frac{\left(\frac{\mathbf{V}}{(\mathbf{WH})^{\beta+2}} \right) \mathbf{H}^T}{\left(\frac{1}{(\mathbf{WH})^{\beta+1}} \right) \mathbf{H}^T} \tag{3.15}$$

Where $\mathbf{1}$ is a matrix of ones with the same size of $\mathbf{V}$, the operation $\odot$ is an element-wise multiplication, and $(.)^2$ are element-wise operations.The matrices $\mathbf{W}$ and $\mathbf{H}$ are initialized by positive random numbers and then updated iteratively using 3.15. This algorithm includes a normalization step at every iteration, which eliminates trivial scale indeterminacies, leaving the cost function unchanged

3.2 Learning Free and Supervised NMF

3.2.1 Learning Free NMF

Learning Free NMF is used to decompose the mixture signal without any prior knowledge about the sources that constitute the mixture. This latter is separated into sources without any training stage. In this context, both of the basis vectors $\mathbf{W}$ and $\mathbf{H}$ are estimated from a mixture spectrogram. The drawback of this type of NMF is that it restricts the model to use only n components for n sources,where each component will correspond to the basis vector of each sources. Furthermore, this operation will require performing permutation in order to select the appropriate basis vector for each source.

3.2.2 Supervised Non Negative Matrix Factorization

In supervised learning, a model aims to learn a function that maps an input to an output. In our application, training non negative matrix factorization for each isolated signal aims to find pre-trained basis vectors that represents each respective source.These methods typically estimate a separate set of basis vectors to represent each source. The basis vectors of all the sources are then concatenated, and used to represent the mixture signal, we say then that we have prior knowledge on the indexes of the components that represent a given source. The advantage of trained NMF is that we can use as much as component as possible to modelize each source, making the basis vectors much more complex, enabling them to express much more salient information about its appropriate source, without the need of doing permutation operation

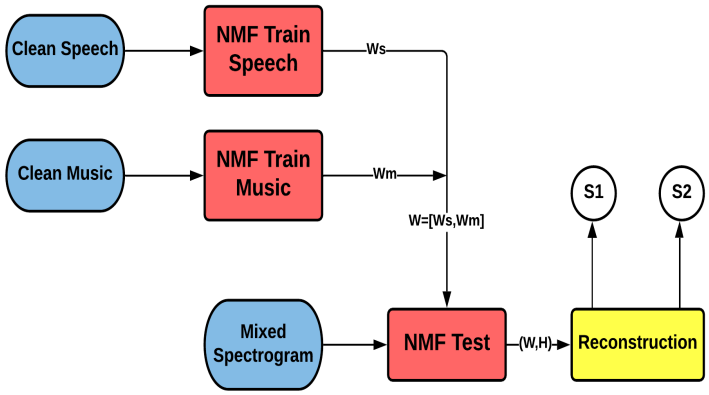


Figure 3.2: Pipeline of NMF with pretrained weights.

Trained Dictionnaries

Given a set of training data for speech and music signals, the STFT is computed for each source, and the magnitude spectrogram **S** and **M** of speech and music signals are calculated, respectively. Mathematically, we can express the process as follow: Take the isolated speech signal for example, for which we want to represent its magnitude spectrum approximately as a set of basis vectors W_s called speech dictionary and a mask matrix H_s formulated by the Constrained Low Rank Matrix such as (same for piano signal) :

$$\boldsymbol{S}_{train} \approx \boldsymbol{W}_s \boldsymbol{H}_s \tag{3.16a}$$

$$\boldsymbol{M}_{train} \approx \boldsymbol{W}_m \boldsymbol{H}_m \tag{3.16b}$$

Basis vectors $\boldsymbol{W}$ and masks $\boldsymbol{H}$ of Eq. 3.16 are calculated using Multiplicative Update cited on 3.15 with respect to the objective function used in this context. In our case we used Itakura Saito

(IS) expressed by Eq.(3.5).
In other word, the aim of using pretrained NMF is to model the training data as a set of basis vectors to represents the spectral characteristics for each source signal, resulting in a dictionary expressing both characteristic of speech and music spectrum expressed by Eq.(3.17)

$$\boldsymbol{W} = [\boldsymbol{W}_s \;\; \boldsymbol{W}_m] \tag{3.17}$$

Decomposition of the mixed signal

In the parameter estimation, the pretrained dictionaries $\boldsymbol{B}$ are kept fixed, and only their gains $\boldsymbol{H}$ are estimated. Once the masks have been estimated, source can be separated from the mixture by applying a mask, as explained in subsection 3.2.4. In the test stage, the estimation of gains is done using the pretrained dictionaries found during the training of each basis vectors on each speech, and music spectrum, respectively. Those latter, are freezed, aiming to extract appropriate gains $\boldsymbol{H}^{'}$ initialized with $NNDSVD$ cited in 3.1.4, from the mixture with the prior knowledge of the spectral characteristics of each source expressed by its basis vectors $\boldsymbol{W}$ (see figure 3.3).

$$H^{'} \leftarrow \mathbf{H}^{'} \odot \frac{\mathbf{W}^T \left(\frac{\mathbf{V}}{(\mathbf{W}\mathbf{H}^{'})^2} \right)}{\mathbf{W}^T \left(\frac{\mathbf{1}}{(\mathbf{W}\mathbf{H}^{'})^1} \right)} \tag{3.18}$$

with $\beta = 0$ because we're minimizing the Itakura-Saito Divergence loss. Finally, the estimated spectrogram of the speech signal is found by multiplying the bases matrix $\boldsymbol{W_s}$ with its corresponding masks matrix $\boldsymbol{H_s^{'}}$, the same process is done to estimate the spectrogram of the music signal.

$$\tilde{\boldsymbol{S}} = \boldsymbol{W}_{\text{speech}} \boldsymbol{H}_S^{'} \tag{3.19a}$$

$$\tilde{\boldsymbol{M}} = \boldsymbol{W}_{\text{music}} \boldsymbol{H}_M^{'} \tag{3.19b}$$

Finally, the algorithm behind the pretrained NMF is illustrated in algorithm 4 as follow:

Algorithm 4: Pretrained NMF steps.

Training:

Input: magnitude spectrum of clean speech and clean music signals $\boldsymbol{S}_{train}$, $\boldsymbol{M}_{train}$,

Output: Trained Dictionnaries $\boldsymbol{W}_s$, $\boldsymbol{W}_m$

Initialization: $\boldsymbol{W}_s, \boldsymbol{H}_s \leftarrow \boldsymbol{nndsvd}(\boldsymbol{S}_{train})$ | $\boldsymbol{W}_m, \boldsymbol{H}_m \leftarrow \boldsymbol{nndsvd}(\boldsymbol{M}_{train})$

Train: *for k in K iterations*:

$\mathrm{H}_s, W_s \leftarrow \boldsymbol{mu}(H_s, W_s, S_{train}, \beta)$,

$\mathrm{H}_m, W_m \leftarrow \boldsymbol{mu}(\mathrm{H}_m, W_m, M_{train}, \beta)$,

Decomposition of the mixed signal using trained dictionnaries:

Input: Pretrained Dictionnaries $\boldsymbol{W} = [W_s \quad W_m]$, Mixture $\boldsymbol{Y}_{mix}$.

Output: Gains of the mixture $\boldsymbol{H}_{mix}$, estimated source spectrum $\tilde{\boldsymbol{S}}$, $\tilde{\boldsymbol{M}}$ such as $\boldsymbol{Y}_{mix} \approx \tilde{\boldsymbol{S}} + \tilde{\boldsymbol{M}}$.

Initialization: $\boldsymbol{H}_m \leftarrow \boldsymbol{nndsvd}(\boldsymbol{Y}_{mix})$

Train: *for k in K iterations*:

$\mathrm{H}_{mix} \leftarrow \boldsymbol{mu}(H_{mix}, W, \boldsymbol{Y}_{mix}, \beta)$,

$\boldsymbol{W}$ *freezed*,

Estimation of source spectrum:

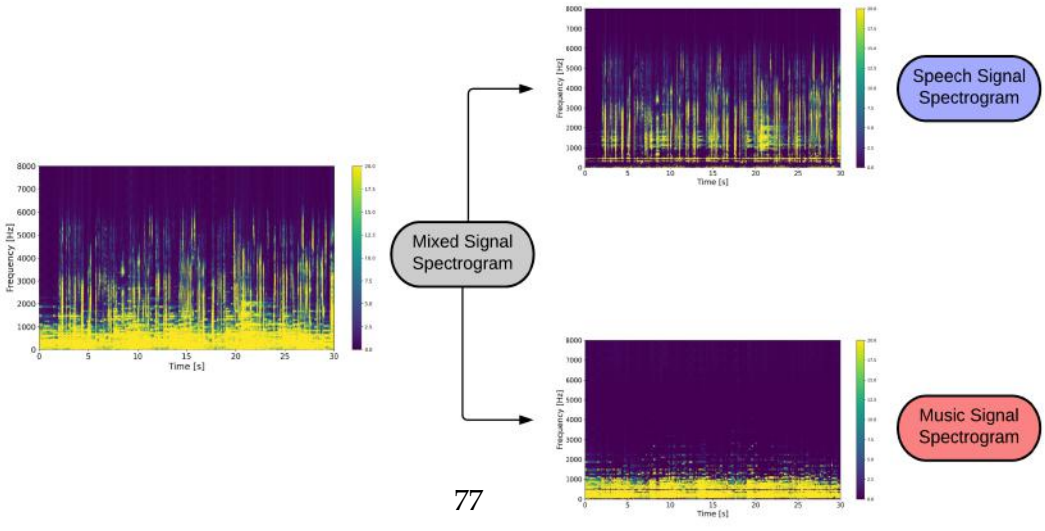
$$\boldsymbol{H}_{mix} = [\mathrm{H}_s^{'} \quad H_m^{'}]$$

$$\tilde{\boldsymbol{Y}} \approx [\boldsymbol{W}_s, \boldsymbol{W}_m]\, \boldsymbol{H}_{mix}$$

Source signals reconstruction and masks:

$$\tilde{\boldsymbol{S}} = \frac{(\boldsymbol{W}_s \boldsymbol{H}_s)^p}{(\boldsymbol{W}_s \boldsymbol{H}_s)^p + (\boldsymbol{W}_m \boldsymbol{H}_m)^p} \odot \boldsymbol{Y}$$

$$\tilde{\boldsymbol{M}} = \frac{(\boldsymbol{W}_m \boldsymbol{H}_m)^p}{(\boldsymbol{W}_s \boldsymbol{H}_s)^p + (\boldsymbol{W}_m \boldsymbol{H}_m)^p} \odot \boldsymbol{Y},$$



3.2.3 Component Effect

The best number of basis vectors depends on the application, the signal type and dimension. Hence, it is a design choice: Larger number of basis vectors may result in lower approximation error, but may result in overtraining and/or a redundant set of basis and require more computation time as well. Thus, there is a desirable number of bases to be chosen for each source.

Figure 3.4 illustrates Itakura-Saito loss values for each pair of speech and music components. The x and y axis represents the number of components used to train the speech and music dictionaries. It can be noticed that increasing the number of components has a good impact on the loss function, the minimum value of the loss is obtained when using the maximum number of components (64,64). Unfortunately, a good minimization of the loss function does not reflect a good separation. This conclusion comes from observing the heatmaps in Figure 3.5 of the SDR for the estimated speech and music signals for each pair of components. We notice that the optimal number of speech components is equal to 24, while the optimal number of music components depends on the SMR. For higher SMR values, the optimal number of the music components decreases because the power of the music becomes negligible compared to the speech power.

Note A high value of SDR in a heatmap corresponds to a light shade.

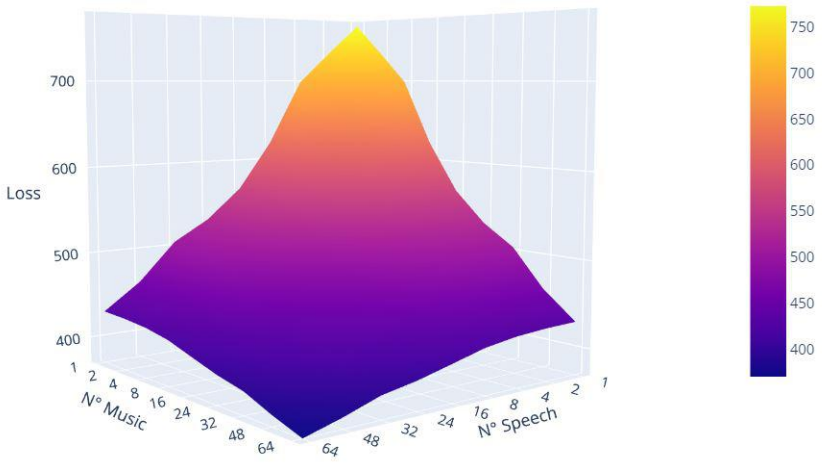


Figure 3.4: Itakura-Saito reconstruction loss for different numbers of NMF components.

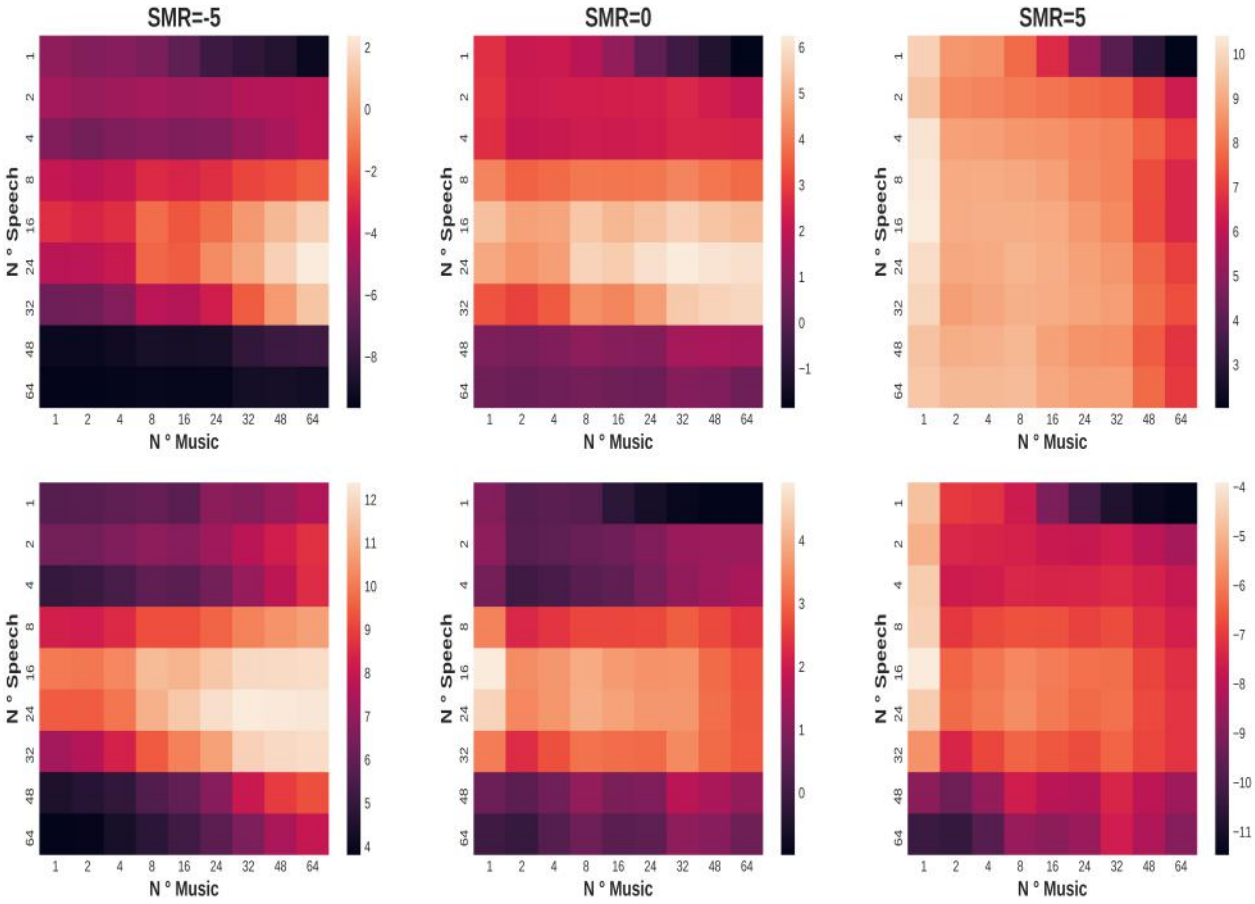


Figure 3.5: Heatmap of Speech SDR (up) and music SDR (bottom) evaluated for different numbers of speech and music components in the pretrained dictionary for values of SMR -5 0 and 5.

3.2.4 Masking

Masking, also known as filtering is an operation comes from a wide range of noise reduction techniques, performed in the time frequency domain by applying a mask (filter) on the mixed spectrogram to extract the different sources. To be more precise, masking is a multiplication operation of each time frequency bin by a gain (attenuation factor) in order to reduce the noise in speech enhancement or in other applications reduce the interference between signals.

Many research papers have been devoted to time frequency masking and they can be grouped into two main approaches : Heuristic and formal. In formal methods, the masks are designed by minimizing a cost function that in general has a trade off between noise reduction and speech distortions. In other words, these methods show some difficulties handling the preservation of speech harmonics and the noise removal. Among these methods : the Wiener filters [49] and Binary masks [50].

Wiener Filter

One way of finding the time frequency mask w is by minimizing the mean square error (MSE) between the target (clean signal) and its estimate $\hat{s}_1 = w \cdot Y$, with Y being the spectrogram of the mixed signal. The error can be formulated as follows :

$$\boxed{w_{opt} = \underset{w}{\operatorname{argmin}}\, \mathbb{E}\{\|s_1 - w^* \cdot Y\|^2\}} \tag{3.20}$$

w_{opt} : Optimal single channel Wiener filter.
s_1 : Clean signal.
Y : Mixed signal spectrogram.

$$\mathbb{E}\{\|s_1 - w^* \cdot Y\|^2\} = \mathbb{E}\{|s_1|^2\} + |w|^2 \cdot \mathbb{E}\{|Y|^2\} - 2 \cdot \Re[w^* \cdot \mathbb{E}\{s_1^* \cdot Y\}] \tag{3.21}$$

First, we compute the phase of w_{opt} and according to equation 3.21, the phase of w only affects the last term $-2 \cdot \Re[w^* \cdot \mathbb{E}\{s_1^* \cdot Y\}$. Minimizing the MSE error requires the maximization of the last term.

$$argmax[\Re[w^* \cdot \mathbb{E}\{s_1^* \cdot Y\}] \implies \boxed{\angle w_{opt} = \angle \mathbb{E}\{s_1^* \cdot Y\}} \tag{3.22}$$

Considering the expression of the phase of w, Eq.(3.21) becomes :

$$\mathbb{E}\{\|s_1 - w^* \cdot Y\|^2\} = \mathbb{E}\{|s_1|^2\} + |w|^2 \cdot \mathbb{E}\{|Y|^2\} - 2 \cdot |w| \cdot |\mathbb{E}\{s_1^* \cdot Y\}| \tag{3.23}$$

Finding the magnitude of w that minimizes the expression of MSE in equation (3.23) is done by computing the first derivative of the MSE and its zeros:

$$\frac{\partial \mathbb{E}\{\|s_1 - w^* \cdot Y\|^2\}}{\partial |w|} = 2 \cdot |w_{opt}| \cdot \mathbb{E}\{|Y|^2\} - 2 \cdot |\mathbb{E}\{s_1^* \cdot Y\}| = 0 \tag{3.24}$$

The solution of Eq.(3.24) is :

$$\boxed{|w_{opt}| = \frac{|\mathbb{E}\{s_1^* \cdot Y\}|}{\mathbb{E}\{|Y|^2\}}} \tag{3.25}$$

Combining the expression of the phase Eq.(3.22) and magnitude Eq.(3.25) we get :

$$\boxed{w_{opt} = \frac{\mathbb{E}\{s_1^* \cdot Y\}}{\mathbb{E}\{|Y|^2\}}} \tag{3.26}$$

In the case of two sources, we can write the mixed spectrogram as the sum of the spectrogram

of the sources :

$$Y = s_1 + s_2 \tag{3.27}$$

Under the assumption that the sources are uncorrelated, the expression of $\mathbb{E}\{s_1^* \cdot Y\}$ can be written as :

$$\mathbb{E}\{s_1^* \cdot s_2\} = 0 \tag{3.28a}$$

$$\mathbb{E}\{s_1^* \cdot Y\} = \mathbb{E}\{s_1^* \cdot s_1\} + \mathbb{E}\{s_1^* \cdot s_2\} = \mathbb{E}\{s_1^* \cdot s_1\} \tag{3.28b}$$

From the other side and under the same assumptions $\mathbb{E}\{|Y|^2\}$ becomes :

$$\mathbb{E}\{|Y|^2\} = \mathbb{E}\{|s_1|^2\} + \mathbb{E}\{|s_2|^2\} \tag{3.29}$$

The final expression of the optimal Wiener filter is :

$$\boxed{w_{opt} = \frac{\mathbb{E}\{|s_1^2|\}}{\mathbb{E}\{|s_1^2|\} + \mathbb{E}\{|s_2^2|\}}} \tag{3.30}$$

This expression can be generalized when the mixed spectrogram is a linear combination of sources with coefficients different than 1. For example : $Y = u \cdot s_1 + v \cdot s_2$. The optimal mask expression becomes :

$$\boxed{w_{opt} = \frac{u^2 \cdot \mathbb{E}\{|s_1^2|\}}{u^2 \cdot \mathbb{E}\{|s_1^2|\} + v^2 \cdot \mathbb{E}\{|s_2^2|\}}} \tag{3.31}$$

This filter was used in the reconstruction of the sources in this paper [51] and in our work.

Generalized Wiener Filter

The parametric Wiener filter is a generalization of the Wiener filter that was presented in [52]. It includes two parameters that are chosen empirically with no strong theoretical background about the setting of these parameters. An example of using it on "Image Power Spectrum Sparsity" was introduced in [53]. This filter was improved in [54] by proposing a new cost function that uses specific parameters to make a trade-off between the speech distortions and noise reduction. The expression can be written as follows :

$$\boxed{H_{mask}(t,f) = \frac{[P_s^p]_{t,f}}{[P_s^p + P_n^p]_{t,f}}} \tag{3.32}$$

P_s : Square of the magnitude spectrum as an estimate of the power spectral density.
P_n : Power spectral density of noise.

This mask is the optimal solution of the cost function that was introduced in [54]. It represents the error between the estimated signal spectrogram and the clean signal spectrogram.

$$\mathcal{E} = \hat{S} - S \tag{3.33a}$$

$$\hat{W} = H_{mask} \odot Y \tag{3.33b}$$

$$Y = S + N \tag{3.33c}$$

$$\mathcal{E} = H \odot Y - S = H \odot [S + N] - S \tag{3.33d}$$

$$\mathcal{E} = [H - \mathbb{1}] \odot S + H \odot N \tag{3.33e}$$

$\mathcal{E}_s = [H - \mathbb{1}] \odot S$: Speech distortion term.
$\mathcal{E}_n = H \odot N$: Noise distortion term.

In a giving time window, the power spectral densities of the speech and the noise can be written as :

$$d_s = \mathbb{E}\{\mathcal{E}_s^2\} = |H - \mathbb{1}|^2 \odot \sigma_s^2 \tag{3.34a}$$

$$d_n = \mathbb{E}\{\mathcal{E}_x^2\} = |H|^2 \odot \sigma_n^2 \tag{3.34b}$$

The cost function that the mask should optimize, is the sum of the power spectral densities of the speech and the noise distortions with the regularization parameter ρ.

$$\boxed{\underset{H}{\operatorname{argmin}}\, \mathbf{J} = d_s^{\alpha} + \rho \cdot d_n^{\alpha}} \tag{3.35}$$

ρ : Weight factor between speech distortion and reduction.
α : Cost function sharpness.

Both the expressions (Eq.3.34a) and (Eq.3.34b) are convex with respect to H, this implies that also the cost function 3.35 is convex which means only one H globally minimizes $\mathbf{J}$. Finding the mask that helps achieving this global minimum can be done by finding the zeros of the first derivative of the cost function :

$$\frac{\partial \mathbf{J}}{\partial H^*} = 0 \implies \frac{\partial [|H - \mathbb{1}|^{2\cdot\alpha} \cdot \sigma_s^{2\cdot\alpha} + \rho \cdot H^{2\cdot\alpha} \cdot \sigma_n^{2\cdot\alpha}]}{\partial H^*} = 0 \tag{3.36a}$$

$$\frac{\partial [[(H - \mathbb{1}) \cdot (H - \mathbb{1})^*]^{\alpha} \cdot \sigma_s^{2\cdot\alpha} + \rho \cdot (H \cdot H^*)^{\alpha} \cdot \sigma_n^{2\cdot\alpha}]}{\partial H^*} = 0 \tag{3.36b}$$

$$(1-\frac{1}{H})^{\alpha}\cdot(1-\frac{1}{H^*})^{\alpha-1}+\rho\cdot(\frac{\sigma_s}{\sigma_n})^{2\cdot\alpha}=0 \tag{3.36c}$$

Considering $\rho=\mu^{\alpha}$ $Z=1-\frac{1}{H}$ and $\eta=\frac{\sigma_s^2}{\sigma_n^2}$

$$Z^{\alpha}\cdot(Z^*)^{\alpha-1}=-(\frac{\mu}{\eta})^{\alpha} \tag{3.37}$$

$$|Z|^{\alpha}\cdot e^{j\cdot\alpha\cdot\phi_z}\cdot|Z|^{\alpha-1}\cdot e^{-j\cdot(\alpha-1)\cdot\phi_z}=-|\frac{\mu}{\eta}|^{\alpha} \tag{3.38}$$

$$|Z|^{2\cdot\alpha-1}\cdot e^{j\cdot\phi_z}=-|\frac{\mu}{\eta}|^{\alpha} \tag{3.39}$$

From (Eq.3.39), we get the phase and the magnitude of the optimal Z:

$$\boxed{e^{j\cdot\phi_z}=-1 \qquad |Z|=|\frac{\mu}{\eta}|^{\frac{\alpha}{2\cdot\alpha-1}}} \tag{3.40}$$

Now that we have the optimal solution Z, we can use the relationship between Z and H to find the expression of the optimal mask H_{opt} in terms of the cost function we have.

$$Z=1-\frac{1}{H} \implies H=\frac{1}{1-Z} \tag{3.41}$$

$$H_{opt}=\frac{1}{1+|\frac{\mu}{\eta}|^{\frac{\alpha}{2\cdot\alpha-1}}} \tag{3.42}$$

Finally, the expression of the optimal mask can be rearranged :

$$\boxed{p=\frac{\alpha}{2\cdot\alpha-1} \qquad H=\frac{\eta^p}{\eta^p+\mu^p}=\frac{\sigma_s^p}{\sigma_s^p+\mu^p\cdot\sigma_n^p}} \tag{3.43}$$

This method has two degrees of freedom, the parameter p and μ ($\mu=\rho^{\frac{1}{\alpha}}$).
Other filters derived from the Wiener Filter have been proposed in the literature such as the parameterized Wiener filter in [52].

$$\hat{S}=(\frac{\sigma_s^2}{\sigma_s^2+k\cdot\sigma_n^2})^{\beta}\cdot Y \tag{3.44}$$

β : Sharpness parameter.
k : Regularization parameter between noise reduction and speech distortions.
Y : Mixed spectrogram.

σ_s : Power spectral density of speech.
σ_n : Power spectral density of noise.

More details about other versions of the Wiener Filter can be found in [55], [54] and the spectral masking and filtering chapter in the book [56].

3.2.5 Sparse Non Negative Matrix Factorization

Non-negative Matrix Factorization computes the decomposition of a matrix to a low rank matrix subject to the constraint of non negativity. In contrast to PCA and ICA that do not restrict the signs of W and H, NMF requires all entries of both matrices to be non-negative. What this means is that the data is described by using additive components only. Furthermore, NMF is known for providing sparse representation of the data.

Sparse coding is a representation learning method that aims to encode sparsely the input data. In other words, this representation encodes much of the data using few 'active' components.

Even though NMF provides a sparse representation, the degree of sparsity remains uncontrolled because there is no parameter in the NMF algorithm that enables the control of this property. Many articles proposed solutions to control the sparsity of the basis vectors W, the gains H or both.

In [12] they introduced two parameters S_w and S_h that limits the sparseness of W and H respectively. The loss function 3.45 is minimized under the optional constraints of sparsity 3.46

$$E(W,H) = \|Y - W \cdot H\|^2 \tag{3.45}$$

$$\mathcal{S}(W) = S_W \tag{3.46a}$$

$$\mathcal{S}(H) = S_H \tag{3.46b}$$

Another approach for introducing sparsity into the model was proposed in [57]. In this paper, they have included the sparsity term in the cost function directly as follows:

$$\boxed{E(W,H) = \|X - W \cdot H\|^2 + \lambda \cdot \sum_{i,j} H_{i,j}} \tag{3.47}$$

The expression of the cost function 3.47 is the one we considered in our implementation of the Sparse Non-negative Matrix Factorization (SNMF).

Using the optimal number of components we found in section 3.2.3, we evaluate the impact of the sparsity parameter on the separation results.

3.3 Results and Discussion

Supervised NMF was trained on 20 min of isolated speech and music signals, and tested on 30 seconds of mixed signal, with 8 basis vectors for speech, and 16 basis vectors for music, making a trained dictionaries with a total of $r = 24$ components such as $\boldsymbol{B} \in \mathbb{R}^{F \times r}$.

Algorithm $(\mathcal{S} = 0)$	Input SMR (db)	SDR Speech (db)	SDR Music (db)
Free	-5	-4.0489	9.2173
Learning	0	**4.9065**	**3.3932**
NMF	5	9.1157	-21.215
Supervised	-5	-2.98	10.498
NMF	0	4.0755	3.11
(8,16)	5	8.44	-6.4225
Supervised (p=2)	-5	**2.3702**	**12.065**
NMF (p=1)	0	4.8368	2.6064
(p=0.1)	5	**10.795**	$-$**8.7938**

Table 3.1: Comparison between Free NMF and Supervised NMF in terms of SDR speech and music signal estimation.

The results presented in 3.1 show that Supervised NMF improves the mixture separation, meaning that the training phase of the dictionaries **W** allows the NMF to have prior information about the speech and the music features.

3.3.1 Sparsity

In this part, we evaluate the impact of the sparsity parameter λ on the quality of the reconstructed signal and the distortions. It can be noticed from Figure 3.6 and 3.7 that the optimal λ for SMR values -5, 0 and 5 is equal to $\lambda = 100$ for both speech and music.

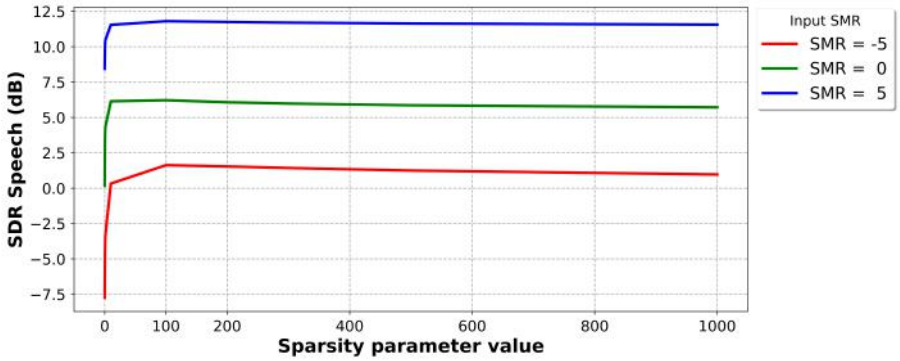


Figure 3.6: SDR of the estimated speech signal for different values of sparsity parameter λ.

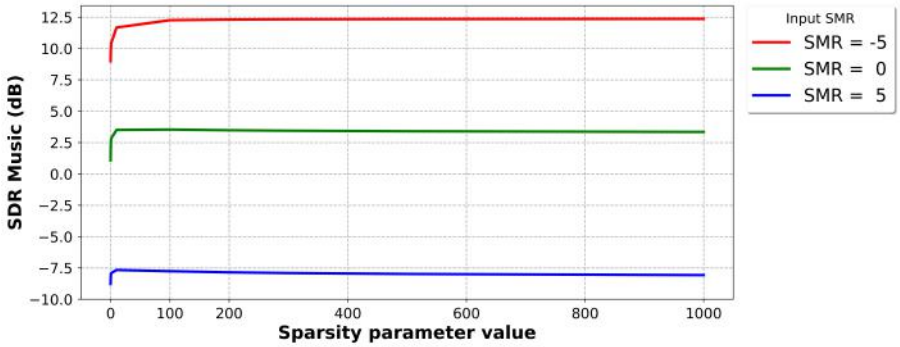


Figure 3.7: SDR of the estimated music signal for different values of sparsity parameter λ.

Table 3.2 illustrates the results of the NMF without sparsity compared to Sparse NMF with the optimal parameter λ_{opt}.

Algorithm	SMR Mixed (dB)	SDR Speech (dB)	SDR Music (dB)
NMF	-5	-7.74	8.99
	0	0.18	1.07
	5	8.44	-8.78
Sparse NMF	-5	1.63	12.25
	0	6.22	3.54
	5	11.80	-7.76

Table 3.2: Comparison between NMF and Sparse NMF in terms of SDR speech and music.

3.3.2 Masks

This section provides a study about the impact of the mask on the reconstructed signal distortions and the relationship between the mask and the SMR. In section 3.2.4, we have seen the following expression of the generalized Wiener Filter for each time frequency bin:

$$\boxed{H_{mask}(t,f) = \frac{[P_s^p]_{t,f}}{[P_s^p + P_n^p]_{t,f}}} \tag{3.48}$$

Where P_s is the power spectral density of the speech and P_m is the power spectral density of the music (interference). The choice of the parameter p is highly dependent on the SMR. As it can be seen in Figure 3.8 and 3.9, high values of p have a positive effect on the reconstruction of speech for zero and negative SMRs, while lower values exhibit good results for positive SMRs.

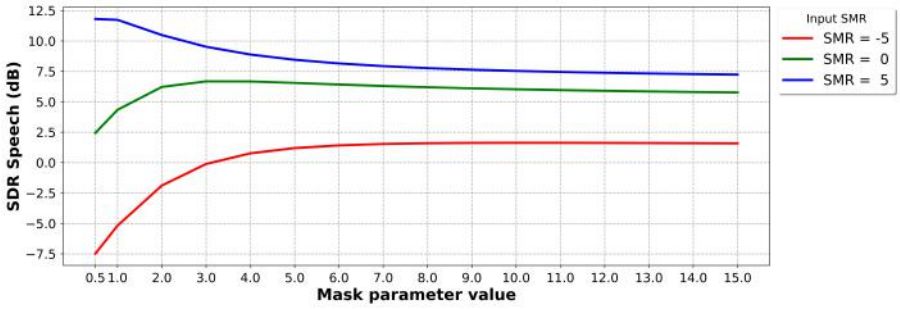


Figure 3.8: SDR of the reconstructed speech signal for different values of the parameter p.

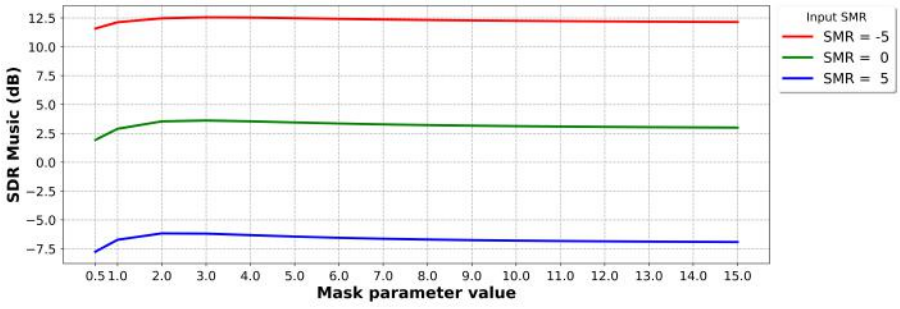
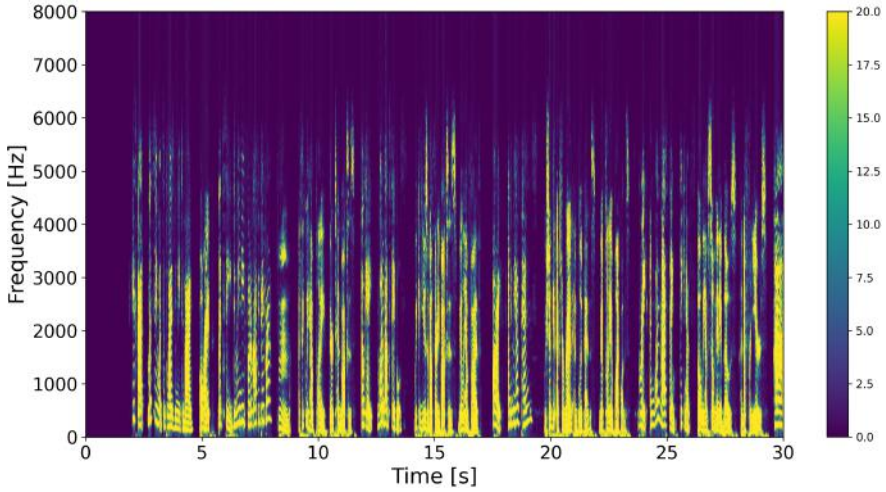
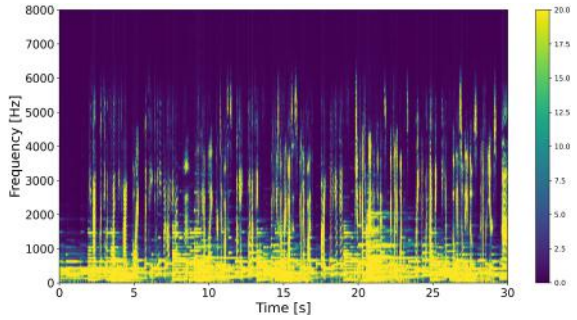


Figure 3.9: SDR of the reconstructed music signal for different values of the parameter p.

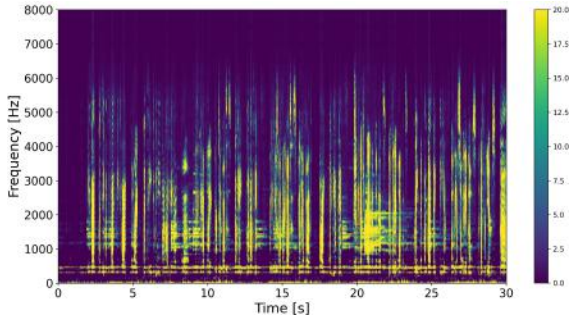
To better understand the effect of the mask parameter p on the estimated sources spectrograms, we visualize the heatmap of estimated speech spectrograms using $p = 0.5$ and $p = 10$.After applying NMF algorithm and masking, we obtain the following spectrogram of speech (see Figure 3.10).



(a) Clean Speech



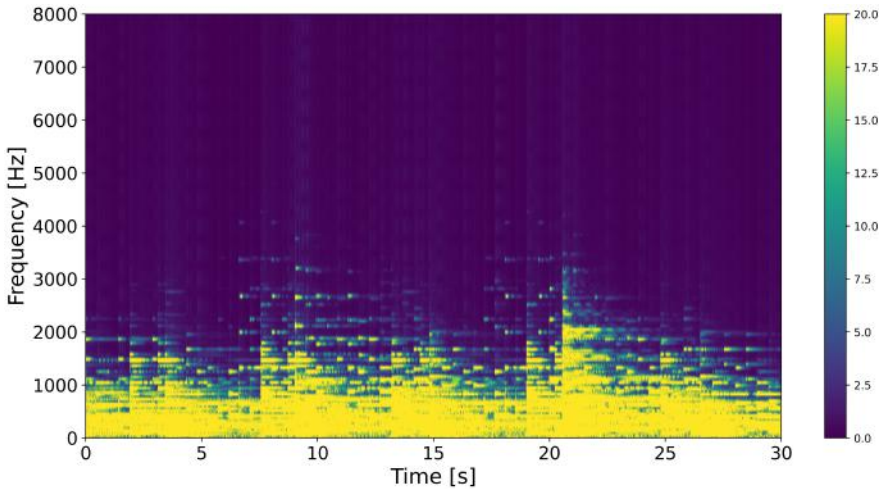
(b) p = 0.5



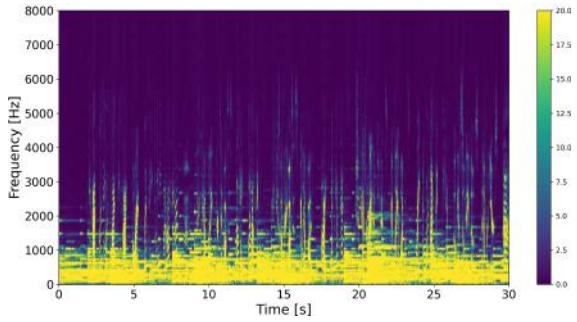
(c) p=10

Figure 3.10: Speech spectrograms for a mask with $p = 0.5$ and $p = 10$.

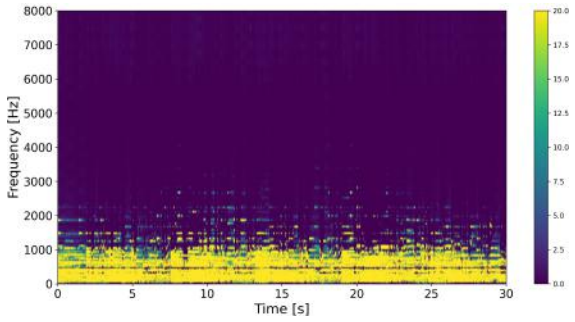
On the other hand, Figure 3.11 shows that for small values of p (e.g. 0.5) the mask did not eliminate well the time frequency bins that belong to the music signal from the estimated speech spectrogram. While a higher value of p (e.g 10) better filters the music from the speech. However, even if setting all the time frequency bins of the music to zero seems to be a good idea to reduce the interference of the music in the speech, it increases the distortions in the speech signal. These distortions can be observed in 3.10 (c) where some low frequencies are cancelled resulting in discontinuities.



(a) Clean Music magnitude spectrogram



(b) p = 0.5



(c) p=10

Figure 3.11: Music Spectrograms for a mask with $p = 0.5$ and $p = 10$.

Therefore, for SMR = 0 where the power of the speech is approximately in the same proportions as the power of music, the parameter p should have a values higher than 0.5 to filter better the time frequency bins that contain music (or speech in the case of the music spectrogram), and lower than 10 to avoid increasing distortions and having a mask that behaves like a binary filter. A parameter $p = 2$ (Wiener Filter) provides the best trade-off.

Mask parameter for other SMRs

The Wiener filter ($p = 2$) allows a good trade-off between the suppression of the interference and the reduction of the distortions. But when it comes to signals that have high difference in power (e.g. SMR=5), the optimal parameter p becomes different from 2.

To better explain this, let's consider a mixed signal with SMR = 5, and reformulate the ex-

pression of the speech mask (Eq.3.48) for each time frequency bin as follows :

$$H_{mask}(t,f) = \frac{1}{1+[(\frac{P_m}{P_s})^p]_{t,f}} \tag{3.49a}$$

‘

$$SMR = 5 \implies P_s \gg P_m \implies \frac{P_s}{P_m} \gg 1 \tag{3.49b}$$

- If $p > 1$:

  In time frequency bins (f, n) that contain speech :

$$p = 10 \wedge \frac{P_s}{P_m} \gg 1 \implies \frac{1}{1+(\frac{P_m}{P_s})^p} \approx 1 \tag{3.50}$$

  $\wedge$: logical AND operator.

$$H_{speech} \approx 1 \tag{3.51}$$

$$H_{music}(t,f) = \frac{1}{1+[(\frac{P_s}{P_m})^p]_{t,f}} \approx 0 \tag{3.52}$$

- If $p < 1$:
  In this case the filter behaves less like a binary filter, and the power gap between the two signals decreases. For example if the power of the speech is 10000 and the power of the music is 100, considering a filter with $p = 0.5$ will change the factor between the speech and the music in the mask $(\frac{P_s}{P_m})^p$ from 100 to 10 and this filter is far from being binary, therefore it reduces the distortions.

3.3.3 Effect of every loss on the SDR

In this subsection, we studied the impact of every loss we introduced in 3.1, to observe their impact on the speech and music separation.

Algorithm $(p=1)$ $(\mathcal{S}=0)$	Input SMR (db)	SDR Speech (db)	SDR Music (db)
KL Divergence	-5	-7.3575	10.141
	0	4.6571	2.9325
	5	5.8678	-7.4583
Itakura Saito	-5	-6.31	11.21
	0	3.06	2.85
	5	9.33	-6.86
Frobenius	-5	-10.279	8.6207
	0	2.4753	2.7561
	5	5.8678	-7.4583

Table 3.3: Loss effects on the estimation of speech and music signal for different SMRs in terms of SDR.

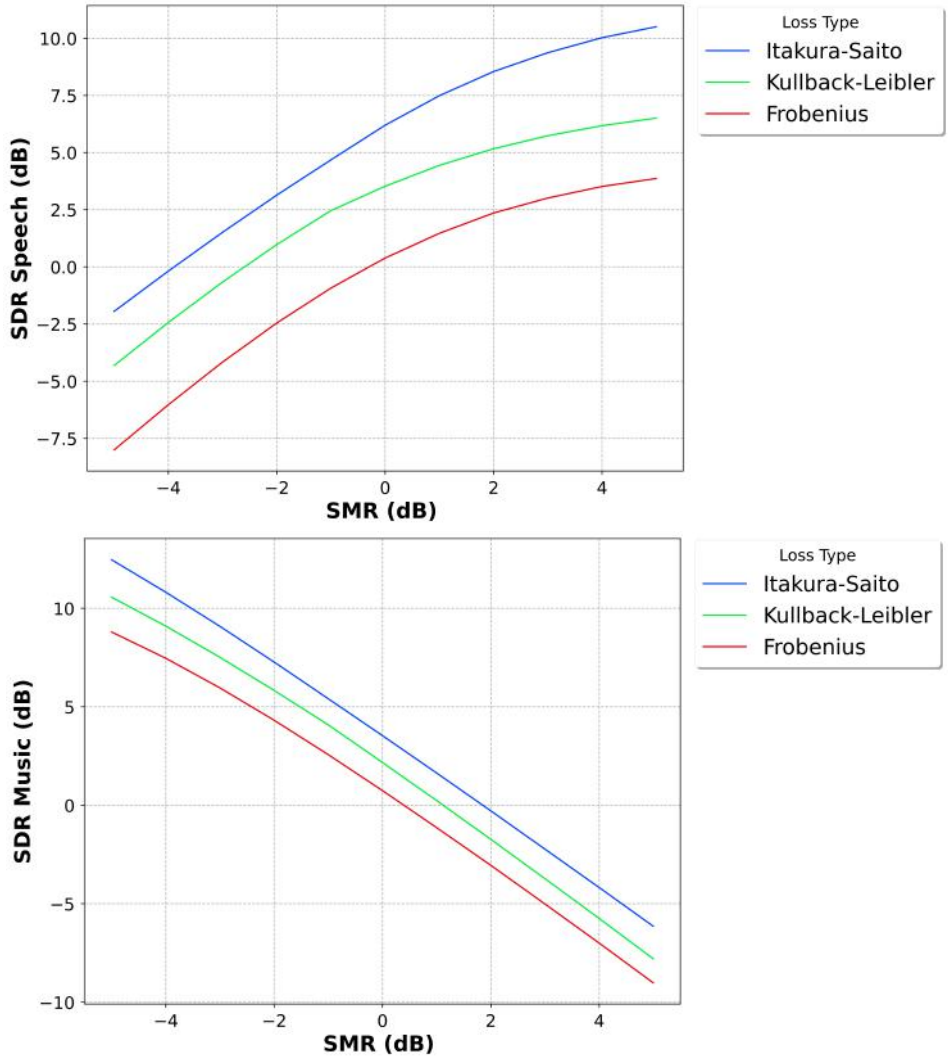


Figure 3.13: SDR of the estimated music signal using SNMF for different losses.

From Figure 3.13 and Table 3.3, we can notice that SNMF which minimizes IS Divergence better improves the SDR than using KL Divergence or Frobenius Norm. Something important to note, is that IS Divergence, as cited in subsection 3.1.1, is **scale invariant**, and is the only one of the β-Divergence function to have this property. It shows that IS Divergence gives the same relative weight to small as well as large time-frequency bins of the magnitude spectrum $\mathbf{X}$n meaning both low-power and high-power signal will be considered with the same relative importance. In contrast, factorizations obtained with $\beta > 0$ such as the Frobenius Norm and Kullback-Leibler Divergence, rely much more on the largest time-frequency bins, making the estimation of the low-power components less precise. In conclusion, we can say that the scale invariance of the IS divergence is appropriate to the decomposition of audio spectra and audio separation, that cares as much about low-power components such as note attacks (piano), as higher-power components such as tonal parts of sustained notes.

Chapter 4

Deep Learning Based Source Separation

4.1 Deep Neural Networks for Speech Separation.

A novel deep learning based source separation method was introduced in [16]. The purpose of this method is to enhance the separation of the model based method "non-negative matrix factorization (NMF)" by adding a deep neural network (classifier) that checks the validity of the estimated sources spectrograms and corrects these estimates. This approach is composed of 4 major steps that can be illustrated in Figure 4.1.

The method can be summarized in the following points:

- **Training Phase :**
  1. Train the clean speech dictionary W_s using the NMF with the multiplicative update algorithm (MU).
  2. Train the clean music dictionary W_m using the NMF with the MU.
- **Testing Phase:**
  1. Estimate the speech and music spectrums $\hat{S} = \{\hat{S}_1, \hat{S}_2\}$ using the test NMF from the mixed signal spectrum.
  2. Train the parameters $\mathbb{W}$ of the DNN using Restricted Boltzmann Machines (RBM) on the normalized estimated sources magnitude spectrums $\hat{S}$.
  3. Train the DNN to classify frames of normalized $\hat{S}$ using the pretrained weights $\mathbb{W}$.
  4. Apply a Wiener filter and reconstruct the estimated signal in the time domain.

$$\hat{S}_1 = \frac{(u \cdot \hat{S}_1)^2}{(u \cdot \hat{S}_1)^2 + (v \cdot \hat{S}_2)^2} \odot Y \tag{4.1a}$$

$$\hat{S}_2 = \frac{(v \cdot \hat{S}_2)^2}{(u \cdot \hat{S}_1)^2 + (v \cdot \hat{S}_2)^2} \odot Y \tag{4.1b}$$

Y : Complex spectrogram of the mixed signal.
u : Gain of source 1 (speech).
v : Gain of source 2 (music).
$\odot$: Element wise multiplication.

4.1.1 Neural Network Architecture

Authors of [16] used non-negative matrix factorization to model each source as non-negative linear combination of its dictionary. However, this property -linear combination- can be very limiting since the variability within sources spectrums can be better modeled using non-linearities

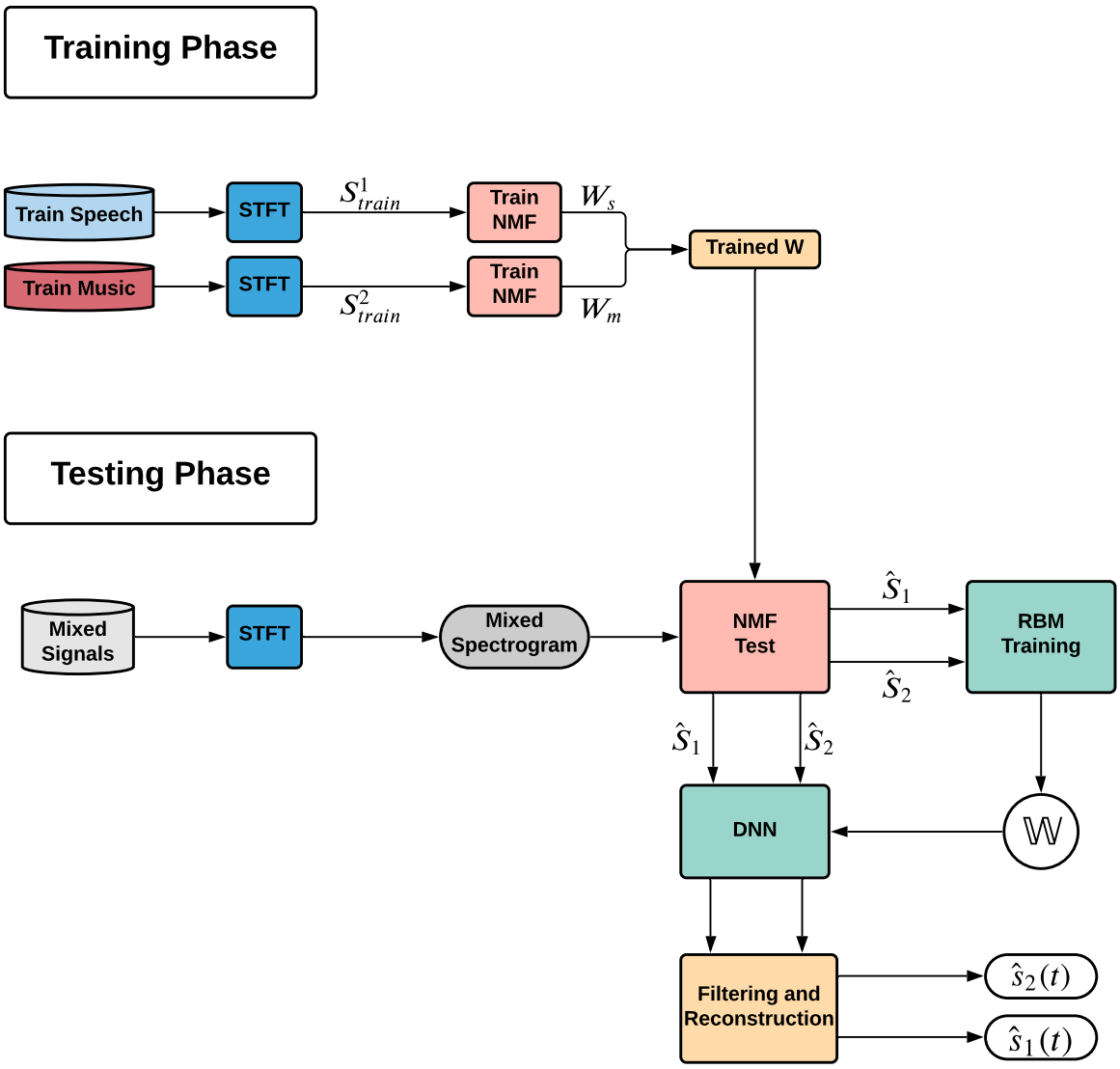


Figure 4.1: Pipeline of the method.

according to the authors. Therefore, the paper proposes to use a DNN to model each source in order to improve the separation. It was also mentioned that any other classifier can be used to replace the neural network (NN), but it will probably not have performances as high as the NN.

The neural network used has 3 hidden layers with 100-50-200 hidden units in each hidden layer respectively. The weights of the network are pretrained using Restricted Boltzmann Machines (RBM) [38]. The output layer has 2 neurons for classification, but in general, the output layer dimension is equal to the number of sources. The output of the model is a one hot encoding,

meaning that the first source have the label "10" and the second source "01". All the activation functions used are sigmoids because of the RBM initialization, more details about the reasons of such use of sigmoid can be found in [38].

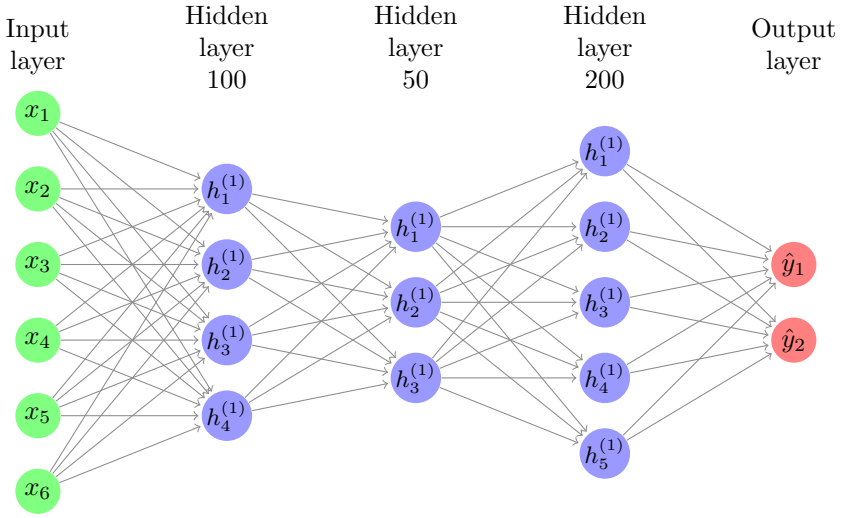


Figure 4.2: Architecture of the DNN.

Optimization Problem

The model is trained to classify the magnitude spectrums of the different sources, while keeping their sum equal to mixed spectrum and respect the non-negativity constraints.

The mixed spectrum is written as a linear combination of the sources' spectrum with the gain coefficients u and v as follows :

$$Y = u \cdot S_1 + v \cdot S_2, \qquad u, v \in \mathbb{R}^d \tag{4.2}$$

d : Dimension of the spectrums (Number of frequencies).

The initialization of the gains u and v is done by dividing the **L2** norm of $\hat{S}_1$ (respectively $\hat{S}_2$) by the **L2** of the mixed spectrum Y.

$$u_0 = \frac{\|\hat{S}_1\|_2}{\|Y\|_2} \qquad v_0 = \frac{\|\hat{S}_2\|_2}{\|Y\|_2} \tag{4.3}$$

The objective function to optimize consists of :

- Classification error.

$$E(S_1) = (1 - f1(S_1))^2 + (f2(S_1))^2, \tag{4.4a}$$

$$E(S_2) = (f1(S_2))^2 + (1 - f2(S_2))^2, \tag{4.4b}$$

$E(S_1)$: It is expected to be minimum when the speech is classified as [1,0].
$E(S_2)$: It is expected to be minimum when the music is classified as [0,1].

- The energy of the least squares difference error between the mixed signal spectrum Y and the linear combination of source spectra estimates.

$$SE(S_1, S_2, u, v, Y) = ||u \cdot S_1 + v \cdot S_2 - Y||^2 \tag{4.5}$$

- Non-negativity constraint.

$$E_{nn} = \sum_{i=1}^{2} min(S_i, 0)^2 + min(u, 0)^2 + min(v, 0)^2 \tag{4.6}$$

The optimization problem can be formulated as follows:

$$\left(\hat{\boldsymbol{S}}_1, \hat{\boldsymbol{S}}_2, \hat{u}, \hat{v}\right) = \underset{\{\boldsymbol{S}_1, \boldsymbol{S}_2, u, v\}}{\operatorname{argmin}} \mathcal{L}\left(\boldsymbol{S}_1, \boldsymbol{S}_2, \boldsymbol{Y}, u, v\right), \tag{4.7}$$

Where the loss function :

$$\boxed{\mathcal{L}(S_1, S_2) = E(S_1) + E(S_2) + \lambda \cdot SE(S_1, S_2, u, v, Y) + \beta \cdot E_{nn}} \tag{4.8}$$

Optimizer

The optimizer used to solve this optimization problem is called L- Broyden, Fletcher, Goldfarb, and Shanno (L-BFGS). It's a second order optimization algorithm, meaning that it makes use of the second-order derivative also called the Hessian matrix to find the local minimum of the objective function. The advantage of using this algorithm is that the Hessian matrix can be used to determine both the direction and the step size to move in order to change the input parameters. The original BFGS algorithm has to store the inverse Hessian matrix which requires $O(n^2)$ memory making it unsuitable for deep learning models that have a large amount of parameters.

Limited Memory BFGS or L-BFGS is an extension to BFGS algorithm that deals with the high memory requirements problem. It assumes some simplifications about the Hessian matrix and applies some approximations to avoid storing the whole inverse matrix.

4.1.2 Filtering and Reconstruction

After solving the optimization problem, a Wiener filter is applied to the spectral estimates to improve the separation. The expressions of the new spectrum estimates take this form :

$$\hat{\boldsymbol{S}}_1 = \frac{\left(\hat{u}\hat{\boldsymbol{S}}_1\right)^2}{\left(\hat{u}\hat{\boldsymbol{S}}_1\right)^2 + \left(\hat{v}\hat{\boldsymbol{S}}_2\right)^2} \odot \boldsymbol{Y} \tag{4.9a}$$

$$\hat{\boldsymbol{S}}_2 = \frac{\left(\hat{v}\hat{\boldsymbol{S}}_2\right)^2}{\left(\hat{u}\hat{\boldsymbol{S}}_1\right)^2 + \left(\hat{v}\hat{\boldsymbol{S}}_2\right)^2} \odot \boldsymbol{Y} \tag{4.9b}$$

$\odot$: Element wise multiplication.
$\hat{u}, \hat{v}$: Estimates of the gains.

DNN Initialization

Authors of [16] used RBM in order to pre-train the DNN architecture ; as an initialization process for the model in order to converge quickly. Specially, they used a stack of RBM to mimic the DNN architecture. The RBM's process described in subsection 1.4.8 remain the same, but with the necessity of assembling multiple RBM together. Finally, They trained the RBM by feeding the estimated magnitude spectra of each source sequentially, representing the visible layer, and so on until the final layer composed of 2 nodes/neurons representing a compressed representation of the imputed magnitude spectra.

4.1.3 Drawbacks of this approach

Although using a DNN to check the validity of the NMF estimated spectrums shows promising results in the original paper, it does come with some weaknesses that we have worked on improving in the next sections. The following points summarize the drawbacks of this method.

- Using an RBM to initialize the DNN weights for each mixed signal is time consuming. A better way would be to initialize the weights of the DNN only one time and use the same pretrained weights for all the test mixed signals.
- The RBM makes the model limited to using sigmoid activation functions. While many other activations can be considered to have a positive impact on the performances of the model, especially when used in the hidden layers such as ReLU.
- RBMs are shallow (two-layer neural nets) and training all the weights of the DNN requires training a stack of RBMs. In contrast, the AutoEncoders that are showing a great success in the reconstruction and the compression (dimentionality reduction) can have a deep architecture which can be better for initializing the weights of the model.

- The non-negativity constraint used $E_{nn} = \sum_{i=1}^{2} min(S_i, 0)^2 + min(u, 0)^2 + min(v, 0)^2$ when applied on normalized values gives a very small value that is negligible compared to the other terms of the objective function, even with a high value of β because it is squared.
- The gain parameters u and v are time independent and do not adapt to the changes of the proportions between the different signals across time.
- Using the optimizer (L-BFGS) to solve the optimization problem is very slow because it's a second order method and it requires computing an approximation of the inverse Hessian matrix.

4.2 Our contribution

In this section, we proposed two approaches to improve the Signal to Distortion Ratio (SDR). Both are based on autoencoders, that aim to enhance the mixture separation in two domains (see figure 4.3):

- Change the magnitude spectrum $\mathbf{s}_1$ and $\mathbf{s}_2$ of the speech and music signal estimated by the SNMF using the DE-SNMF in the time-frequency domain.
- Change the gains $\mathbf{H}$ estimated by the SNMF using the Denoising-SNMF in the temporal domain.

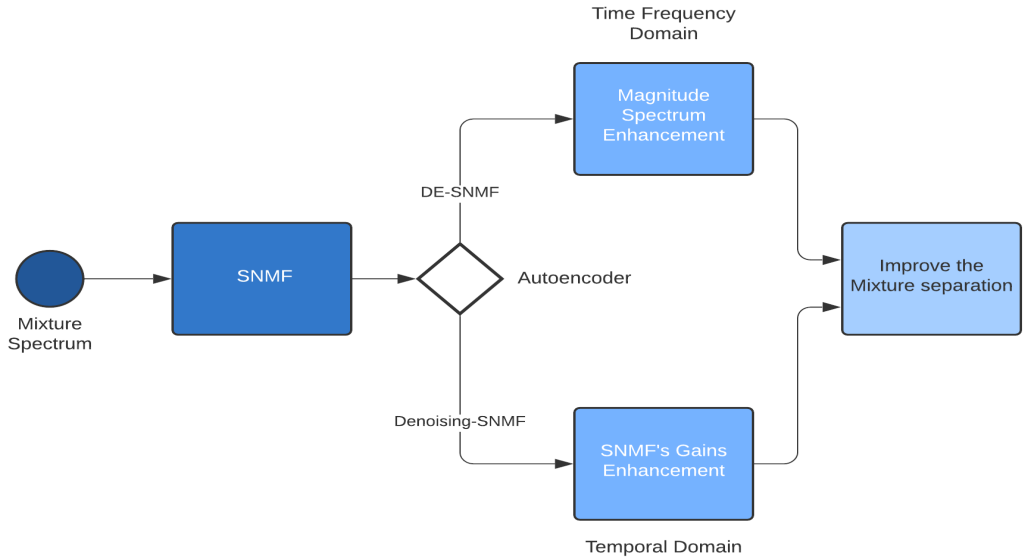


Figure 4.3: Proposed approaches based on autoencoder to improve the mixture separation

4.2.1 Autoencoders

Autoencoders are special types of neural networks that learn to reconstruct input data from its compressed version that hold meaningful representation through $l \in \{1..L\}$ hidden layers. These kind of architectures are composed of two symmetrical part, as shown in the figure 4.4 :

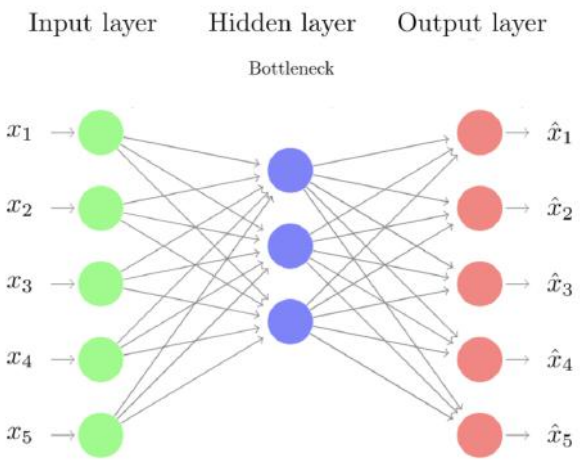


Figure 4.4: Typical autoencoder architecture

The encoder

The encoder part is responsible of encoding, representing the original input space $X \in \mathbf{R}^{N_X \times D_X}$ in a smaller space $h^{[l]} \in \mathbf{R}^{N_X \times D_H^{[l]}}$ with $D_H^{[l]} < D_X$ where $h^{[l]}$ denotes the l^{th} hidden layer, and $D_H^{[l]}$ denotes its dimensionality, using an ascendant combination of layers. The final hidden encoding layer is usually called bottleneck, because it forces the network to compress the data into what we called a latent space. We use the encoder function (Eq. 4.10) to encode the input for the hidden layer based on the visible layer's vector x_t:

$$\mathbf{h}_t^{[l]} = f(\mathbf{x}_t^{[l])}) = \sigma\left(\mathbf{W}_{en}^{[l]^T}\mathbf{x}_t^{[l])} + \mathbf{b}_{en}^{[l]}\right) \in \mathbb{R}^{D_H^{[l]}} \tag{4.10}$$

Where$D_H^{[l]}$ is the dimensionality of the l^{th} hidden layer's vector, $\mathbf{W}_{en}^{[l]} \in \mathbb{R}^{D_X^{[l]} \times D_H^{[l]}}$ is the weight matrix of the encoder, $\mathbf{b}_{en}^{[l]} \in \mathbb{R}^{D_H^{[l]}}$ is the bias vector of the l^{th} decoder layer.

The decoder

This part is responsible of decoding the latent representation, which aims to reconstruct the original input space $\widehat{X} \in \mathbf{R}^{N_X \times D_X}$ from its compressed representation. To reconstruct the data in the reconstruction layer, we use Eq.(4.11) to decode the data in the hidden layer:

$$\hat{\mathbf{x}}_t^{[l]} = g(\mathbf{h}_t^{[l]}) = \sigma\left(\mathbf{W}_{de}^{[l]T}\mathbf{h}_t^{[l]} + \mathbf{b}_{de}^{[l]}\right) \in \mathbb{R}^{D_X^{[l]}} \tag{4.11}$$

In Eq.(4.11), $\mathbf{W}_{de}^{[l]} \in \mathbb{R}^{D_X^{[l]}} \times D_H^{[l]}$ and $\mathbf{b}_{de}^{[l]} \in \mathbb{R}^{D_X^{[l]}}$ are the weight matrix and bias vector of the decoder, respectively, and $\hat{x}_t$ is the reconstruction of the input x_t at the t timestamp.

If an autoencoder succeeds in simply learning to reconstruct the input perfectly, then the architecture didn't learn useful representation. Instead, autoencoders are designed to be unable to learn a perfect copy of the input. Therefore, they are restricted in ways that allow them to reconstruct only approximately, and copy only the input that resembles the training data. Because the model is forced to prioritize which aspects of the input should be copied, it often learns useful properties of the data. This chapter presented autoencoders showing how the naive architectures that were first defined for them evolved to powerful models with the core abilities to learn a meaningful representation of the input and to model generative processes. To conclude, the goal of autoencoders is to get a compressed and meaningful representation. We would like to have a representation that is meaningful to us, and at the same time good for reconstruction. In that trade off, it is important to find the architectures which serves all needs.

4.2.2 Adaptive Moment Estimation

Adaptive moment estimation known in the deep learning community as Adam is an optimization algorithm that was introduced in [58] as an extension to stochastic gradient descent. This algorithm combines the advantages of AdaGrad and RMSProp (section 1.4.7). Whereas RMSProp uses an exponential moving average of the gradient, Adam also makes use of the average of the second moments of the gradients (the uncentered variance). In other words, the algorithm computes an exponential moving average of the gradient and the squared gradient. The decay rates are controlled through the parameters β_1 and β_2.

A paper was published in 2016 titled "An overview of gradient descent optimization algorithms" [59] presents an overview of the most popular optimization algorithms in deep learning. The paper in the section "Which optimizer to choose?" mentioned that for sparse data, adaptive learning rate methods are likely to achieve best performances. It is also noted that the use of the second moments helps Adam slightly outperform RMSprop towards the end of optimization as gradients become sparser.

$$m_t = \beta_1 \cdot m_{t-1} + (1 - \beta_1) \cdot g_t \tag{4.12a}$$

$$v_t = \beta_2 \cdot v_{t-1} + (1 - \beta_2) \cdot g_t^2 \tag{4.12b}$$

m_t : Estimate of the first order moment (mean) of the gradients.
v_t : Estimate of the second order moment (uncentered variance) of the gradients.

Note : v_t is the estimate of the **uncentered** variance because the expression of the variance is $var(x) = \mathbb{E}[x^2] - (\mathbb{E}[x])^2$ whereas the update of v_t contains only the term g_t^2 without subtracting

the square of the mean of the gradients.

Initializing the moments v_t and m_t with zeros causes the gradients to be biased towards zero. For an initial estimate m_0, the first order moment $m_1 = \beta_1 \cdot m_0 + (1 - \beta_1) \cdot g_1$ is highly dependent on m_0. Therefore, the first steps of the moving average is heavily biased towards the initial m_0. Avoiding this problem requires setting a bias-corrected moments estimate as follows :

$$m_1 = \beta_1 \cdot m_0 + (1 - \beta_1) \cdot g_1 \implies \hat{m}_1 = g_1 = \frac{m_1 - \beta_1 \cdot m_0}{1 - \beta_1} \tag{4.13}$$

Considering $m_0 = 0$ makes the estimate of the moments as following:

$$\hat{m}_1 = \frac{m_1}{1 - \beta_1} \tag{4.14}$$

The general expression for each step t becomes :

$$\hat{m}_t = \frac{m_t}{1 - \beta_1^t} \tag{4.15a}$$

$$\hat{v}_t = \frac{v_t}{1 - \beta_2^t} \tag{4.15b}$$

Finally, the update rule of Adam algorithm is the formula 4.16

$$\Theta_t = \Theta_{t-1} - \frac{\eta}{\sqrt{\hat{v}_t} + \epsilon} \cdot \hat{m}_t \tag{4.16}$$

η : Learning rate.
ϵ : Constant empirically set to 10^{-8}.
β_1 and β_2 : Parameters that have usually the default values of 0.9 and 0.999 for better results.

4.2.3 Deep Enhanced Sparse NMF

In this section, we propose a new approach based on deep learning to enhance the performances of the Sparse NMF. The pipeline of this approach can be summarized as follows (Figure 4.5).

1. Estimate the magnitude spectrum of speech and music using Sparse NMF with pretrained dictionary W and time frequency (e.g. Wiener Filter).

2. Train an autoencoder on speech and music spectrograms estimated by the Sparse NMF on the training data that was used to obtain the pretrained dictionary W.

3. Use the encoder's pretrained weights to initialize the Deep Neural Network (DNN) weights.

4. The DNN will update its weights as well as the input magnitude spectrograms of the speech and music that were estimated by the SNMF.

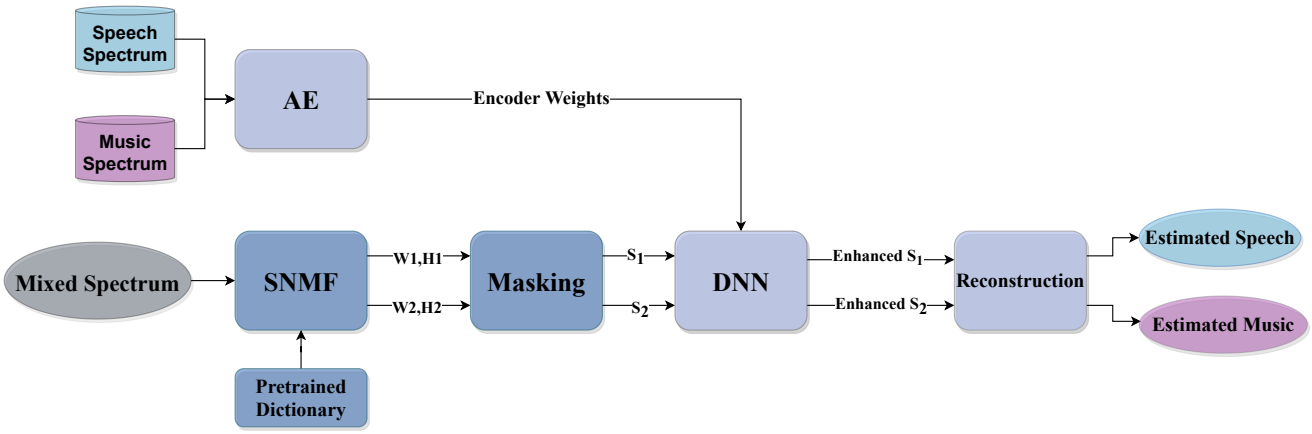


Figure 4.5: Pipeline of Deep Enhanced Sparse NMF.

Architecture

The Neural Network contains hidden layers with 128-64-32-2 hidden units respectively. It's main purpose is to update the magnitude spectrogram of the speech and music predicted by the Sparse NMF. The DNN learns to classify each sample of the two spectrograms into two classes (Speech and music) and it is constrained to keep the magnitude spectrograms non-negative and their sum must remain equal to the magnitude spectrogram of the mixed signal. Therefore the cost function contains 3 terms :

- Classification error.

$$E(S_1) = (1 - f_1(S_1))^2 + (f_2(S_1))^2, \tag{4.17a}$$

$$E(S_2) = (f_1(S_2))^2 + (1 - f_2(S_2))^2, \tag{4.17b}$$

  $E(S_1)$: Classification error of the first source (Speech) where the error is minimized when the DNN output is [1,0].
  $E(S_2)$: Classification error of the second source (Music) where the error is minimized when the DNN output is [0,1].

- The Frobenius norm of the difference between the mixed signal magnitude spectrogram and the sum of the estimated speech and music spectrogram.

$$SE(S_1, S_2, S) = ||S_1 + S_2 - Y||_F \tag{4.18}$$

- Non-negativity constraint.

$$E_{nn} = \sum_{i=1}^{2} min(S_i, 0) \tag{4.19}$$

The cost function can be written as follows :

$$\boxed{\mathcal{L}(S_1, S_2) = E(S_1) + E(S_2) + \lambda \cdot SE(S_1, S_2, S) + \beta \cdot E_{nn}} \tag{4.20}$$

The parameters λ and β are regularization parameters that have a very high impact on the performances of the model. Therefore, these two hyper-parameters should be considered in the tuning process. Values of $\lambda = 10$ and $\beta = 6$ yield empirically to better results because they help the model to achieve higher values of SDR in fewer epochs and offers a good trade off between the non-negativity and the summation constraints.

The optimization problem consists of minimizing the cost function 4.21 with respect to the model parameters $\mathbb{W}$ and the magnitude spectrograms S_1 and S_2 of the speech and the music.

$$\{\hat{S}_1, \hat{S}_2, \mathbb{W}\} = \underset{S_1, S_2, \mathbb{W}}{argmin}\, \mathcal{L}(S_1, S_2) \tag{4.21}$$

The optimizer used to solve this problem is Adaptive Moment Estimation (Adam) that was presented in subsection 4.2.2.

Undercomplete AutoEncoder

The AutoEncoder we used is undercomplete (dimension of the hidden layers is less than the input dimension), symmetrical and has 3 hidden layers in its encoder and decoder. The number of nodes in each hidden layer is 128-64-32-2-32-64-128 as shown in Figure 4.6. This autoencoder is trained to reconstruct the magnitude spectrogram of speech and music estimated using the Sparse NMF. The data that was used to train it is the same as the one used to get the pretraind dictionary W of the SNMF.

The purpose behind using an undercomplete AutoEncoder is:

- Its ability to extract important features from the data due to its dimentionality reduction properties. Therefore, we use it as a feature extractor to initialize our DNN (Classifier).
- It's a better approach than training the Restritced Boltzman Machine (RBM) only on the test data as they did in [16] because the AutoEncoder in our method is trained on the training data (Bigger size than test data) and this will help the DNN to use the salient features learned from the training data allow the model to learn the general distribution of the data.

- The AutoEncoder is only trained once. Which is an improvement in terms of time complexity compared to [16].

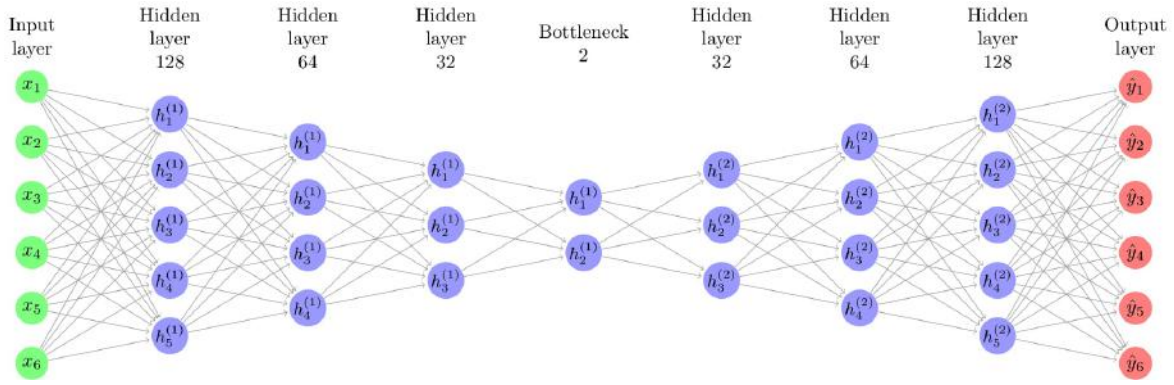


Figure 4.6: AutoEncoder Architecture.

The training process of the AE is based on the optimization of the cost function (Eq.4.22) that represents the Frobenius Norm of the difference between the reconstructed spectrogram of the signal (speech or music) and the estimated spectrogram using the Sparse NMF (input of the AutoEncoder).

$$\mathcal{L}(S, \hat{S}) = \|S - \hat{S}\|_F \tag{4.22}$$

$\hat{S}$: Reconstruction of the spectrogram.
S : Spectrogram estimated by the SNMF.

Moreover, after trying many architectures and configurations, we concluded that the bias term should not be considered in the AutoEncoder because the cost function of the AE did not decrease (see Figure 4.7. Besides, We tried an architecture of AE that contains all the layers of the DNN (including classification layer) and another architecture that does not include the last layer. While the loss of the second AE seems to be less than the first, training all the layers seems to have a good impact on the convergence of the DNN and for this reason we used the AE that has a bottleneck layer dimension equal to 2.

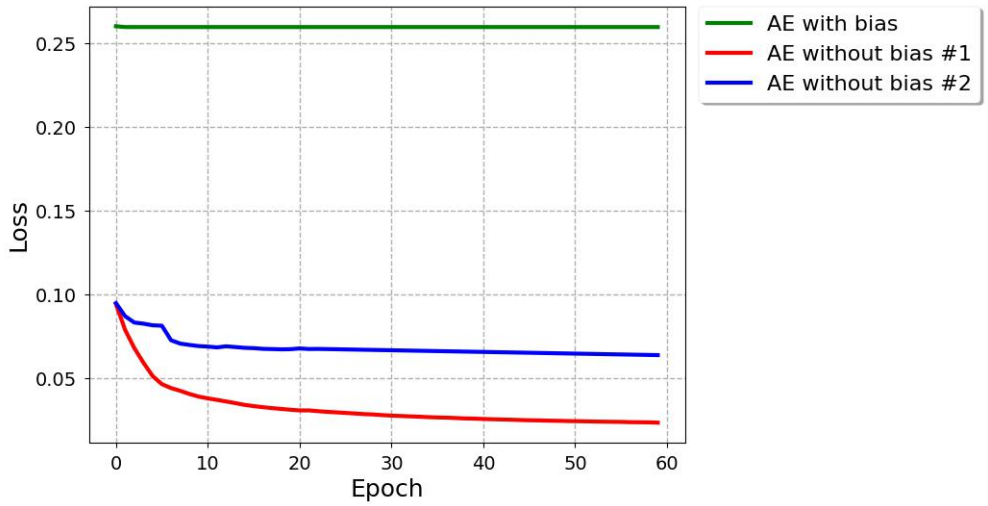


Figure 4.7: Evolution of the cost function of the AutoEncoder with and without bias.

AE trained with only SMR = 0

We notice that the DNN model performed well for SMR = -5 and SMR = 0 but did not improve the SDR for an input mixed signal with SMR = 5. This is due to the fact that for mixing the signals in SMR = -5 we have multiplied the music signal by a scale factor and kept the speech signal as it is. Therefore, the power of the speech signal in SMR= -5 and SMR = 0 is the same and the difference is in the power of the music signal. This explains why the trained AE on SMR = 0 gives also good performances for SMR = -5. However, for an input mixed signal with SMR = 5, the speech signal was multiplied by a scale factor, this makes it different from the speech signal used for training the AutoEncoder and this is the reason why it did not improve the NMF SDR. Generalizing the results of mixed signals with SMR = 0 and -5 requires training the AutoEncoder on different SMRs and Speech power or using a different kind of normalization to set the speech signal to the same power as of the training data of the Auto Encoder. A solution to this problem is described in section 4.3.1.

Algorithm	SMR Input (dB)	SDR Speech (dB)	SDR Music (dB)
Sparse NMF	-5	0.82	12.58
	0	5.98	3.44
	5	8.71	-6.7
DE-SNMF	-5	2.5	12
	0	6.99	3.41
	5	8.67	-6.7

Table 4.1: Comparaison between DE-SNMF and SNMF.

4.2.4 Denoising Sparse NMF

In this section, we propose a second new approach based on deep learning to enhance mixture separation did earlier by the Trained Sparse NMF. This method is characterized by its simplicity, because it targets the noisy aspect of time series data. The pipeline of this approach can be summarized as follows (Figure 4.5), and consists of 4 main steps:

1. Estimate the activations **H** using Sparse NMF with pretrained dictionary W.
2. Initialize weights of the autoencoder using *He* Initialization.
3. Train the denoising autoencoder on normalized activations **H** estimated by the Sparse NMF during the test stage.
4. Use the disrupted activations $\tilde{\mathbf{H}}$ to estimate the new magnitude spectrum of speech and music source through masking operation after the model inference.

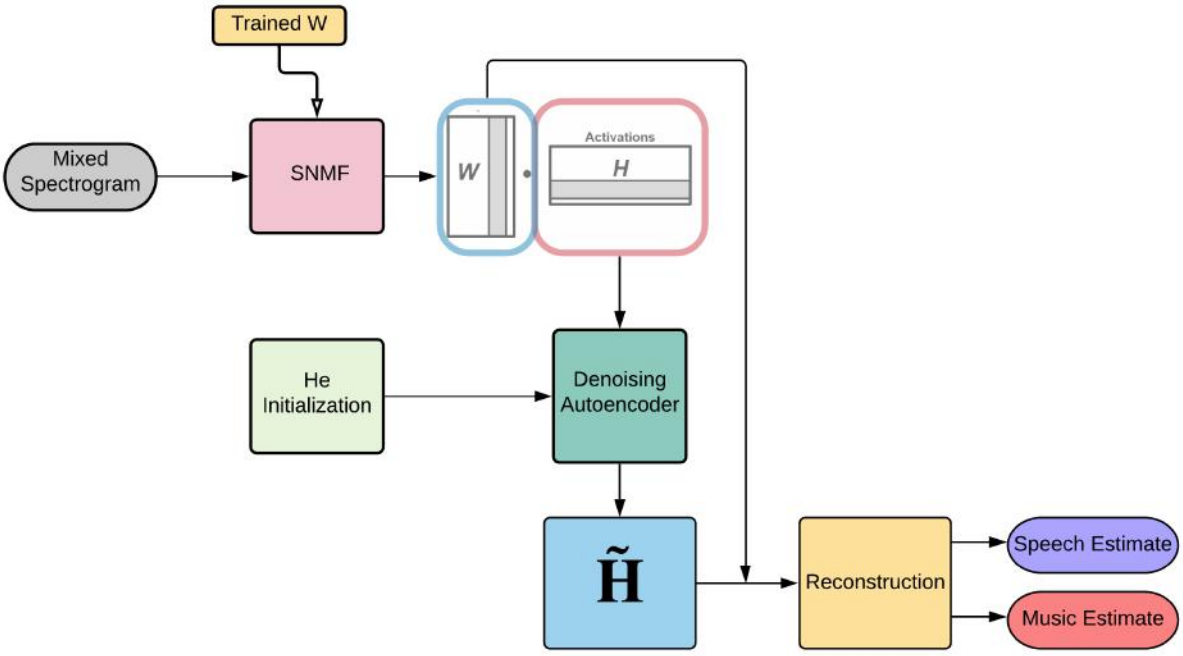


Figure 4.8: Pipeline of Denoising Sparse NMF.

4.2.5 Denoising Autoencoder

Denoising NMF takes advantages of the denoising autoencoder (DAE) to enhance the Signal to Distortion Ratio (SDR). Whereas most of the autoencoders are interested in the compression of

the input in order to learn an undercomplete representation in order to capture the most salient features of the training data, we used the Denoising Autoencoder as a robust architecture for error correction. In this model, the input is disrupted by some noise using Dropout that we cited in Section (1.4.6) and the denoising autoencoder is expected to reconstruct the clean version of the input, as illustrated in Figure (4.9).

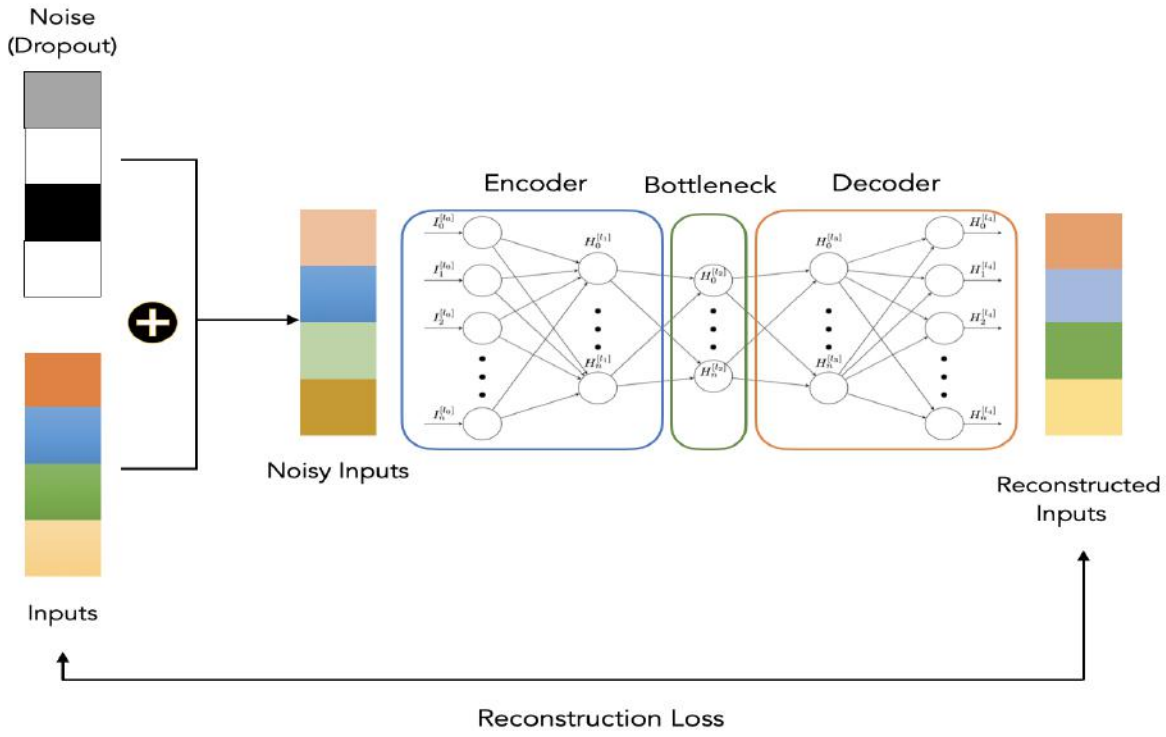


Figure 4.9: Denoising Autoencoder [60]

Traditionally, autoencoders minimize the reconstruction loss function Eq. 4.23:

$$L(\boldsymbol{x}, g(f(\boldsymbol{x}))) \tag{4.23}$$

where L is a loss function penalizing $f(g(\boldsymbol{x}))$ for being dissimilar from $\boldsymbol{x}$, such as the $L2$ norm of their difference. This encourages $f \circ g$ to learn to be merely an identity function if they have the capacity to do so.
A denoising autoencoder instead minimizes Eq. 4.24:

$$L(\boldsymbol{x}, g(f(\tilde{\boldsymbol{x}}))) \tag{4.24}$$

where $\tilde{\boldsymbol{x}}$ is a copy of $\boldsymbol{x}$ that has been corrupted by some form of noise, that we created using dropout that we introducted in subsection (1.4.6). We then constraint the Denoising autoencoder to undo this corruption rather than simply copying their input, it will constrain the DAE to learn

the general distribution of the input, while learning to cancel noise present in the data. In summary, in denoising autoencoders, the emphasis is on letting the encoder be resistant to some perturbations of the input, in our case, to enhance the SDR by denoising the gains $\mathbf{H}$ of the Non Negative Matrix Factorization (3.1) applied on the magnitude spectrum of the mixed signal.

The Architecture

The Neural Network contains 5 hidden layers with 40-20-10-20-40 hidden units respectively, while the input and output layer contain as much as neurons as the number of components used in SNMF. The main purpose of our algorithm is to reconstruct the masks or activations $\mathbf{H}$ estimated by the Trained Sparse NMF in order to improve the Signal to Distorsion Ratio (SDR). The DNN learns an encoding of the input data $\mathbf{H}$ constrained to extract only the salient features from the input during the encoding phase, which enables a kind of data denoising which enhances the quality of the mixture separation.

The optimization problem consists of minimizing the Mean Squared Error loss between the activations $\mathbf{H}$ estimated by the SNMF, and the uncorrupted reconstruction of $\tilde{\mathbf{H}}$ calculated during the forward-pass, with respect to the model parameters.

$$\text{MSE} = \frac{1}{n}\sum_{i=0}^{n-1}\left(\mathbf{H}_i - \widetilde{\mathbf{H}}_i\right)^2 \tag{4.25}$$

Where n design the number of point, $\mathbf{H}_i$ the observed values, and $\widetilde{\mathbf{H}}_i$ represents the predicted/reconstructed values. Furthermore, the optimizer used to solve this problem is Adaptive Moment Estimation (Adam) that was presented in section 4.2.2 and also used in the Deep Enhanced Sparse NMF 4.2.3.

4.3 Results and Discussion

4.3.1 Deep Enhanced Sparse NMF

Classification.

As presented in 4.2.3, the DE-SNMF method is based on a DNN that classifies the input spectrogram into a speech and music. The classification of the model is evaluated using the activations of the two neurons that represent a one hot encoding (10 for speech and 01 for music). Table 4.2 and 4.3 summarizes the accuracy of the model for two different types of inputs:

- The music spectrogram is scaled with respect to the training music data of the AutoEncoder.
- The music spectrogram is not scaled.

SMR	-5	0	5
Speech	100%	100%	97.88%
Music	100%	100%	43.18%

Table 4.2: Classification accuracy of speech and music without scaling music spectrogram.

SMR	-5	0	5
Speech	100%	100%	98.76%
Music	100%	100%	84.09%

Table 4.3: Classification accuracy of speech and music with scaled music spectrogram.

The accuracy of the model is high for the different SMR values in the case of the unscaled music spectrogram. However, we can observe a slighlty lower accuracy for speech in SMR = 5 and a very low accuracy (43.18 %) for the music. This is due to the fact that in SMR = 5 the music signal have been attenuated to create the mixed signal.

$$Mixed = \frac{1}{scale} \cdot music + speech \tag{4.26}$$

This attenuation has caused perturbations to the model because the AutoEncoder was trained on spectrograms of mixed signals with SMR = 0 and the spectrograms estimated using the Sparse NMF did not lead to good results in terms of music SDR for SMR = 5. The estimated music spectrogram contains a lot of distortions that make the task of classifying the music spectrogram harder. To solve this problem, we have scaled the music spectrogram estimated using the test Sparse NMF with respect to the power of the music spectrogram used for training the AutoEncoder. In other words, we want to have the power of the test music to be approximately in the same scale as the power of the music used in the training.

$$\alpha = \frac{\sum\limits_{i,j}|M_2|}{\sum\limits_{i,j}|\hat{S}_2|} \tag{4.27a}$$

$$\hat{S}_2 = \hat{S}_2 \cdot \alpha \tag{4.27b}$$

$\hat{S}_2$: Spectrogram of music estimated using the test Sparse NMF.
M_2 : Spectrogram of music used to train the AutoEncoder.
α : Scaling factor.

It can be noticed that scaling the music spectrogram highly impacts the classification of the DNN model in a positive way and increases the accuracy of classifying the music spectrogram for SMR = 5 from 43.18 % to 84.09 % while keeping a perfect accuracy for other SMR levels. A

more visual approach to understand this improvement is by visualizing the distribution of the first neurone activation that in the best cases should always output a 1 for speech and 0 for music. Figures 4.10 and 4.11 illustrates this distribution for a DNN trained on scaled and unscaled music spectrograms respectively.

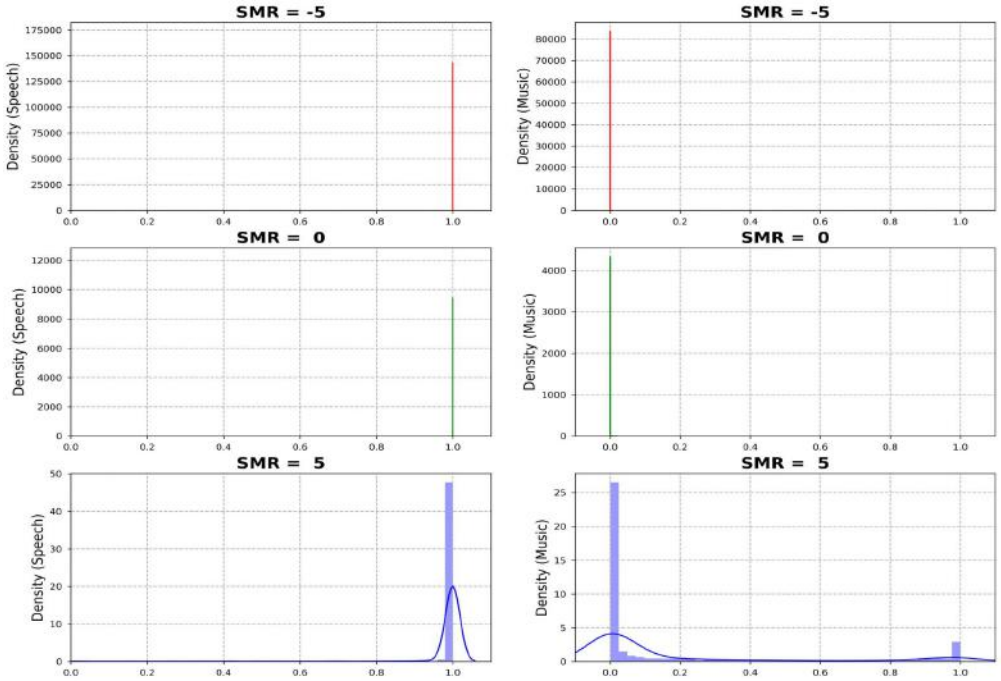


Figure 4.10: Distribution of the activation of the first neuron for a DNN trained on scaled music spectrograms.

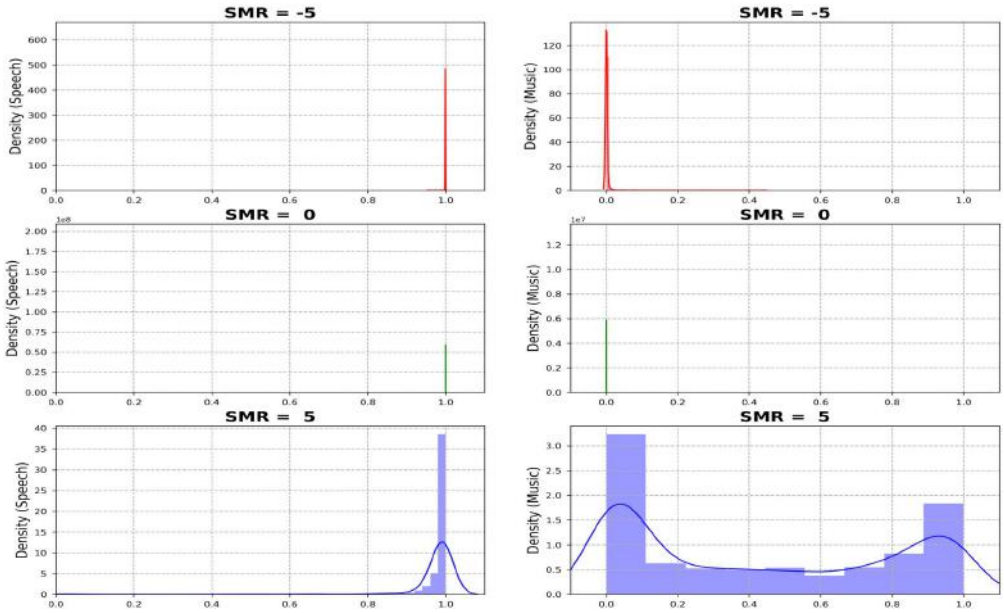


Figure 4.11: Distribution of the activation of the first neuron for a DNN trained on unscaled music spectrograms.

Two main conclusions can be drawn from the distribution plots :

- The density function has higher values for scaled music spectrograms than unscaled music spectrogram. For example, for SMR = -5 the density function of the speech activation achieves 14000 counts for scaled music, while its value barely reaches 500 counts.
- For SMR = 5 the distribution of the music activations density function is flattened for an unscaled music spactrogram. This means the model is confused to classify the music in SMR = 5. However, scaling the music leads to a better distribution of the activations as shown in Figure 4.10.
- Even though the accuracy of the model in SMR = -5 and SMR = 0 is the same for scaled and unscaled music, The probability of a good classification (Activation of the neuron) is higher when the DNN is trained on scaled music spectrograms.

Loss and Reconstruction

Updating the speech and music spectrograms using the DNN improved the separation and reduced the distortions that are present in the reconstructed signals. Table 4.4 summarizes the results of the Sparse NMF with Wiener filter and our model DE-SNMF. Our proposed estimation approach has increased significantly the speech SDR.

Algorithm	SMR Mixed (dB)	SDR Speech (dB)	SDR Music (dB)
SNMF	-5	-1.89	12.47
	0	6.22	3.54
	5	10.49	-6.17
DE-SNMF	-5	**3.56**	12.07
	0	**7.04**	3.48
	5	**12.97**	**-0.86**

Table 4.4: DE-SNMF SDR results in dB evaluated for different input SMR values.

The results we have obtained using the DE-SNMF outperforms the results of the DNN presented in [51]. But since we did not use the same dataset, the comparison of the SDR results is not very significant. Therefore, instead of comparing directly the SDR we compare the relative improvement between the NMF and the DNN (Table 4.5). We conclude from the table that the enhancement of the signal in our method surpasses the model in [51] for SMR = 5 and SMR = -5 which means that our model leads to a better reconstruction even when the speech power is very small compared to music. Finally, Figures 4.12, 4.14 and 4.13 represents the evolution of the loss, the classification loss and the SDR overs the training epochs. We can notice that when

the global loss and the classification loss converges the SDR becomes more stables in a certain value. Therefore, the criteria to stop the training depends on the decreasing factor of the speech classification loss because it's more linked to the SDR (see Figure 4.14). It's important to note that the model takes around 50 seconds to enhance the magnitude spectrogram of each source estimed previously by the Trained SNMF.

Algorithm	SMR Mixed (dB)	SDR Speech (dB)	SDR Music (dB)
DNN of [51]	−5	1.3	1.15
	0	1.22	0.94
	5	0.97	1.04
DE-SNMF	−5	**5.42**	1.04
	0	0.82	−0.06
	5	**2.48**	**5.33**

Table 4.5: DE-SNMF SDR results in dB evaluated for different input SMR values.

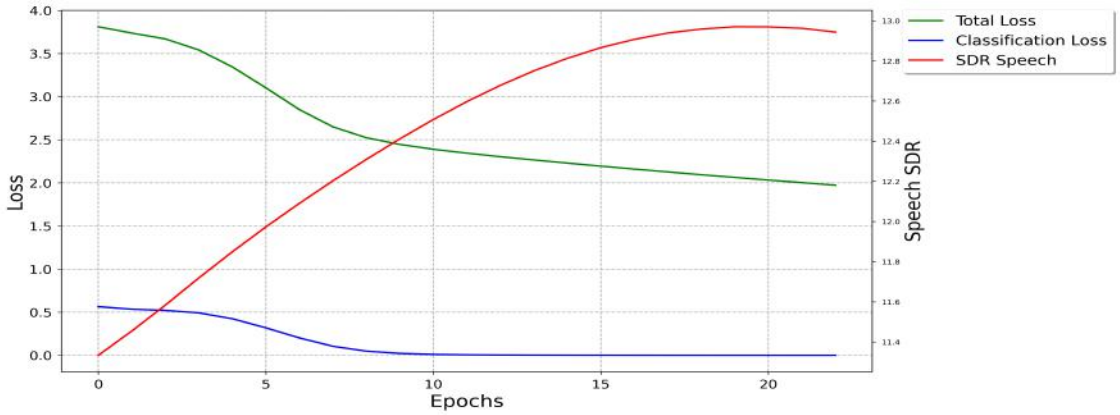


Figure 4.12: Global loss, classification loss and SDR of the speech signal for SMR = 5 using DE-SNMF method.

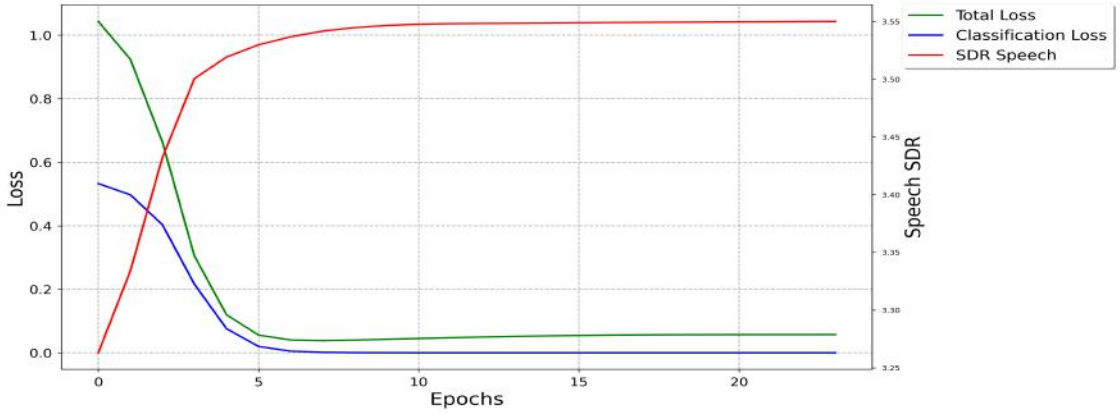


Figure 4.13: Global loss, classification loss and SDR of the speech signal for SMR = -5 using DE-SNMF method.

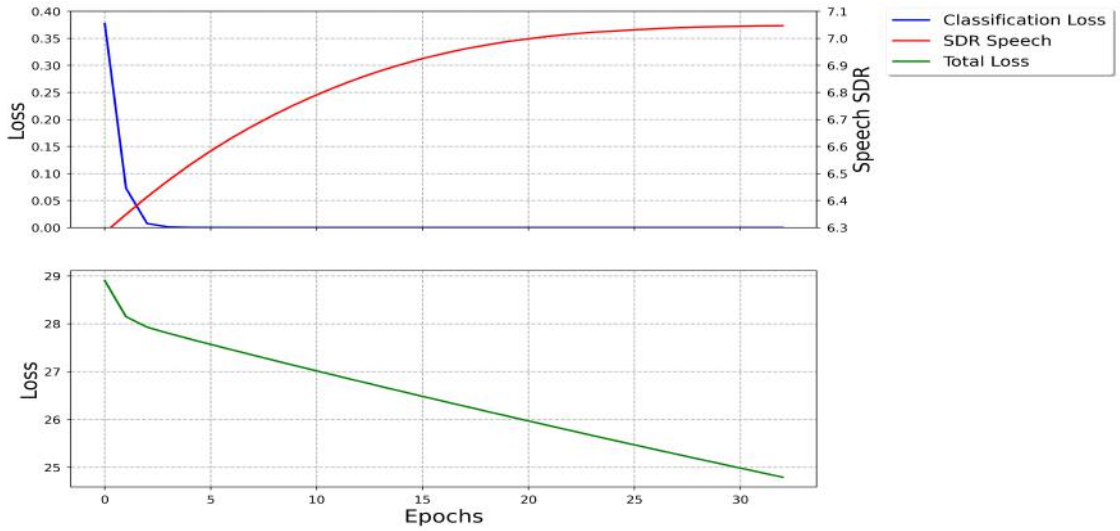


Figure 4.14: Global loss, classification loss and SDR of the speech signal for SMR = 0 using DE-SNMF method.

4.3.2 Denoising SNMF

We applied the Denoising-SNMF to separate speech and music signals from their mixture signal. The proposed algorithm was applied on a collection of speech and piano data sampled at 44kHz then down-sampled at 16kHz. For speech data, we used the training and testing male and female speech conversation data from an Audio Book [61]. For music data, we downloaded piano music from Youtube [62]. For the initialization of the NMF we used Nonnegative Double Singular Value Decomposition (NNDSVD) [63] that respects the non negativity constraint of the NMF, we used a dictionnary size of 24 and 48 to represent the speech and the music sources respectively. We then used 20 minutes of both clean speech and clean piano music to train dictionnaries of SNMF, and test the latter on a mixture of 30 seconds by updating only the masks $\boldsymbol{H}$ while keeping the

basis vectors $\boldsymbol{W}$ fixed. A test SNMF was used to obtain the activations $\mathbf{H}$ of the separation, with a sparsity $\lambda = 100$.The Denoising SNMF was responsible of enhancing the SDR by applying a denoising on the mask matrix $\mathbf{H}$ generated by the test SNMF. The number of nodes in each hidden layer were 40-20-10-20-40 with 5 hidden layers, while the input layer was composed of $k = 72$ neurons, equal to the total number of SNMF's components. For the activation function, we used the Rectified Linear Unit (ReLU) [27] at each layer including the output layer because it respects the non negativity constraint of the problem, and is faster to compute than the sigmoid function, while saturating less. For the initialization, the autoencoder was initialized using He Initialization [36], we used 200 epochs for the training of the DAE with a batch size of $m = 10$ and a learning rate of $\eta = 10^3$, the model takes around 50 seconds to enhance the separation. Performance measurements of the separation algorithm were done using the signal to distorition radio (SDR) from the BSS Eval Matlab toolbox [64].

Algorithm ($\lambda = 100$)	Input SMR (db)	SDR Speech (db)	SDR Music (db)
Sparse NMF	-5	1.6278	12.253
	0	6.1960	3.5360
	5	10.487	-6.1668
Denoising SNMF	-5	**4.0737**	**13.033**
	0	**8.102**	**4.26**
	5	**12.49**	-5.71

Table 4.6: SDR in Db for estimated speech and music signal

The results presented in Table 4.6 represent the SDR for both estimated speech and music signal, we can notice a great improvement in SDR, while performing better than regular and sparse trained NMF, for all input speech-to-music ratio (SMR) values from -5 to 0 to 5 dB. The improvement in the SDR is usually between 1 - 3 dB over SNMF using optimal filters, and as high as 6 dB compared to regular SNMF. Furthermore, from figure 4.15, we can notice that the SDR Ratio of speech and music increased over the decreasing of the loss, in other word, we can say that the model learned to separate the mixture.

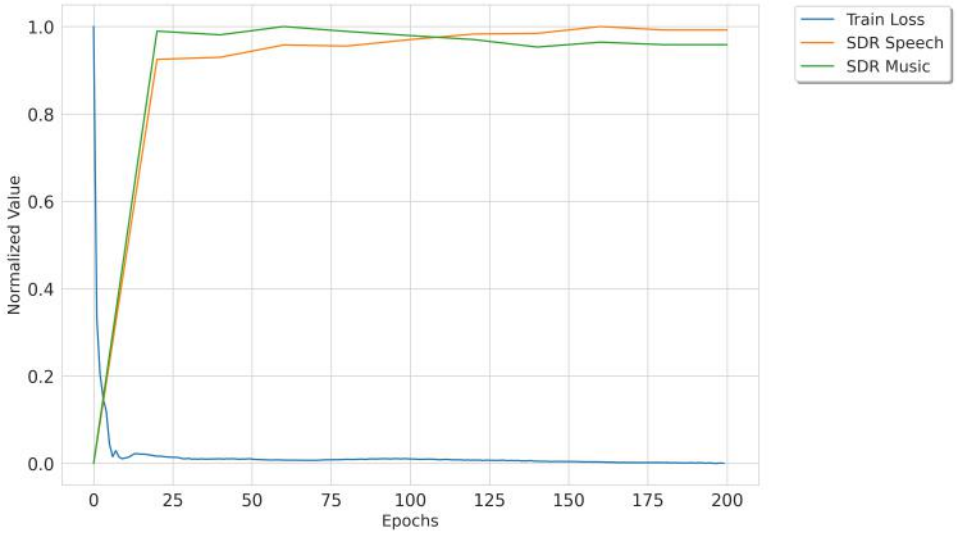
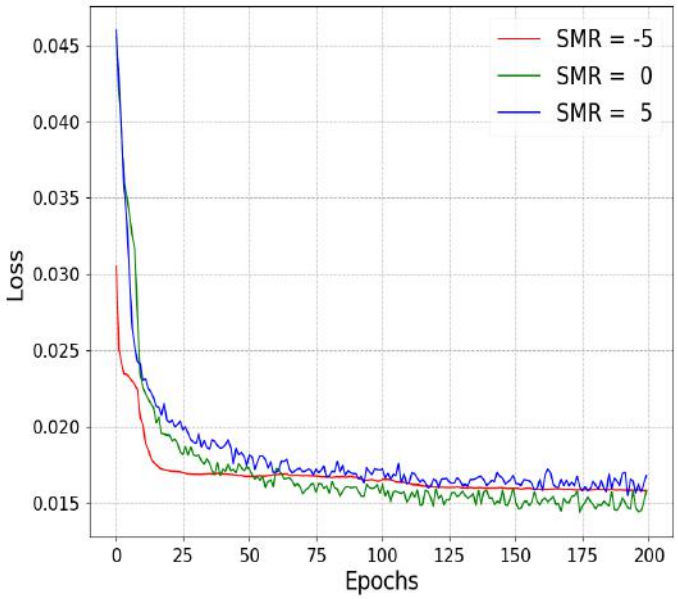
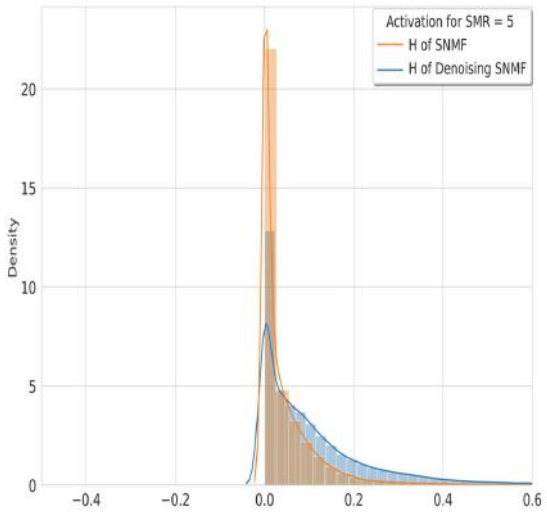


Figure 4.15: Normalized Speech and Music SDR evolution over training loss for SMR = 5

On the other hand, From Figure 4.16a, we can observe that for every SMR, the reconstruction loss is decreasing, meaning that the autoencoder is converging. Furthermore, from Figure 4.16b, we can notice that the distribution of the input and the reconstruction are different. While the input distribution is zero mean centered, the reconstructed input has less density around 0, allowing to have a much more interesting distribution of values by its variance, we can say that our architecture didn't reconstruct perfectly the input.



(a) Learning Curve over different SMR



(b) Distribution of the activation H before and after the inference of the Denoising SNMF

Figure 4.16: Learning Curves and weights' distributions using the denoising SNMF.

In fact, an autoencoder that reconstructs its inputs perfectly does not necessarily mean that it is a good autoencoder, it is usually an over-complete autoencoder that has learned to copy its inputs to the encoding layer. In other words, it would perfectly reconstruct its inputs without actually learning any useful patterns in the data. This illustrates that a perfect reconstruction does not guarantee that the autoencoder has learned something of interest. On the other hand, if it produces very bad reconstruction, then it is undoubtedly a bad autoencoder.

Chapter 5

Deep Unfolding Based Source Separation

5.1 Introduction to Deep Recurrent NMF

The second approach in our project is a method based on the Deep Unfolding techniques also known as unrolling algorithms.

The speech separation using Deep Recurrent NMF (DR-NMF) is performed in 3 essential steps that are presented in figure 5.1:

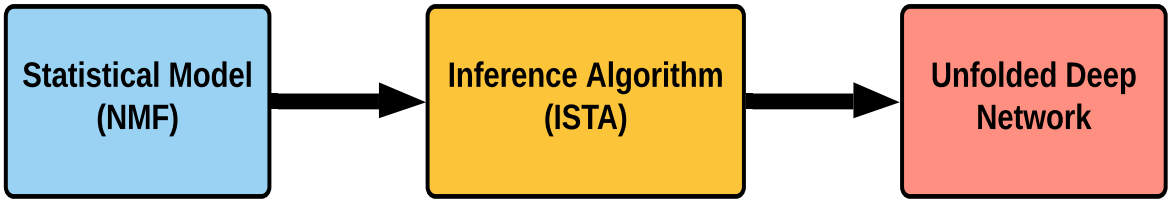


Figure 5.1: Deep Recurent NMF Steps

The constructed recurrent Neural Network is a result of unfolding the inference algorithm ISTA Of the Non negative Matrix Factorization. This method shows a better performance than state of the art deep learning methods like LSTMs in generalisation and therefore DR-NMF is less sensitive to overfitting and requires less data in the training process. Reducing the training data will help at reducing the hardware requirements, time and cost of the training. Besides, building the neural network from unfolding algorithm is highly interpretable compared to a standard deep learning neural network that is considered as a black box.

5.2 Statistical Model

The Sparse Non negative Matrix Factorization [12] was used to decompose the magnitude spectrum into two non negative matrices $\boldsymbol{W}$ and $\boldsymbol{H}$ like it was done in previous sections.
Authors in [6] used the Frobenius Norm as an objective function, which is equivalent to fix $\beta = 2$, minimized using MU optimizer described in subsection 3.1.5. They also trained the dictionnaries differently from the Trained NMF we introduced in section 3.1, as it is illustrated in Figure 5.2

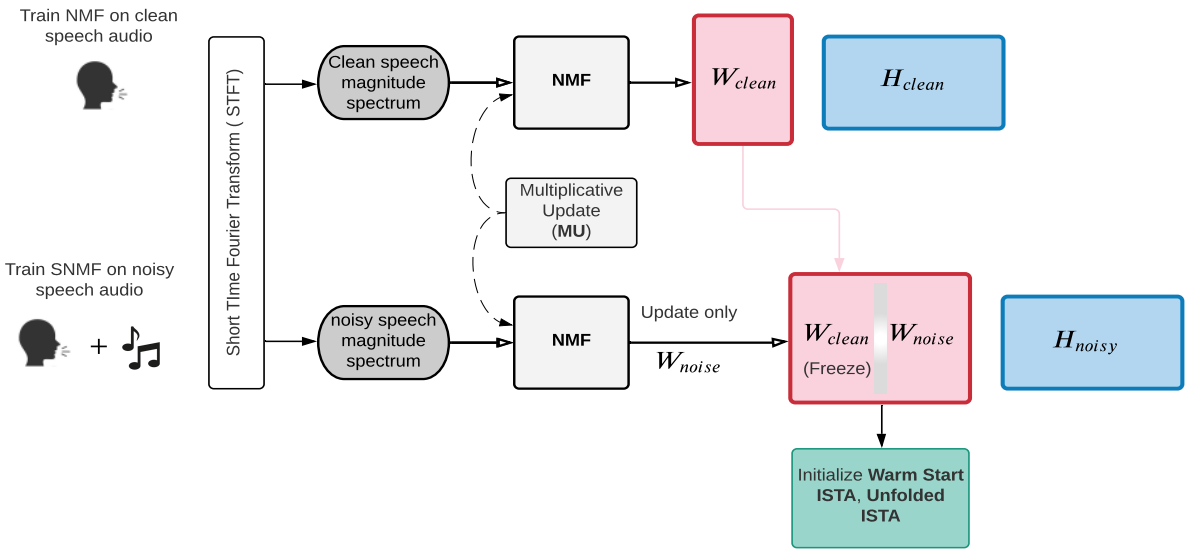


Figure 5.2: SNMF Training pipeline to initialize WS-ISTA, and DR-NMF

The overall training procedure expressed by the Figure 5.2 above is described by the following steps:

- Train clean speech dictionary $\boldsymbol{W}_{clean}$ on clean speech signal using sparse NMF optimized by Multiplicative Update (MU).
- Train the overall dictionary $\boldsymbol{W} = [\boldsymbol{W}_{clean}, \boldsymbol{W}_{noise}]$ on noisy speech signal, updating only the noisy dictionary $\boldsymbol{W}_{noise}$ while keeping clean dictionary $\boldsymbol{W}_{clean}$ freezed.
- Use $\boldsymbol{W}$ to initialize weights of the Warm Start ISTA, and DR-NMF.

Authors in [6] emphasised on extracting speech from the noise only, without carrying about the latter. Plus, we can say their pipeline is suited to situations where we don't have access to dictionary trained directly on clean music, or other type of considered noise, making their approach convenient to speech enhancement. Indeed, they have trained their SNMF on the clean speech signal then the mixture.

On the other hand, we focused our study on the separation of the speech from the music source as well as extracting the music from the mixture, enhancing then the estimated spectrum of each source. Furthermore, training NMF on clean music signal, allows to have a prior knowledge on the music schema, expressed by its trained dictionary $\boldsymbol{W}_{music}$. Moreover, we used the IS-Divergence 3.1.1 that underlies previous work in the area of automatic music transcription and single-channel audio source separation, thanks to its scale-invariant property.

5.3 Inference Algorithm

The optimization problem we need to solve is the minimization of a cost function that contains a smooth term and a non-smooth term (LASSO).

LASSO [65] stands for Least Absolute Shrinkage and Selection Operator. It adds penalty term to the cost function (regularization term). This term is the absolute sum of the coefficients of the model. Higher values of coefficients penalize the cost function, causing the model to decrease the values.

$$\boxed{\underset{h\geq 0}{argmin}\ D_\beta(X||W\cdot H)+\lambda\cdot\|H\|_1} \tag{5.1}$$

W : Dictionary matrix (Basis vectors).
H : Gain matrix.
$D_\beta(X||W\cdot H)$: β- Divergence loss between the signal spectrogram X and the reconstruction of its non-negative decomposition $\hat{X}=W\cdot H$ across time and frequency.

$$D_\beta(X||W\cdot H)=\sum_{t,f}D_\beta(X_{t,f}||\hat{X}_{t,f}) \tag{5.2}$$

Authors in [6] use the β-Divergence function with $\beta=2$ that represents the summed square error between X and $\hat{X}$. However, other values of β that corresponds to other losses like Kullback–Leibler and Itakura–Saito can be considered.

For $\beta=2$, the objective function takes the following form :

$$\boxed{\underset{h\geq 0}{argmin}\ \frac{1}{2}\cdot\|X-W\cdot H\|^2+\lambda\cdot\|H\|_1} \tag{5.3}$$

5.3.1 Iterative soft-thresholding algorithm (ISTA)

To minimize the objective function 5.3 we used the Iterative soft-thresholding algorithm (ISTA) instead of the usual Multiplicative Update (MU) algorithm 3.1.5. The motivation behind this choice is that the multiplicative updates algorithm is slower than ISTA and the expressions of the multiplicative update are quite challenging for the backpropagation of the unfolded model.

To assure the non-negativity of the gains matrix **H** the $soft$ function that has the following expression was used in the original paper [6] :

$$\text{soft}_{\lambda/\alpha}(z_i)=\frac{z_i}{|z_i|}\cdot\text{ReLU}(z_i-\frac{\lambda}{\alpha},0) \tag{5.4}$$

Algorithm 5: Iterative soft-thresholding algorithm (ISTA).

Input: Dictionary $\mathbf{W}$, Spectrogram $\mathbf{X}$, Initial gains $\mathbf{H}^{(0)}$
Input: Step size $\frac{1}{\alpha}$, sparsity parameter λ, max ISTA iterations K
for $k = 1 : K$ **do**
  $\mathbf{Z} \leftarrow \left(\mathbf{I} - \frac{1}{\alpha}\mathbf{W}^T\mathbf{W}\right)\mathbf{H}^{(k-1)} + \frac{1}{\alpha}\mathbf{W}^T\mathbf{X}$
  $\mathbf{H}^{(k)} \leftarrow \text{soft}_{\lambda/\alpha}(\mathbf{Z})$
end
return $\mathbf{H}^{(K)}$

However, we noticed that the quotient $\frac{z_i}{|z_i|}$ will always have a magnitude of 1 and a phase 0 or π which makes a part of the gains negative. To avoid this problem we omitted this quotient. We have also replaced the threshold parameter $\frac{\lambda}{\alpha}$ that controls the sparsity of the gains $\mathbf{H}$ with the sparsity parameter λ. This allows us to link the sparsity to a single parameter λ instead of two parameters. The expression of the soft function for each element of $\mathbf{z}$ becomes:

$$\boxed{\text{soft}_\lambda(z_i) = \text{ReLU}(z_i - \lambda, 0)} \tag{5.5}$$

One of the limits of ISTA is that it does not consider the temporal dependencies between the different frames. ISTA runs independently on each time frame, meaning for each column j of the mixed spectrogram $\mathbf{X}$, ISTA computes the column j of the gains matrix without taking into account the other adjacent columns that have a potential correlation with the column j. To deal with this, a slightly modified version of ISTA was proposed in [6].

5.3.2 Warm Start ISTA

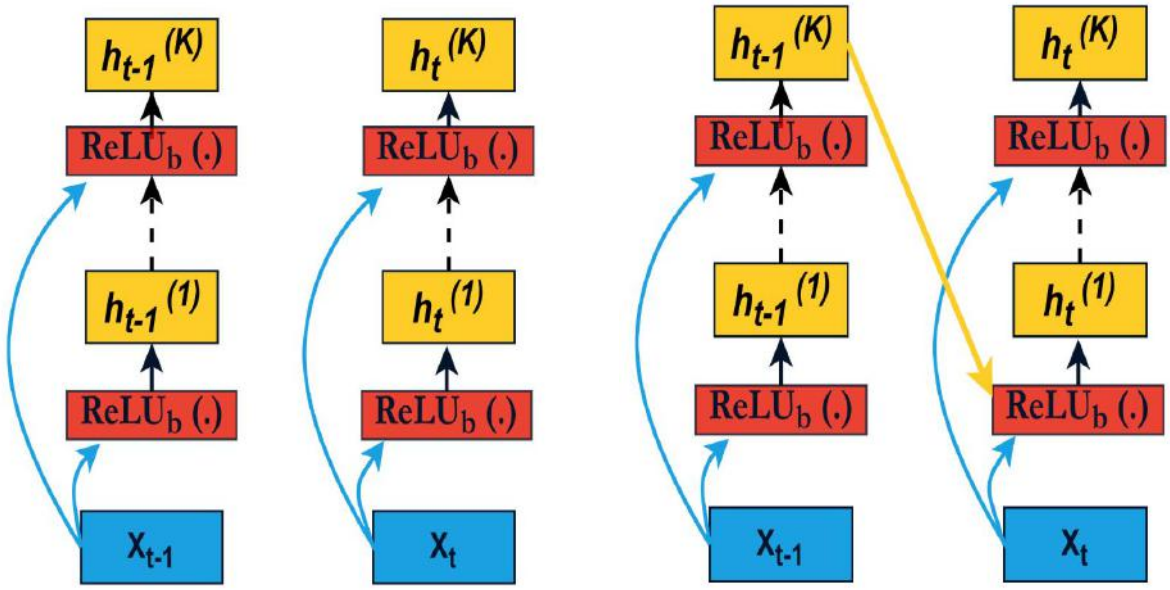
Warm Start ISTA is a version of the ISTA algorithm that solves the NMF problem without neglecting the temporal dependencies between the frames. It takes advantage of a "Warm Start Initialization" that consists of initializing the gain $\mathbf{h}$ at timestamp t and iteration $k = 0$ with the gain of the last iteration K of the previous frame. This provides a good initialization of the gains at each frame.

$$\boxed{\mathbf{h}_t^{(0)} = \mathbf{h}_{t-1}^{(K)}} \tag{5.6}$$

Algorithm 6: Warm Start Iterative soft-thresholding algorithm (WSISTA).

Input: Dictionary $\mathbf{W}$, Spectrogram observation of $\mathbf{X}$: $x_{1:T}$, initial gains $\mathbf{h}_0^{(k)}$
Input: Step size $\frac{1}{\alpha}$, sparsity parameter λ, max WSISTA iterations K.
for $t = 1: T$ **do**
  $\mathbf{h}^{(0)} \leftarrow \mathbf{h}_{t-1}^{(K)}$
  for $k = 1 : K$ **do**
    $\mathbf{z} \leftarrow \left(\mathbf{I} - \frac{1}{\alpha}\mathbf{W}^T\mathbf{W}\right)\mathbf{h}_t^{(k-1)} + \frac{1}{\alpha}\mathbf{W}^T\mathbf{x}_t$
    $\mathbf{h}_t^{(k)} \leftarrow \text{soft}_{\lambda/\alpha}(\mathbf{z})$
  end
end
return $\mathbf{h}^{(K)}$

Figure 5.3 illustrates the difference between the two algorithms using their time-unrolled graph.



(a) ISTA **(b)** Warm Start ISTA

Figure 5.3: Time-unrolled graph of ISTA and Warm Start ISTA.

5.3.3 Warm Start ISTA with memory

Warm start ISTA considers only the gain $\mathbf{h}^{(K)}$ of the previous frame. We propose in our work an improved version that takes advantage of memory to compute the gains as it is shown in Figure 5.4.

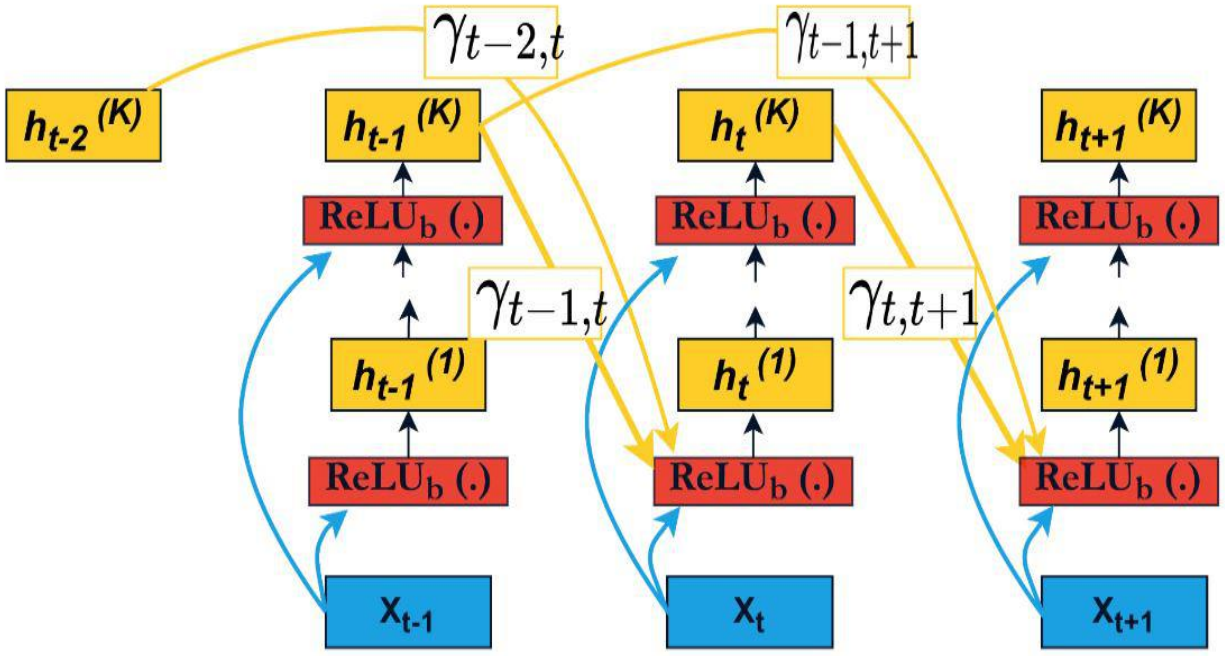


Figure 5.4: Time-unrerolled graph of Warm Start ISTA with memory

The memory consists of a weighted moving average over the last N frames.

$$\boxed{\mathbf{h}_t^{(0)} = \frac{\sum_{i=1}^{N} \gamma_{t-i,t} \cdot \mathbf{h}_{t-i}^{(K)}}{\sum_{i=1}^{N} \gamma_{t-i,t}}} \tag{5.7}$$

$\gamma_{t-i,t}$: Weights of the moving average used to initialize gain h_t.

Adding memory makes the algorithm takes full advantage of the temporal dependencies and gives a better initial estimate of the gains at each frame. It also gives a smooth estimate of $\mathbf{h}$ since it averages over the past frames.

Optimal Mask Feedback

We have seen earlier in chapter 3 that the SDR of the reconstruction is highly dependent on the value of the parameter p used in the generalized Wiener Filter. Moreover, we noticed that the optimal value of p depends on the SMR. To solve this problem, we propose an adaptive value of p that depends on the SMR of the reconstructed signal.

1. Apply NMF using Warm start ISTA with memory.
2. Reconstruct the estimated speech and music signal with a default value $p = 5$ which is the optimal value for an SMR equal to zero.
3. Compute the SMR of the estimated speech and music signals and map it to the corresponding input SMR. There is a linear relationship between the two SMRs according to the graph shown in Figure 5.5
4. Automatic selection of the new value of p according to the mapped SMR.

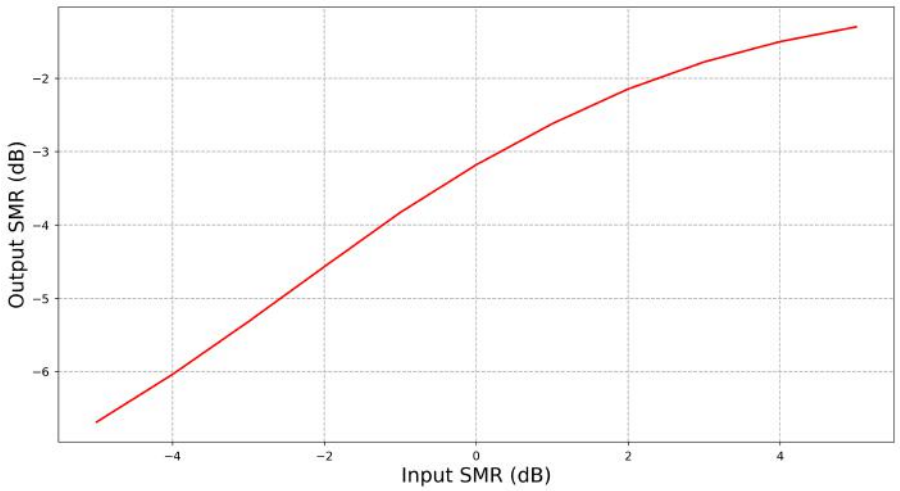


Figure 5.5: Map between input signals SMR and estimated signals SMR.

5.3.4 Results

We present in Table 5.2 the results of each improvement we made on the original algorithm described in this section and its resulting Signal-to-Distortion Ratio (SDR). We can observe that the memory term improves more the results of the music because it has more temporal dependencies than the speech. Moreover, using the automatic optimal mask increases significantly the results for both speech and music. We demonstrated in chapter 3 the effect of the mask on the results of the NMF and the need to have an automatic method to choose the parameter p of the

mask. This solution we provided here can be considered to solve the problem of choosing the mask manually or fixing its value and sacrificing the results.

Algorithm	SMR Input (db)	SDR Speech (db)	SDR Music (db)
Warm Start ISTA	−5	0.41	11.19
	0	5.7210	1.7425
	5	10.436	−9.3511
Warm Start ISTA with memory (Our approach)	−5	0.21	11.69
	0	6.5	2.25
	5	8.06	−7.85
Warm Start ISTA with memory and optimal mask (Our approach)	−5	**2.58**	11.70
	0	**6.5**	2.25
	5	**11.2**	−7.94

Table 5.1: SDR in Db for estimated speech and music signal, using Warm start ISTA;

5.4 Deep recurrent NMF

In this section, we explain how we use Deep Unfolding to combine the Warm Start ISTA algorithm with Deep Learning techniques to solve the NMF problem and improve the separation results. As discussed in 1.5, unrolling algorithms helps developing efficient, high-performance and interpretable networks that can be trained on limited datasets. Warm Start ISTA is a perfect example of an iterative algorithm that can be unfolded across its K iterations. The resulting Deep Network have the following characteristics :

- Each iteration of Warm start ISTA is considered as a layer of the network.
- While Warm start ISTA use a **single fixed** pretrained dictionary W (no update of W), DR-NMF network uses the pretrained dictionary W to initialize the weights of the Neural Network, therefore, each layer have its own weights $W^{(k)}$ that can be interpreted as the basis vectors of each iteration. These weights are updated during the backpropagation of the DR-NMF network.
- The activations of the neurons in each layer are the gains of the NMF for a specific warm start ISTA iteration k.
- The number of neurons in each layer is equal to the number of components of the NMF.

- The difference between the Recurrent Neural Network (RNN) and DR-NMF from architecture perspective lays in the presence of the input **x** in all the layers of the DR-NMF and the sequential connections in RNN are within the same layer while the DR-NMF links the activation of the last layer with the input of the first layer. Figure 5.6 illustrates this difference as well as the DR-NMF cell structure.

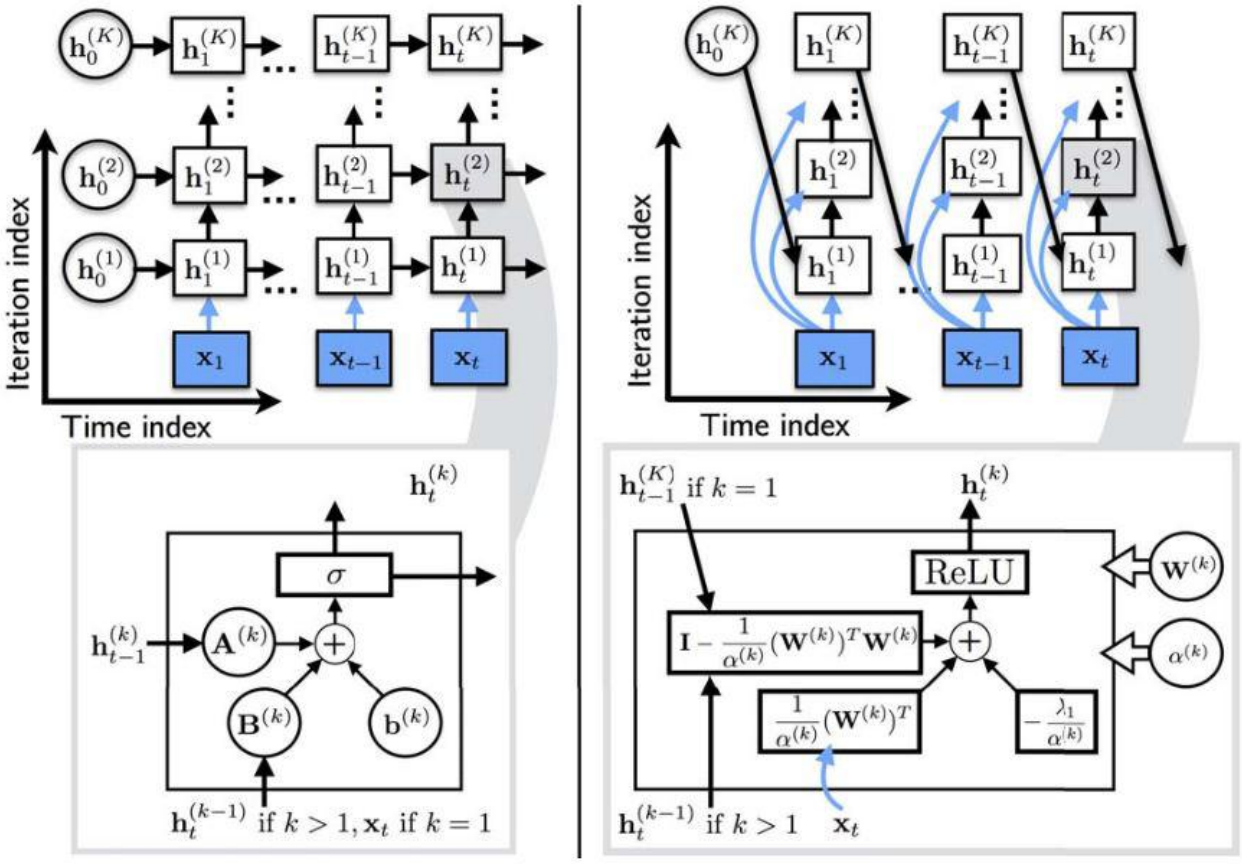


Figure 5.6: DR-NMF architecture unfolded across time (Right) compared to the unfolded architecture of the RNN (Left) [6]

5.4.1 Initialization

DR-NMF weights are initialized using the pretrained dictionaries of the NMF W. In the paper [6] they initialized W with $log(\epsilon + W)$ and instead of optimizing W, they updated $\hat{W}$ and set the model weights as follows :

$$W = exp(\hat{W}) \cdot diag^{-1}(\sqrt{\sum_f exp(\hat{W}_{f,:})^2}) \tag{5.8}$$

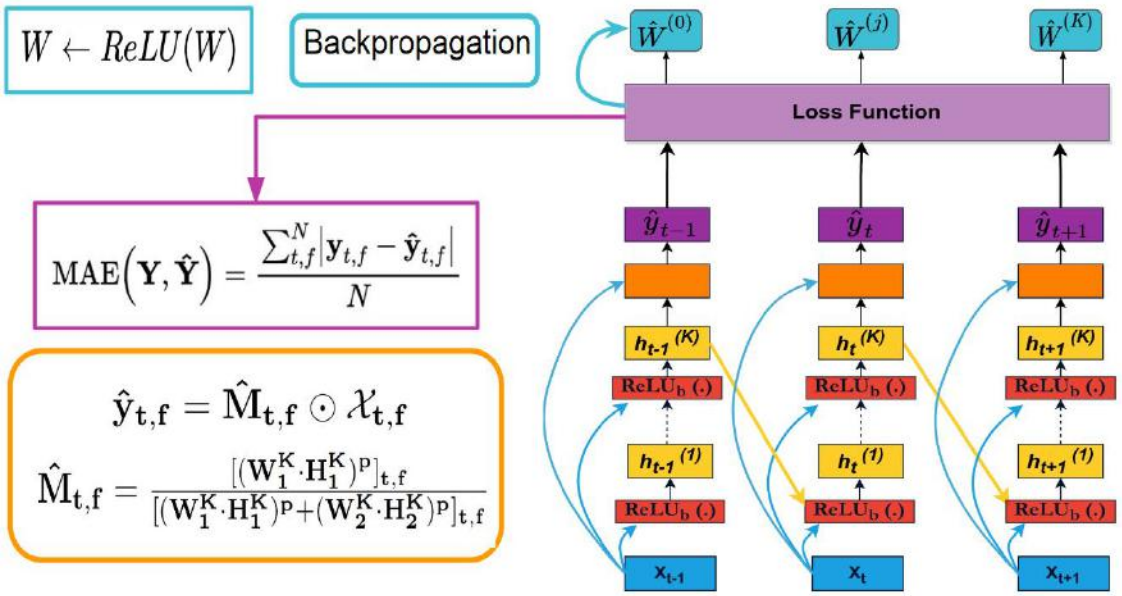


Figure 5.7: Unfolded network and its optimization problem.

This expression is used in order to assure the non-negativity of the weights W after the updates. However, according to our tests on our dataset, using exponential functions produces many NaN values [66]. Besides, the expression 5.8 is computationally expensive and makes the backpropagation slower. To solve the NaN problem and make the model train faster we have used a ReLU function to update the weights.

$$\boxed{W = \mathrm{ReLU}(\hat{W})} \tag{5.9}$$

Therefore, the activation of each layer can be written as :

$$\mathbf{z} \leftarrow \left(\mathbf{I} - \frac{1}{\alpha}\mathrm{ReLU}(\hat{\mathbf{W}})^T \mathrm{ReLU}(\hat{\mathbf{W}})\right) \mathbf{H}^{(k-1)} + \frac{1}{\alpha}\mathrm{ReLU}(\hat{\mathbf{W}})^T \mathbf{x} \tag{5.10a}$$

$$\mathbf{H}^{(k)} \leftarrow \mathrm{ReLU}(\mathbf{z}) \tag{5.10b}$$

The unfolded network can be summarized in Figure. 5.7:

5.4.2 Loss Function

The objective function used by [6] was the *Mean Squared Error* applied between the true clean spectrum $\boldsymbol{Y}$ and the estimated clean spectrum $\tilde{\boldsymbol{M}} \odot \boldsymbol{X}$.

$$MSE\left(\mathbf{Y}, q_\theta(\mathbf{X})\right) = \frac{\sum_{f,t}^{n} \left(\mathbf{Y}_{f,t} - \hat{\mathbf{M}}_{f,t}\mathbf{X}_{f,t}\right)^2}{n} \tag{5.11}$$

The minimization of the MSE aims to maximize the signal-to-noise ratio SNR of magnitude spectra in the time-frequency-domain. Furthermore, the MSE emphasized to back-propagate gradients of larger error much more than small error because of the square operation, which means that the larger error contribute much more than smaller error that are mitigated.

In our study, we have used the *Mean Absolute Error* (MAE) in order to consider both small and large error in the minimization process, because we consider that trained dictionaries $\boldsymbol{W} = [\boldsymbol{W}_{speech}, \boldsymbol{W}_{music}]$ have already reached a satisfactory minima with respect to the NMF decomposition, making smaller error as important as larger error, which allows smaller gradient as well as larger gradient to flow in the back-propagation.

$$MAE\left(\mathbf{Y}, q_\theta(\mathbf{X})\right) = \frac{\sum_{f,t}^{n} \left|\mathbf{Y}_{f,t} - \hat{\mathbf{M}}_{f,t}\mathbf{X}_{f,t}\right|}{n} \tag{5.12}$$

5.5 Results and Discussion

We applied the Deep Recurrent NMF to separate speech and music from the mixture signal. The proposed method was applied to a collection of speech and piano data sampled at 16kHz frequency. For speech data, we used male and female speech conversation data from an AudioBook [61]. For music data, we downloaded piano music from Youtube [62].

The basis vectors (weights of the unfolded network) are initialized with the pretrained dictionary of the NMF W that consists of 16 and 48 components to represent the speech and the music sources respectively. On the other hand, we propose to use the Nonnegative Double Singular Value Decomposition **NNDSVD** [63] for the initialization of the first gain h_0 instead of making this gain a learnable parameter as they did in the paper [6].

The unfolded network is trained on 5 min of data by updating its weights (basis vectors). The model is then tested on 30 seconds of mixed signal spectrum. During test time, the weights of the model $\mathbf{W}$ are fixed, and only the activations (gains) $\mathbf{H}$ are updated just like the regular NMF.

Two baseline methods are compared to DR-NMF: SNMF using multiplicative update (MU) with 200 iterations, which is used to initialize the DR-NMF networks, and Warm Start ISTA.

The DR-NMF networks achieve competitive separation performance as shown in table 5.2 compared to Sparse-NMF as well as DE-SNMF and Denoising NMF while keeping the interpretability of the model possible and very fast execution time. The total number of layers used in DR-NMF was equal to $K = 5$, we used a step size $\frac{1}{\alpha} = 0.02$, and a learning rate of $\eta = 0.005$ using Stochastic Gradient Descent (SGD) that minimized the Mean Absolute Error (MAE) between the clean speech spectrum and the reconstructed speech . In order to constraint both matrices $\boldsymbol{W}$ and $\boldsymbol{H}$ to be positive, we used the Rectified Linear Unit (ReLU) [27] at each layer k, while adding non-linearity to the model. We also used a mask value $p = 2$ to reconstruct the estimated signals.

Performance measurements of the separation algorithm were done using the signal to distortion radio (SDR) from the BSS Eval Matlab toolbox [64].

Algorithm	Input SMR (db)	SDR Speech (db)	SDR Music (db)
Trained Sparse NMF	-5	1.63	12.25
	0	6.22	3.54
	5	11.80	-7.76
Warm Start ISTA	-5	0.41	11.19
	0	5.7210	1.7425
	5	10.436	-9.3511
Deep Unfolded ISTA (dev)	-5	**4.3166**	**13.822**
	0	**8.1059**	**7.05**
	5	**13.523**	**2.15**
Deep Unfolded ISTA (test)	-5	**3.06**	**12.73**
	0	**6.44**	**4.03**
	5	**12.45**	**1.08**

Table 5.2: SDR in Db for estimated speech and music signal, using DR-NMF

It can be noticed that DR-NMF outperforms Sparse NMF, for the three values of SMR = $(-5, 0, 5)$, with a particular enhancement in the music estimated spectrum, with an improvement reaching 10dB for SMR = 5, 3.5dB for SMR = 0, and minimum of 1.5dB for SMR = -5. On the other hand, the improvement of the speech signal is also observed, with a maximum of 3dB for SMR = 5, 2dB for SMR = 0, and a minimum of 1.7dB for SMR = -5.

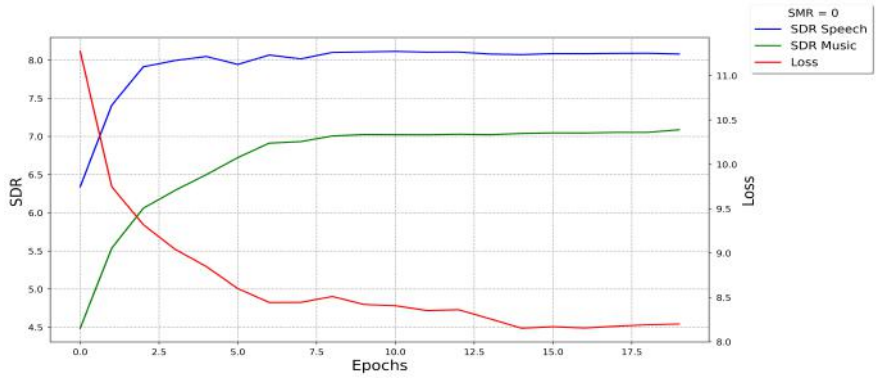


Figure 5.8: Learning Curve and SDR speech and music evolution over DR-NMF training for SMR=0.

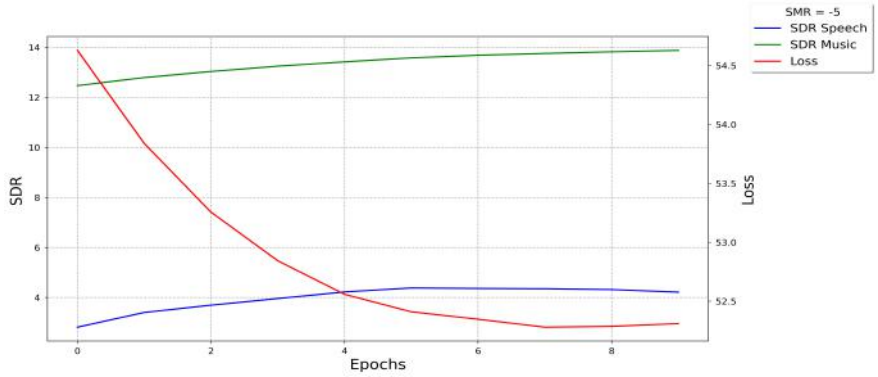


Figure 5.9: Learning Curve and SDR speech and music evolution over DR-NMF training for SMR=-5.

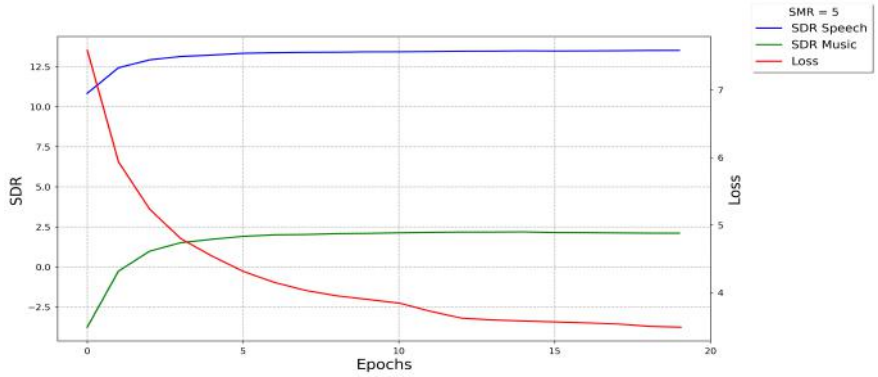


Figure 5.10: Learning Curve and SDR speech and music evolution over DR-NMF training for SMR=5.

From Figures 5.8, 5.9 and 5.10, we can notice that while the objective function decreases, the SDR of both the estimated speech and music signal increases. Plus, we can notice that the DR-NMF is faster than other methods, needing less iterations with a minimum of 10 to 20 epochs to converge toward a solution while enhancing speech and music signal better than SNMF.

Finally, the inference using the unfolded network consists of a forward propagation of the mixed spectrum across the network. This process takes around 50 ms for estimating 1 second, which is quite faster than using the first approach that we described in chapter 4, that takes up to 5 seconds for a mixed signal with a duration of 1 second. The fast execution time of the unfolded network makes it a great choice for real time applications in source separation.

Conclusion

The research community witnessed a great success of the non-negative matrix factorization in a wide variety of source separation applications especially audio signal processing. However, deep neural networks (DNN) have demonstrated more promising performances in innumerable applications of computer vision, pattern recognition, and speech processing at the expense of interpretability. The study presented in this thesis puts the light on two different paradigms that combine the model based methods like the non-negative matrix factorization and deep learning methods : The sequential stack of the two methods and the deep unfolding.

The first approach fuses the NMF and a deep neural network in two sequential stages where the DNN plays the role of a separation enhancer that updates the spectrograms that were estimated using the NMF. We presented in chapter 4 two instances of this approach : Deep Enhanced Sparse NMF and Denoising Sparse NMF. Both methods rely on the autoencoders (AE), but have two different perspectives, while the DE-SNMF uses AE as a feature extractor to classify the sources and acts directly on the sources spectrum, Denoising-SNMF takes advantage of a denoising autoencoder to enhance the gains estimated using the SNMF. The results we obtained using our two methods are quite promising and demonstrated a great ability of improving the spectra estimates of the SNMF by reducing the distortions in their different forms (interference, artifacts). Besides, the two methods have outperformed the work presented in [16] that used the same sequential two stages method, in terms of SDR improvement with respect to the NMF.

The second approach based on the deep unfolding paradigm has revealed the capacity to outperform state of the art deep learning methods such LSTMs [6]. The original method was proposed to extract only a single speech signal from the mixture. In our thesis, we propose modifications to adapt this method to multi-source separation and extraction (speech and music) and we improved the original architecture by changing the expressions of the network's weights to give more stability to the algorithm (avoid divergence) in our application. The experimental results show that this method improves significantly the separation for both speech and music compared to the Sparse NMF. In fact, the signal to distortions ratio of the SNMF spectra estimates has increased by a value up 2.68 dB for the speech signal and up to 9.91 dB for the music signal depending on the SMR value.

While both approaches led to outstanding results, the unfolded SNMF was able to estimate the sources spectra with lower levels of distortions and its execution time is far less than the first sequential approach making it a great choice for real time applications such as hearing aids. The fast execution of this method can be attributed to the vectorization used in the feed forward propagation through the network that takes less time than using the original iterative algorithm ISTA.

We also propose an extension to the Warm Start ISTA to improve the initialization of the gains at each frame by taking a weighted average of the previous gains which forms a memory term that provides a smooth estimate of the gains. Furthermore, It allows to take advantage of the temporal dependency of the spectra. This contribution helped the Warm Start ISTA algorithm to gain up to 2 dB in terms of SDR.

Finally, we should mention that the proposed approaches have been implemented using Pytorch V1.9.0, and the codes have been made available in Github repositories [67] and [68].

Future works The implemented models presented in this thesis demonstrated encouraging results. This work can be the object of various improvements in both the time-frequency representation of the signals and the separation techniques.

First, other types of time-frequency (TF) representations that use a fixed, nonlinear frequency scale can be considered, because the amplitude of natural sounds in high frequencies and low frequencies varies differently, an example of such representations are the Mel scale and the equivalent rectangular bandwidth scale.

Next, variants of the separation's algorithm can be examined, both about the model-based method such as convolutive NMF, and the deep learning method such as LSTM, RNN, CNN based autoencoders that capture the temporal dependencies of the input. We can also cite self-attention mechanism that weighing the significance of each part of the input data providing context to each point. It should be noted that, this mechanism encounters a great success in the field of natural language processing (NLP) and computer vision, which can be then very promising in the field of source separation.

Finally, the concept of memory can be applied to the unrolling algorithm, by considering its weights as learnable parameters of the unfolded network.

Bibliography

[1] World Health Organization (WHO), *Audio pollution*. [Online]. Available: https://www.euro.who.int/en/health-topics/environment-and-health/noise/data-and-statistics.

[2] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, vol. 521, pp. 436–44, May 2015. DOI: 10.1038/nature14539.

[3] F. Schroff, D. Kalenichenko, and J. Philbin, "Facenet: A unified embedding for face recognition and clustering," *CoRR*, vol. abs/1503.03832, 2015. arXiv: 1503.03832. [Online]. Available: http://arxiv.org/abs/1503.03832.

[4] H. Li, "Deep learning for natural language processing: Advantages and challenges," *National Science Review*, vol. 5, pp. 24–26, Jan. 2018. DOI: 10.1093/nsr/nwx110.

[5] "Comparison of subjective assessment and precise quantitative assessment of lesion distribution in diabetic retinopathy," DOI: doi:10.1001/jamaophthalmol.2018.0070.

[6] S. Wisdom, T. Powers, J. W. Pitton, and L. Atlas, "Deep recurrent NMF for speech separation by unfolding iterative thresholding," *CoRR*, vol. abs/1709.07124, 2017. arXiv: 1709.07124. [Online]. Available: http://arxiv.org/abs/1709.07124.

[7] K. Gregor and Y. Lecun, "Learning fast approximations of sparse coding," Aug. 2010.

[8] J. M. Kevin J.P Woods, "Schema learning for the cocktail party problem," [Online]. Available: https://doi.org/10.1073/pnas.18016141151.

[9] "Auditory cortex," [Online]. Available: https://en.wikipedia.org/wiki/Auditory_cortex.

[10] T. Kristjansson, H. Attias, and J. Hershey, "Single microphone source separation using high resolution signal reconstruction," in *2004 IEEE International Conference on Acoustics, Speech, and Signal Processing*, vol. 2, 2004, pp. ii–817. DOI: 10.1109/ICASSP.2004.1326383.

[11] T. Virtanen, "Speech recognition using factorial hidden markov models for separation in the feature space.," Jan. 2006.

[12] P. O. Hoyer, "Non-negative matrix factorization with sparseness constraints," *CoRR*, vol. cs.LG/0408058, 2004. [Online]. Available: http://arxiv.org/abs/cs.LG/0408058.

[13] M. Schmidt and R. Olsson, "Single-channel speech separation using sparse non-negative matrix factorization," Jan. 2006.

[14] C. Févotte, N. Bertin, and J.-L. Durrieu, "Nonnegative matrix factorization with the itakura-saito divergence: With application to music analysis," *Neural Computation*, vol. 21, no. 3, pp. 793–830, 2009. DOI: 10.1162/neco.2008.04-08-771.

[15] P. D. O'Grady and B. A. Pearlmutter, "Convolutive non-negative matrix factorisation with a sparseness constraint," in *2006 16th IEEE Signal Processing Society Workshop on Machine Learning for Signal Processing*, 2006, pp. 427–432. DOI: 10.1109/MLSP.2006.275588.

[16] E. M. Grais, M. U. Sen, and H. Erdogan, "Deep neural networks for single channel source separation," *CoRR*, vol. abs/1311.2746, 2013. arXiv: 1311.2746. [Online]. Available: http://arxiv.org/abs/1311.2746.

[17] F. Weninger, J. Hershey, J. Le Roux, and B. Schuller, "Discriminatively trained recurrent neural networks for single-channel speech separation," *2014 IEEE Global Conference on Signal and Information Processing, GlobalSIP 2014*, pp. 577–581, Feb. 2015. DOI: 10.1109/GlobalSIP.2014.7032183.

[18] F. Weninger, H. Erdogan, S. Watanabe, E. Vincent, J. Le Roux, J. Hershey, and B. Schuller, "Speech enhancement with lstm recurrent neural networks and its application to noise-robust asr," vol. 9237, Aug. 2015, ISBN: 978-3-319-22481-7. DOI: 10.1007/978-3-319-22482-4_11.

[19] A. Belouchrani, *Analyse temps frequence*, 2021.

[20] H. Jeon, Y. Jung, S. Lee, and Y. Jung, "Area-efficient short-time fourier transform processor for time–frequency analysis of non-stationary signals," *Applied Sciences*, vol. 10, p. 7208, Oct. 2020. DOI: 10.3390/app10207208.

[21] T. Lindeberg, "Scale Invariant Feature Transform," *Scholarpedia*, vol. 7, no. 5, p. 10 491, 2012, revision #153939. DOI: 10.4249/scholarpedia.10491.

[22] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning Representations by Back-propagating Errors," *Nature*, vol. 323, no. 6088, pp. 533–536, 1986. DOI: 10.1038/323533a0. [Online]. Available: http://www.nature.com/articles/323533a0.

[23] M. D. Zeiler and R. Fergus, "Visualizing and understanding convolutional networks," *CoRR*, vol. abs/1311.2901, 2013. arXiv: 1311.2901. [Online]. Available: http://arxiv.org/abs/1311.2901.

[24] W. S. McCulloch and W. Pitts, "A logical calculus of the ideas immanent in nervous activity," *The Bulletin of Mathematical Biophysics*, vol. 5, no. 4, pp. 115–133, 1943. DOI: 10.1007/bf02478259.

[25] F. Rosenblatt, "The perceptron: A probabilistic model for information storage and organization in the brain," *Psychological Review*, pp. 65–386, 1958.

[26] N. Kumar, "Sigmoid-neuron explained," [Online]. Available: https://towardsdatascience.com/sigmoid-neuron-deep-neural-networks-a4cd35b629d7.

[27] R. Hahnloser, R. Sarpeshkar, M. Mahowald, R. Douglas, and H. Seung, "Digital selection and analogue amplification coexist in a cortex-inspired silicon circuit," *Nature*, vol. 405, pp. 947–51, Jul. 2000. DOI: 10.1038/35016072.

[28] "Perceptron drawbacks," [Online]. Available: http://matlab.izmiran.ru/help/toolbox/nnet/percep11.html.

[29] D. E. Rumelhart and J. L. McClelland, "Learning internal representations by error propagation," in *Parallel Distributed Processing: Explorations in the Microstructure of Cognition: Foundations*. 1987, pp. 318–362.

[30] "Logistic function," [Online]. Available: https://en.wikipedia.org/wiki/Sigmoid_function.

[31] X. Glorot and Y. Bengio, "Understanding the difficulty of training deep feedforward neural networks," in *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, Y. W. Teh and M. Titterington, Eds., ser. Proceedings of Machine Learning Research, vol. 9, Chia Laguna Resort, Sardinia, Italy: PMLR, 2010, pp. 249–256. [Online]. Available: http://proceedings.mlr.press/v9/glorot10a.html.

[32] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: A simple way to prevent neural networks from overfitting," *Journal of Machine Learning Research*, vol. 15, no. 56, pp. 1929–1958, 2014. [Online]. Available: http://jmlr.org/papers/v15/srivastava14a.html.

[33] D. Wilson and T. R. Martinez, "The general inefficiency of batch training for gradient descent learning," *Neural Networks*, vol. 16, no. 10, pp. 1429–1451, 2003, ISSN: 0893-6080. DOI: https://doi.org/10.1016/S0893-6080(03)00138-2. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0893608003001382.

[34] J. Duchi, E. Hazan, and Y. Singer, "Adaptive subgradient methods for online learning and stochastic optimization," *Journal of Machine Learning Research*, vol. 12, pp. 2121–2159, Jul. 2011.

[35] V. Bushaev, "Rmsprop," [Online]. Available: https://towardsdatascience.com/understanding-rmsprop-faster-neural-network-learning-62e116fcf29a.

[36] K. He, X. Zhang, S. Ren, and J. Sun, *Delving deep into rectifiers: Surpassing human-level performance on imagenet classification*, 2015. arXiv: 1502.01852 [cs.CV].

[37] X. Glorot and Y. Bengio, "Understanding the difficulty of training deep feedforward neural networks," in *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, Y. W. Teh and M. Titterington, Eds., ser. Proceedings of Machine Learning Research, vol. 9, Chia Laguna Resort, Sardinia, Italy: PMLR, 2010, pp. 249–256. [Online]. Available: http://proceedings.mlr.press/v9/glorot10a.html.

[38] R. Salakhutdinov, A. Mnih, and G. Hinton, "Restricted boltzmann machines for collaborative filtering," in *Proceedings of the 24th International Conference on Machine Learning*, ser. ICML '07, Corvalis, Oregon, USA: Association for Computing Machinery, 2007, 791–798, ISBN: 9781595937933. DOI: 10.1145/1273496.1273596. [Online]. Available: https://doi.org/10.1145/1273496.1273596.

[39] J. R. Hershey, J. L. Roux, and F. Weninger, "Deep unfolding: Model-based inspiration of novel deep architectures," *CoRR*, vol. abs/1409.2574, 2014. arXiv: 1409.2574. [Online]. Available: http://arxiv.org/abs/1409.2574.

[40] *Numpy*, https://numpy.org/contribute/, [accessed 13-August-2020].

[41] *Scipy*, https://www.scipy.org/, [accessed 13-August-2020].

[42] *Matplotlib*, https://matplotlib.org/, [accessed 13-August-2020].

[43] *Sckitlearn*, https://scikit-learn.org/, [accessed 13-August-2020].

[44] *Pytorch*, https://pytorch.org/.

[45] E. Vincent, R. Gribonval, and C. Fevotte, "Performance measurement in blind audio source separation," *IEEE Transactions on Audio, Speech, and Language Processing*, vol. 14, no. 4, pp. 1462–1469, 2006, ISSN: 1558-7924. DOI: 10.1109/TSA.2005.858005.

[46] A. Basu, I. R. Harris, N. L. Hjort, and M. C. Jones, "Robust and efficient estimation by minimising a density power divergence," *Biometrika*, vol. 85, no. 3, pp. 549–559, 1998, ISSN: 00063444. [Online]. Available: http://www.jstor.org/stable/2337385.

[47] A. Cichocki, R. Zdunek, and S.-i. Amari, "Csiszár's divergences for non-negative matrix factorization: Family of new algorithms," in *Independent Component Analysis and Blind Signal Separation*, J. Rosca, D. Erdogmus, J. C. Príncipe, and S. Haykin, Eds., Berlin, Heidelberg: Springer Berlin Heidelberg, 2006, pp. 32–39, ISBN: 978-3-540-32631-1.

[48] G.-J. Song and M. K.-P. Ng, *Nonnegative low rank matrix approximation for nonnegative matrices*, 2020. arXiv: 1912.06836 [math.OC].

[49] N. Madhu, A. Spriet, S. Jansen, R. Koning, and J. Wouters, "The potential for speech intelligibility improvement using the ideal binary mask and the ideal wiener filter in single channel noise reduction systems: Application to auditory prostheses," *IEEE Transactions on Audio, Speech, and Language Processing*, vol. 21, no. 1, pp. 63–72, 2013. DOI: 10.1109/TASL.2012.2213248.

[50] M. Cooke, P. Green, L. Josifovski, and A. Vizinho, "Robust automatic speech recognition with missing and unreliable acoustic data," *Speech Communication*, vol. 34, no. 3, pp. 267–285, 2001, ISSN: 0167-6393. DOI: https://doi.org/10.1016/S0167-6393(00)00034-0. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0167639300000340.

[51] E. M. Grais, M. U. Sen, and H. Erdogan, *Deep neural networks for single channel source separation*, 2013. arXiv: 1311.2746 [cs.NE].

[52] J. Lim and A. Oppenheim, "Enhancement and bandwidth compression of noisy speech," *Proceedings of the IEEE*, vol. 67, no. 12, pp. 1586–1604, 1979. DOI: 10.1109/PROC.1979.11540.

[53] N. J. Nyunt, Y. Sugiura, and T. Shimamura, "Parametric wiener filter with parameters estimation on image power spectrum sparsity," in *2017 6th International Conference on Informatics, Electronics and Vision 2017 7th International Symposium in Computational Medical and Health Technology (ICIEV-ISCMHT)*, 2017, pp. 1–6. DOI: 10.1109/ICIEV.2017.8338580.

[54] R. A. Chiea, M. H. Costa, and G. Barrault, "New insights on the optimality of parameterized wiener filters for speech enhancement applications," *Speech Communication*, vol. 109, pp. 46–54, 2019, ISSN: 0167-6393. DOI: https://doi.org/10.1016/j.specom.2019.03.005. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0167639318303194.

[55] M. Fontaine, A. Liutkus, L. Girin, and R. Badeau, "Explaining the Parameterized Wiener Filter with Alpha-Stable Processes," in *WASPAA 2017 - IEEE Workshop on Applications of Signal Processing to Audio and Acoustics*, ser. Proc. IEEE Workshop on Applications of Signal Processing to Audio and Acoustics (WASPAA), New Paltz, New York, United States, Oct. 2017. [Online]. Available: https://hal.archives-ouvertes.fr/hal-01548508.

[56] T. Gerkmann and E. Vincent, "Spectral masking and filtering," in *Audio Source Separation and Speech Enhancement*. John Wiley Sons, Ltd, 2018, ch. 5, pp. 65–85, ISBN: 9781119279860. DOI: https://doi.org/10.1002/9781119279860.ch5. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/9781119279860.ch5. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1002/9781119279860.ch5.

[57] M. Schmidt and R. Olsson, "Single-channel speech separation using sparse non-negative matrix factorization," Jan. 2006.

[58] D. P. Kingma and J. Ba, *Adam: A method for stochastic optimization*, 2017. arXiv: 1412.6980 [cs.LG].

[59] S. Ruder, *An overview of gradient descent optimization algorithms*, 2017. arXiv: 1609.04747 [cs.LG].

[60] D. Bank, N. Koenigstein, and R. Giryes, *Autoencoders*, 2021. arXiv: 2003.05991 [cs.LG].

[61] S. L. channel, "Learn business english conversation for the office and workplace," [Online]. Available: https://www.youtube.com/watch?v=k_ofXhe_tEY.

[62] Rousseau, "The most beautiful relaxing piano pieces (vol. 1)," [Online]. Available: https://www.youtube.com/watch?v=WJ3-F02-F_Y.

[63] H. Qiao, "New SVD based initialization strategy for non-negative matrix factorization," *CoRR*, vol. abs/1410.2786, 2014. arXiv: 1410.2786. [Online]. Available: http://arxiv.org/abs/1410.2786.

[64] E. Vincent, R. Gribonval, and C. Fevotte, "Performance measurement in blind audio source separation," *IEEE Transactions on Audio, Speech, and Language Processing*, vol. 14, no. 4, pp. 1462–1469, 2006. DOI: 10.1109/TSA.2005.858005.

[65] R. Tibshirani, "Regression shrinkage and selection via the lasso," *Journal of the Royal Statistical Society. Series B (Methodological)*, vol. 58, no. 1, pp. 267–288, 1996, ISSN: 00359246. [Online]. Available: http://www.jstor.org/stable/2346178.

[66] "Explication of nan values," [Online]. Available: https://en.wikipedia.org/wiki/NaN.

[67] Y. M. Bouaouni and R. Ait Ali Yahia, *Audio source separation enhancement using Deep Enhanced SNMF and Denoising SNMF*, 2021. [Online]. Available: https://github.com/yacinebouaouni/Deep-Learning-Based-Blind-Source-Separation.

[68] R. Ait Ali Yahia and Y. M. Bouaouni, *Deep recurrent NMF for speech separation by unfolding iterative thresholding*, 2021. [Online]. Available: https://github.com/yacinebouaouni/Deep-Unfolded-NMF-for-Speech-Source-Separation.