Friedrich-Alexander-Universität Erlangen-Nürnberg

FRIEDRICH-ALEXANDER
UNIVERSITÄT
ERLANGEN-NÜRNBERG

Master Thesis

DNN-Based Matrix Factorization with Applications to Drum Sound Decomposition

submitted by

Edgar Andrés Suárez Guarnizo

submitted

April 6, 2020

Supervisor / Advisor

Prof. Dr. Meinard Müller
Dr. Christian Dittmar

Michael Krause
Yiğitcan Özer

Reviewers

Prof. Dr. Meinard Müller

AUDIO LABS

International Audio Laboratories Erlangen
A joint institution of the
Friedrich-Alexander-Universität Erlangen-Nürnberg (FAU) and
the Fraunhofer-Institut für Integrierte Schaltungen IIS.

Fraunhofer IIS

Erklärung

Hiermit versichere ich an Eides statt, dass ich die vorliegende Arbeit selbstständig und ohne Benutzung anderer als der angegebenen Hilfsmittel angefertigt habe. Die aus anderen Quellen oder indirekt übernommenen Daten und Konzepte sind unter Angabe der Quelle gekennzeichnet. Die Arbeit wurde bisher weder im In- noch im Ausland in gleicher oder ähnlicher Form in einem Verfahren zur Erlangung eines akademischen Grades vorgelegt.

Erlangen, April 6, 2020

Edgar Andrés Suárez Guarnizo

Acknowledgements

I would like to express my gratitude to those who supported me during this project.

To Prof. Dr. Meinard Müller, the person who conceived the idea of this work and took the first step to make it a reality, thank you for allowing me to work with you in a topic I am so passionate about. Most importantly, thanks for considering my work valuable and for always motivating students to work harder and perform better.

To my supervisor Michael Krause, thank you for your guidance and for sharing with me your knowledge in the field of machine learning and programming. Thanks for being receptive and open to questions and comments, and for being curious and formulate interesting questions, that have lead to interesting results in this work. I wish you a very successful PhD career.

Thanks to my supervisors Christian Dittmar and Yiğitcan Özer making me a part of their group and for their valuable help and recommendations. Your good research has been the foundation of this thesis. I am looking forward to work with you in the future.

Special thanks to the Friedrich Alexander Universität, Fraunhofer IIS and the International Audio Laboratories Erlangen, which have provided me with the necessary resources and knowledge to complete this work. This place has allowed me to meet people with vast experience and friendly attitude, that share my interest for music and engineering. Working in this environment is a privilege and a dream come true.

To my ASC friends, Ahmad, Mohamed, Adela, Andy, Junseong, Sebastian, David, Slavica and Satnam, with whom I have shared the experience of studying to Erlangen. Thanks for your support, for your company, your opinions and for being my teachers as well as my classmates. I am grateful to have met people with such a high level of skill and talent, and who share my enthusiasm for learning and openly discussing ideas no matter how crazy they seem. The doors of my home will always be open for you.

Finally, to my family and to God, now and always. Without you, none of this would have been possible. This work is also yours.

Abstract

Non-negative matrix factorization (NMF) is an iterative optimization algorithm based on update rules that preserve non-negativity. NMF generates a low-rank approximation of a non-negative input matrix by the combination of a small set of non-negative vectors. Due to the absence of negative elements, these vectors act as parts that are put together to approximate the input instead of cancelling each other out. In the context of music processing, NMF has been extensively applied to sound decomposition, a source separation problem where the goal is to retrieve the sound sources present in the magnitude spectrogram of an input sound mixture. By incorporating prior musical knowledge at the initialization stage, one can guide the NMF algorithm to converge to factor matrices with explicit musical meaning.

In recent years, deep neural network (DNN) architectures have also been used for sound decomposition tasks. In particular, the non-negative autoencoder (NAE) also performs a low-rank approximation of its input, indirectly learning essential features of the different sound sources. In NAE models, non-negativity is enforced through the use of rectifier activation functions. Moreover, prior musical knowledge can be integrated in the learning process of NAEs through the use of weight constraints and structured dropout in its hidden layers.

This thesis draws a detailed comparison between various NMF and NAE approaches. By doing so, the aim is to discover the musical meaning behind DNN parameters and outputs, contributing to the development of musically informed DNN configurations. The comparison is made in the context of drum sound decomposition, where the sound sources correspond to drum strokes present in tracks from the publicly available "IDMT-SMT-Drums" dataset. The results show that NMF and NAEs perform similarly, both qualitatively and in terms of the signal-to-distortion ratio (SDR) of the reconstructed sound source audio signals.

Contents

Chapter 1

Introduction

A music recording can be considered a combination of sound sources interacting to express a cohesive, unified musical idea. In music processing, sound decomposition seeks to reverse this process by identifying the behavior of the individual sound sources that make up a recording. Sound decomposition is a particular case of source separation for music information retrieval (MIR) [25]. Sound decomposition is the baseline of multiple MIR tasks such as singing voice extraction [10], audio mosaicing [11] and music transcription [38], among others.

Non-negative matrix factorization (NMF) [18] and non-negative factor deconvolution (NMFD) [30] are two machine learning algorithms that have been extensively used for sound decomposition, being applied to the magnitude spectrogram of music signals [41]. NMF and NMFD are based on two main principles: First, they generate a low-rank approximation of their input by the combination of a small set of non-negative vectors. This forces the algorithm to learn the most important features of the input data in order to minimize the approximation error. Second, they enforce non-negativity, making the vectors act as parts that are put together to approximate the input, instead of cancelling each other out.

Furthermore, it has been shown that incorporating prior musical score information in the initialization stage of NMF and NMFD leads to the learning of musically meaningful features [12] [19]. For example, one may initialize the factor matrices such that one of them corresponds to the approximated frequency content of each sound source, while the other contains the estimated time intervals where each of the sources are active throughout the music piece. This initialization scheme guides the optimization process, making the algorithm build upon the given initial structure to approximate the input matrix.

A good example of the use of score-informed matrix factorization models can be found in the work by Dittmar et.al [9]. This work sets a standard on the evaluation of informed sound decomposition algorithms, by comparing various initialization approaches using the Signal-to-

distortion ratio performance measure (SDR) [37] to evaluate the decomposition quality. All the above in the context of drum sound decomposition using multiple drumset recordings that make up the "IDMT-SMT-Drums" data set [8]. Drum sound decomposition is a particularly challenging task in music processing. The sounds generated by drums and cymbals usually have a broadband, noise-like spectrum, with no identifiable harmonic structure. Additionally, the different elements of the drumset are often played simultaneously. This means the drum sounds are usually overlapped in both time and frequency, which leads to audible cross-talk artifacts in the decomposition results [40].

The wide variety of implementations and the effectiveness of NMF and NMFD, added to the increasing use of deep neural networks (DNN) and deep learning models in MIR [3], has motivated the development of DNN architectures that emulate matrix factorization properties. For instance, Smaragdis et.al [31] and Venkataramani et.al [34] have proposed the use of non-negative autoencoders (NAE) as a DNN-based matrix factorization model. NAEs explicitly incorporate the low-rank approximation and non-negativity characteristics of NMF, and combine them with the flexibility and scalability of DNN models.

These works suggest that NAEs can perform better than their matrix factorization counterparts, but the results in [31] and [34] are mostly focused on the separation of speech mixtures and not in sound decomposition. To account for this, Ewert et.al [13] uses NAEs in a musical context, and also outlines various ways of incorporating score information into the NAE learning process. The results in [13] also suggest a better performance of DNN-based methods, but were obtained using a data set containing only 10 classical piano pieces, and lacks generality.

Consequently, this thesis intends to bring together the NAE models in [31] and [34] and the score-informed configurations in [13] into the drum sound decomposition scenario described in [9], to compare them with the NMF and NMFD algorithms. By doing so, the aim is to discover the musical meaning behind DNN parameters and outputs, in hopes of contributing to the development of musically informed DNN configurations.

1.1 Structure of this Thesis

Chapter 2 formally introduces the concept of NMF and NMFD, and describes the multiplicative update approach to generate non-negative factor matrices. The chapter also illustrates how NMF and NMFD are used for sound decomposition, and how music information from music scores can be introduced.

Chapter 3 introduces the Non-negative autoencoder (NAE), the main DNN architecture used throughout this work. The chapter describes the functionalities of the autoencoder and shows how its configuration relates to NMF. The chapter analyses multiple NAE variants, such as score-

informed models and NAEs using convolutional layers, and compares their sound decomposition results with the NMF and NMFD results in Chapter 2.

Using the methods and results from the two previous chapters, Chapter 4 presents a quantitative performance comparison NMF and DNN approaches, by applying them to the "IDMT-SMT-Drums" data set.

Chapter 5 summarizes the main results of this thesis and outlines future work.

1.2 Main Contributions

The main contributions of this work are as follows:

First, this thesis provides a qualitative analysis of the implementation of NMF and NMFD algorithms in the context of sound decomposition, additionally showing the effect of introducing music score information into the decomposition process.

Second, this thesis evaluates the use of NAEs as a NMF-based DNN architecture, and its potential use in sound decomposition. This work explores different NAE models, and provides a deeper understanding of different methods for introducing music score information, finding common ground with the NMF results of the previous chapter. The implementation of the described NAE architectures, configurations and visualization tools constitute the main technical contribution of this thesis.

Third, both NMF and NAE models are compared in a sound decomposition scenario by computing the SDR (signal-to-distorsion ratio) of the generated source signals, using the "IDMT-SMT-Drums" public data set.

Finally, this work proposes several ways in which the models and methods described can be further extended to tackle more complex tasks.

Chapter 2

Matrix Factorization for Spectral Decomposition

This chapter introduces the reader to the concept of non-negative matrix factorization (NMF) and its use in sound decomposition. Section 2.1 is an overview of the most important theory of NMF, followed by a description of its implementation as an iterative algorithm using multiplicative updates in Section 2.2. Section 2.3 will show the use of NMF in sound decomposition by using two concrete running examples. Section 2.4 describes the non-negative factor deconvolution (NMFD) method, and its implication on the sound decomposition results.

The theory introduced in this chapter closely follows [18] and Section 8.3 of [25].

2.1 Overview

Non-negative matrix factorization [18] is a type of matrix factorization algorithm where a non-negative matrix i.e. a matrix with only real, non-negative elements, is approximated by the product of two non-negative matrices. Formally, given a non-negative matrix $\mathbf{V} \in \mathbb{R}_{\geq 0}^{K \times M}$, NMF aims to find non-negative matrices $\mathbf{W} \in \mathbb{R}_{\geq 0}^{K \times R}$ and $\mathbf{H} \in \mathbb{R}_{\geq 0}^{R \times M}$ such that

$$\mathbf{V} \approx \mathbf{WH}. \tag{2.1}$$

In NMF, each data vector column $\mathbf{v}_m \in \mathbf{V}$ for $m \in [1 : M]$ is approximated by a linear combination of columns in matrix $\mathbf{W}$. This linear combination is defined by a row vector $\mathbf{h}_m \in \mathbf{H}$, hence $\mathbf{v}_m \approx \mathbf{W} \cdot \mathbf{h_m}$. NMF can be seen as the projection of the vectors in $\mathbf{V}$ into a new dimensional space spanned by the vectors in $\mathbf{W}$. The parameter $R \in \mathbb{N}$ corresponds to the *rank* of the approximation. In practice, this parameter is given or deduced from the input data,

depending on the application. In this work it is assumed that $R \ll \min(K, M)$. In this case, the matrix $\widehat{\mathbf{V}} = \mathbf{WH}$ can be seen as a *low-rank approximation* of matrix $\mathbf{V}$.

At first glance, NMF seems rather similar to other matrix factorization methods such as Principal component analysis (PCA) [7]. However, the non-negativity constraint prevents linear combinations of the vectors in matrix $\mathbf{W}$ from canceling each other out. Instead, the vectors can be seen as parts which combine to build an approximation of the original data matrix. For this reason NMF is regarded as a "parts of a whole" approach [17], where the vectors in $\mathbf{W}$ are directly interpreted as *features* of the input data.

The non-negative factor matrices $\mathbf{W}$ and $\mathbf{H}$ are obtained by solving the following optimization problem:

$$\begin{aligned} \min_{\mathbf{W},\mathbf{H}} \quad & D(\mathbf{V}||\mathbf{WH}), \\ \text{subject to} \quad & \mathbf{W} \in \mathbb{R}_{\geq 0}^{K \times R}, \mathbf{H} \in \mathbb{R}_{\geq 0}^{R \times M}, \end{aligned} \tag{2.2}$$

where the cost function $D(\cdot||\cdot) : \mathbb{R}^{(K \times M)^2} \to \mathbb{R}$ is a measure of divergence or similarity between two matrices. In the following, D is a modified version of the Kullback-Leibler divergence (KLD), defined as follows: Let $\mathbf{A}$,$\mathbf{B} \in \mathbb{R}^{K \times M}$ be two matrices with coefficients a_{ij} and b_{ij} for $i \in [1, K]$ and $j \in [1, M]$. The divergence between matrix $\mathbf{A}$ and $\mathbf{B}$, denoted $D(\mathbf{A}||\mathbf{B})$, is defined as:

$$\begin{aligned} D(\mathbf{A}||\mathbf{B}) &:= \sum_{i=1}^{K} \sum_{j=1}^{M} \left(a_{ij} \ln \left(\frac{a_{ij}}{b_{ij}} \right) - a_{ij} + b_{ij} \right), \\ &= \|\mathbf{A} \odot \ln(\mathbf{A} \oslash \mathbf{B}) - \mathbf{A} + \mathbf{B}\|_F^2 . \end{aligned} \tag{2.3}$$

The function $\|\cdot\|_F$ denotes the Frobenius or matrix norm, and $\odot$ and $\oslash$ are element-wise multiplication and division operators, respectively.

The simplicity of the formulation of NMF can be deceiving. Despite choosing a convex cost function and dealing only with inequality constraints, Problem (2.2) is a *non-convex problem* for $\mathbf{W}$ and $\mathbf{H}$ simultaneously. There is no algorithm that guarantees convergence to a global optimum [33], but numerical schemes in continuous optimization may achieve stationary point convergence.

2.2 Multiplicative Updates

Given that Problem (2.2) is non-convex, a simple approach would be to relaxing one of its constraints, generating a *convex relaxation* of the problem. This is done by fixing the values of one of the factor matrices, and use them to optimize the values of the other. For example,

considering matrices $\mathbf{V} \in \mathbb{R}_{\geq 0}^{K \times M}$ and $\mathbf{H} \in \mathbb{R}_{\geq 0}^{R \times M}$ to be fixed, the convex relaxation problem for matrix $\mathbf{W}$ is formulated as follows:

$$\begin{aligned} \min_{\mathbf{W}} \quad & D(\mathbf{V}||\mathbf{WH}), \\ \text{subject to} \quad & \mathbf{W} \in \mathbb{R}_{\geq 0}^{K \times R}. \end{aligned} \tag{2.4}$$

Convex relaxations can be solved using known numerical optimization methods. Nevertheless, to ultimately find optimal values for the original problem, the procedure must be done in two steps: First, optimal values for one of the factor matrices must be calculated; second, the values obtained are used to optimize the other factor matrix. This procedure is repeated until a certain stop criterion is reached e.g. a certain number of iterations. This alternating approach is known as *block coordinate descent* [39], and constitutes a common framework in NMF algorithms.

A common optimization method such as gradient descent can be used for approximating the minimum point of a given convex relaxation. Let $\mathbf{W}^{(\ell)}$ be the values of matrix $\mathbf{W}$ at iteration (ℓ). Gradient descent seeks to find values for $\mathbf{W}^{(\ell+1)}$ such that the cost function decreases. Because the chosen cost function in Equation (2.3) is differentiable, it decreases faster in the direction of the negative gradient, so for $\mathbf{W}^{(\ell+1)}$ the gradient descent update rule can be written as:

$$\begin{aligned} \mathbf{W}^{(\ell+1)} :=& \mathbf{W}^{(\ell)} - \boldsymbol{\gamma}^{(\ell)} \odot \left(\nabla_{\mathbf{W}^{(\ell)}} D\left(\mathbf{V}||\mathbf{W}^{(\ell)}\mathbf{H}\right)\right), \\ =& \mathbf{W}^{(\ell)} + \boldsymbol{\gamma}^{(\ell)} \odot \left(\left(\mathbf{V} \oslash \mathbf{W}^{(\ell)}\mathbf{H}\right)\mathbf{H}^{\top} - \mathbb{1}\mathbf{H}^{\top}\right), \end{aligned} \tag{2.5}$$

where $\mathbb{1} \in \mathbb{R}^{K \times M}$ denotes an all-one matrix. Matrix $\boldsymbol{\gamma}^{(\ell)} \in \mathbb{R}^{K \times R}$ contains the *step sizes* of the elements of $\mathbf{W}$ at iteration (ℓ). The step sizes determine the magnitude of the variation of the values of $\mathbf{W}^{(\ell)}$ in the descent direction.

For gradient descent, it can be shown that $D\left(\mathbf{V}||\mathbf{W}^{(\ell+1)}\mathbf{H}\right) \leq D\left(\mathbf{V}||\mathbf{W}^{(\ell)}\mathbf{H}\right)$ and the condition $D\left(\mathbf{V}||\mathbf{W}^{(\ell+1)}\mathbf{H}\right) = D\left(\mathbf{V}||\mathbf{W}^{(\ell)}\mathbf{H}\right)$ is met when a stationary point can be reached if the step size is chosen properly. If the step size is too large, the update might miss the stationary point, making the optimization process diverge; if it is too small, the optimization will take more iterations to reach it. Choosing the right values for $\boldsymbol{\gamma}^{(\ell)}$ becomes critical for convergence, so they must be properly *scaled* down or up at each iteration.

Lee and Seung [18] propose using following step size to guarantee convergence to a stationary point:

$$\boldsymbol{\gamma}^{(\ell)} = \mathbf{W}^{(\ell)} \oslash \mathbb{1}\mathbf{H}^{\top}. \tag{2.6}$$

Given this step size, the update rule for $\mathbf{W}^{(\ell+1)}$ then becomes:

$$\mathbf{W}^{(\ell+1)} = \mathbf{W}^{(\ell)} \odot \left(\left(\mathbf{V} \oslash \mathbf{W}^{(\ell)}\mathbf{H}\right)\mathbf{H}^{\top}\right) \oslash \mathbb{1}\mathbf{H}^{\top}. \tag{2.7}$$

Algorithm 1: NMF using Multiplicative Updates

Input : Data matrix $\mathbf{V} \in \mathbb{R}_{\geq 0}^{K \times M}$.

Output : Factor matrices $\mathbf{W} \in \mathbb{R}_{\geq 0}^{K \times R}$ and $\mathbf{H} \in \mathbb{R}_{\geq 0}^{R \times M}$, such that $\mathbf{V} \approx \mathbf{WH}$.
Low rank approximation matrix $\widehat{\mathbf{V}} \in \mathbb{R}_{\geq 0}^{K \times M}$.

1 *Set parameter* R;

2 *Initialize matrices* $\mathbf{W}^{(0)}$ *and* $\mathbf{H}^{(0)}$;

3 *Define* L *(number of iterations)*;

4 $\ell = 0$;

while $\ell < L$ **do**

5 $\quad \mathbf{W}^{(\ell+1)} = \mathbf{W}^{(\ell)} \odot \left(\left(\mathbf{V} \oslash \mathbf{W}^{(\ell)} \mathbf{H}^{(\ell)} \right) \left(\mathbf{H}^{(\ell)} \right)^{\top} \right) \oslash \mathbb{1} \left(\mathbf{H}^{(\ell)} \right)^{\top}$;

6 $\quad \mathbf{H}^{(\ell+1)} = \mathbf{H}^{(\ell)} \odot \left(\left(\mathbf{W}^{(\ell+1)} \right)^{\top} \left(\mathbf{V} \oslash \mathbf{W}^{(\ell+1)} \mathbf{H}^{(\ell)} \right) \right) \oslash \left(\mathbf{W}^{(\ell+1)} \right)^{\top} \mathbb{1}$;

7 $\quad \ell = \ell + 1$;

end

8 $\mathbf{W} = \mathbf{W}^{L}$, $\mathbf{H} = \mathbf{H}^{L}$, $\widehat{\mathbf{V}} = \mathbf{WH}$.

This update rule contains only multiplication and division operations, and is therefore called a *multiplicative* update rule. The update rule for matrix $\mathbf{H}^{(\ell+1)}$ is derived by fixing $\mathbf{W}^{(\ell+1)}$ and using a similar step size value (with the role of $\mathbf{H}^{(\ell)}$ and $\mathbf{W}^{(\ell+1)}$ inverted). The update rule obtained is:

$$\mathbf{H}^{(\ell+1)} = \mathbf{H}^{(\ell)} \odot \left(\left(\mathbf{W}^{(\ell+1)} \right)^{\top} \left(\mathbf{V} \oslash \mathbf{W}^{(\ell+1)} \mathbf{H}^{(\ell)} \right) \right) \oslash \left(\mathbf{W}^{(\ell+1)} \right)^{\top} \mathbb{1}. \tag{2.8}$$

To apply these updates in Problem 2.2, they must be computed in an alternating fashion, following the block coordinate descent approach. An example procedure for NMF optimization using multiplicative updates is shown in Algorithm 1. By default, the values of matrices $\mathbf{W}^{(0)}$ and $\mathbf{H}^{(0)}$ are randomly generated, following a uniform or normal probability distribution over the non-negative subspace. The algorithm is stopped after L iterations.

Multiplicative updates are a simple way of guaranteeing convergence and enforcing non-negativity at the same time; if the initialization values are non-negative, the updates will remain non-negative, as only multiplicative operations are computed in each iteration. At the same time, any zero values in $\mathbf{W}^{(0)}$ and $\mathbf{H}^{(0)}$ will remain zero throughout the optimization process. These properties will prove to be important for sound decomposition tasks, and will be further discussed in Section 2.3.3.

2.3 NMF-Based Spectral Decomposition

As introduced in Chapter 1, a music recording can be thought of as the superposition of multiple sound sources. Formally, let $x : \mathbb{Z} \to \mathbb{R}$ be a real-valued, discrete time-domain signal corresponding to a music recording made up of $R \in \mathbb{N}$ sound sources. The goal of sound decomposition is to extract the sound sources present in the input signal and generate separate audio signals x_r, for $r \in [1 : R]$, such that $x = \sum_{r=1}^{R} x_r$.

A common first step in sound decomposition tasks is to first generate a time-frequency representation of the input signal [25]. This representation provides information about the behavior of the frequency content of the signal through time. In Section 2.1, NMF was defined as a "parts of a whole" approach. Therefore, applying NMF to the time-frequency representation of a music signal could retrieve information about the frequency content of the sound sources that make up the input signal, and their presence in time.

In this work, to obtain a time-frequency representation of x, the discrete version of the Short-time Fourier transform (STFT) is used. This is done by defining a window function $w : [0 : N-1] \to \mathbb{R}$ of even block size N, a hop size parameter $H \in \mathbb{N}$, and applying

$$\mathcal{X}(m,k) := \sum_{n=0}^{N-1} x(n+mH)w(n)\exp(-2\pi ikn/N), \tag{2.9}$$

where $\mathcal{X}(m,k) \in \mathbb{C}$ denotes the complex valued time-frequency coefficient of $\mathcal{X}$ for the k-th spectral bin at time frame m, for $k \in [1 : K]$ and $m \in [1 : M]$. The *magnitude spectrogram* of $\mathcal{X}$, obtained by setting

$$\mathbf{V} := |\mathcal{X}|^{\top}, \tag{2.10}$$

will be the input non-negative matrix for NMF.

In the magnitude spectrogram $\mathbf{V} \in \mathbb{R}_{\geq 0}^{K \times M}$, K stands for the number of frequency bins and M for the number of time frames[1]. If the original audio signal is made up of R sound sources, the columns in the factor matrix $\mathbf{W} \in \mathbb{R}_{\geq 0}^{K \times R}$ computed using NMF would contain an approximated frequency spectrum of each sound source. Consequently, the rows of matrix $\mathbf{H} \in \mathbb{R}_{\geq 0}^{R \times M}$ would encode the time frames when the sound sources are active. The approximated magnitude spectrogram for a particular sound source $\widehat{\mathbf{V}}_r$ for $r \in [1 : R]$ would be calculated by computing

[1] All magnitude spectrograms in this work were computed using a Hann function window w of block size $N = 2048$ and hop size $H = 512$. For a signal sampled at $Fs = 44100$ Hz, there will be a frequency resolution of $Fs/N \approx 21.5$ Hz and time resolution of $H/Fs \approx 11.6$ ms.

the inner product of its corresponding column and row of the factor matrices:

$$\widehat{\mathbf{V}}_r := \mathbf{w}_r \mathbf{h}_r, \tag{2.11}$$

with $\mathbf{w}_r \in \mathbf{W}$ being the r-th column vector of $\mathbf{W}$ and $\mathbf{h}_r \in \mathbf{H}$ the r-th row vector of $\mathbf{H}$. NMF is therefore performing a *spectral decomposition* of the input magnitude spectrogram $\mathbf{V}$, generating individual magnitude spectrograms for each of the sound sources from a single input. Nonetheless, generating an approximated component time-domain audio signal $\hat{x}_r$ from its magnitude spectrogram $\widehat{\mathbf{V}}_r$ is less trivial, and requires applying signal reconstruction techniques. This problem will be addressed in Section 4.2.

The remaining part of this section is organized as follows: First the audio running examples used throughout this work are presented in Section 2.3.1. Next, the NMF algorithm is applied to the presented examples in Section 2.3.2. The use of score information and onset models is discussed in Section 2.3.3.

2.3.1 Running Examples

Throughout this work, two main running examples will be used in order to illustrate the performance of spectral decomposition algorithms.

The first running example is a monophonic piano recording, consisting of three notes played with the same intensity. The piece is played at a tempo of 60 bpm, such that the duration of each note is exactly half a second. The music score, time domain signal, magnitude spectrogram and ground truth annotations for this example are shown in Figure 2.1.

The music score in Figure 2.1(a) indicates that the notes A4, C5 and E5 are first played one after the other, followed by groups of two notes at a single time (also called *intervals*), and finally all three notes played simultaneously forming the A minor chord. The amplitude of the time domain waveform in Figure 2.1(b) is proportional to the number of notes being played at a given time.

The magnitude spectrogram in Figure 2.1(c) shows the start of each note is marked by the presence of a vertical line, which in the frequency domain represents a flat, noise-like spectrum. This structure is called an *onset*, and physically corresponds to the instant when the hammer of the piano hits the string that produces the sound. The horizontal lines observed after the onset correspond to the fundamental frequency of each note and its harmonics, which are generated by the vibration modes of the piano string. There is also a noticeable overlap between the decay of the previous note and the start of the next one.

The magnitude spectrograms used in this work, such as the one in Figure 2.1(c), are max-

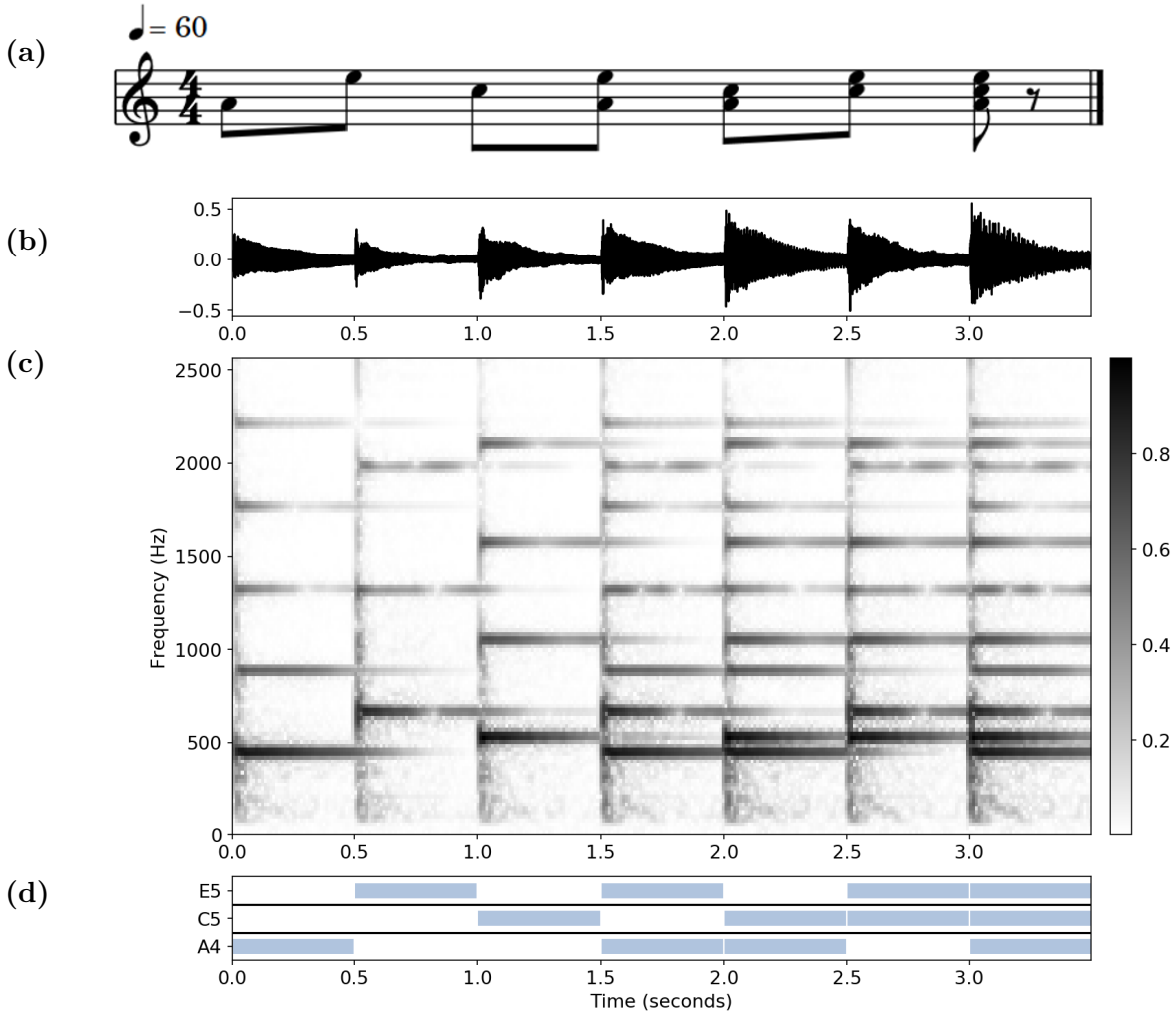

Figure 2.1. Representations for the piano running example. **(a)** Time-aligned music score. **(b)** Audio signal in the time domain. **(c)** Magnitude spectrogram, max-normalized and log-compressed. The vertical axis for the piano running example is limited to the $[0:2500]$ Hz range to focus on the lower part of the frequency spectrum. **(d)** Ground truth annotations.

normalized, i.e.

$$\mathbf{V}_{\text{max-norm}} = \mathbf{V}/\max(\mathbf{V}), \tag{2.12}$$

with $\max(\cdot) : \mathbb{R}^{K\times M} \to \mathbb{R}$ denoting the maximum values of all elements in $\mathbf{V}$. This operation is performed such that the values of $\mathbf{V}$ lie within the $[0:1]$ range. Additionally, and solely for the sake of better visualization of the spectrogram, logarithmic compression [25] is applied:

$$\mathbf{V}_{\text{log-comp}} = \log(1 + \gamma\mathbf{V}_{\text{max-norm}}), \tag{2.13}$$

with $\gamma \geq 1$ being a parameter used to adjust the compression rate.

Note Number	69	76	72	69	76	69	72	72	76	69	72	76
Start Time	0	0.5	1	1.5	1.5	2.0	2.0	2.5	2.5	3.0	3.0	3.0
End Time	0.5	1	1.5	2.0	2.0	2.5	2.5	3.0	3.0	3.5	3.5	3.5

Table 2.1. Ground truth annotations table for the piano example in Figure 2.1. The note number 69, 72 and 76 correspond to notes A4, C5 and E5 respectively.

For music recordings such as this piano example, ground truth annotations are usually available in the form of a table, where one axis corresponds to sound events, and the other to different characteristics of each event such as start time, end time, or MIDI note number. The ground truth annotations for the piano example are shown in Table 2.1.

Figure 2.1(d) displays the annotations in Table 2.1 in the form of a *piano roll representation*, and illustrates the start and duration of each note by the location and width of the rectangles over a grid. The grid rows correspond to the notes present in the music piece. Both the spectrogram and ground truth annotations will be the main reference for qualitatively evaluating spectral decomposition results.

The above piano example will be helpful when explaining certain spectral decomposition details which are easier to visualize using the clear frequency structures of the piano notes. However, the main focus of this thesis is drums sounds. Therefore the main running example used throughout this work will be a monophonic drums recording which includes the three core drumset components: kick drum (KD), snare drum (SD), and hi-hat (HH). The music score, time domain signal, magnitude spectrogram and ground truth annotations for this example are shown in Figure 2.2.

First of all, the drums score in Figure 2.2(a) is different from the piano music score in Figure 2.1(a). Lines and spaces do not represent pitch (indicated by the absence of a clef), but strokes on different drumset components. The space above the fifth line of the staff (counting down to up) corresponds to HH; the fourth space corresponds to SD; the first space, to KD. The ($\times$) symbol replacing the note head means that HH is meant to be played closed.

The time domain representation in Figure 2.2(b) shows impulse-like signals with short decay times, which in the magnitude spectrogram (Figure 2.2(c)), are represented as broadband spectra. In particular, KD presents strong low frequency content, and noticeable frequency overtones above 18 kHz (possibly due to a compression artifact enhanced by the logarithmic compression). SD is also present in the low frequencies, but has longer decay times and slightly above the frequency range of KD. HH is spread across the entire spectrum, centered around the [2500 : 12500] Hz range. There are clear frequency overlaps between the components and no visible tonal or harmonic structure. For example, it is difficult to distinguish the SD stroke in second 1.0 from the simultaneous HH and SD strokes in second 2.5. This makes the spectrogram decomposition task even more challenging for this type of instrument.

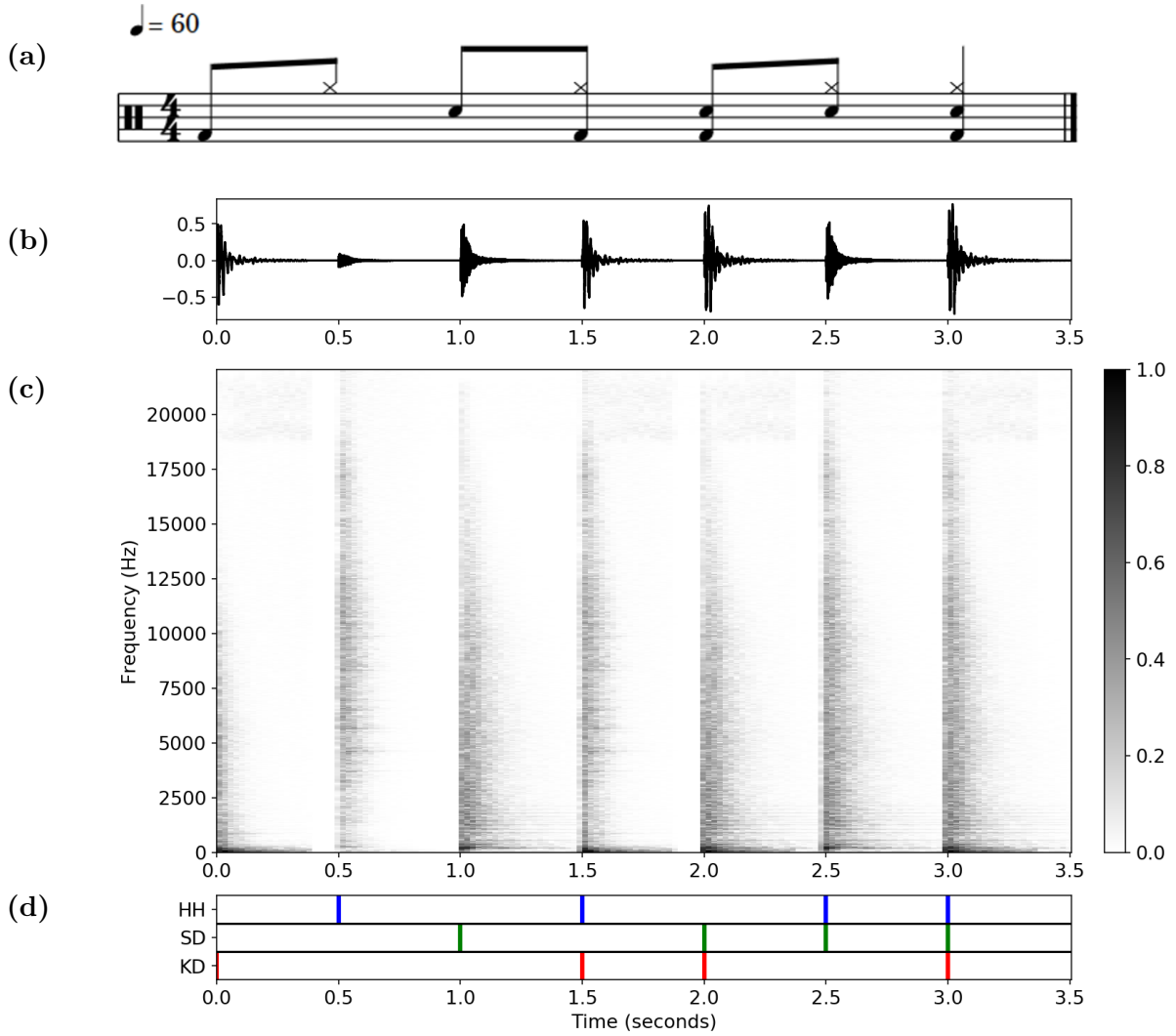

Figure 2.2. Representations for the drums running example. **(a)** Time-aligned music score. **(b)** Audio signal in the time domain. **(c)** Magnitude spectrogram. **(d)** Ground truth annotations.

The ground truth annotations in Figure 2.2(d) represent the presence of each component by vertical lines instead of the rectangles in Figure 2.1(d). This is because the duration of the sound in percussive instruments is short, but and also (usually) not controllable. Therefore, only the stroke event is relevant.

2.3.2 NMF with Random Initialization

As mentioned at the beginning of this section, the magnitude spectrogram is used as the input matrix $\mathbf{V}$ to perform spectral decomposition using NMF. The decomposition is performed at a *note level*. This means that, for both running examples, the sound sources denote different sounds produced by an instrument (piano or drumset), rather than representing multiple instruments in

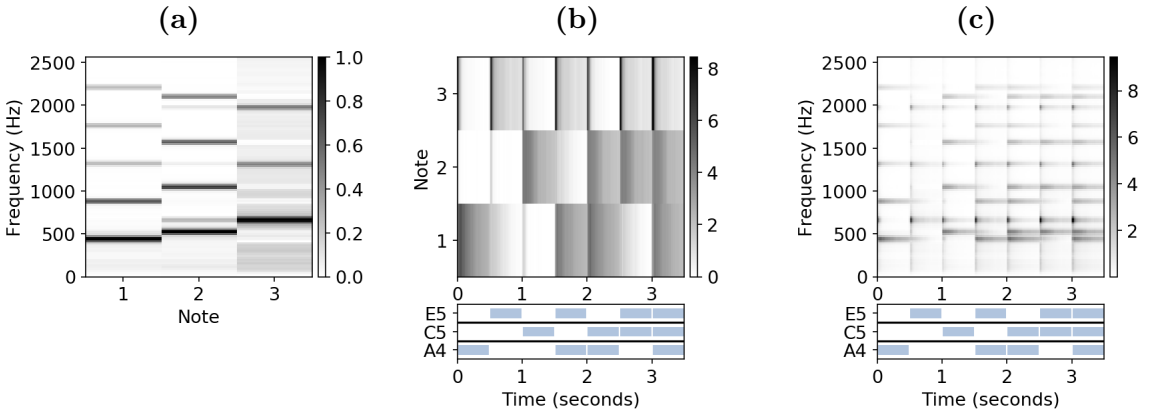

Figure 2.3. Randomly initialized NMF for the piano running example in Figure 2.1. **(a)** Learned template matrix $\mathbf{W}$. **(b)** Learned activations matrix $\mathbf{H}$ with reference ground truth annotations (below). **(c)** Low rank approximation matrix $\widehat{\mathbf{V}} = \mathbf{WH}$ with reference ground truth annotations (below).

a recording. The components in the piano example are the notes being played i.e. A4, C5 and E5; in the drums example, KD, SD and HH. Consequently, for both running examples $R = 3$.

NMF was implemented following Algorithm 1. Matrices $\mathbf{W}^{(0)}$ and $\mathbf{H}^{(0)}$ was initialized following a uniform distribution in the interval $[0, 1]$ for both examples. As a stop criterion, the algorithm was set to run for $L = 1000$ iterations. The output of the NMF algorithm for the piano example is shown in Figure 2.3. The ground truth annotations from Figure 2.1(d) are plotted underneath matrices $\mathbf{H}$ and $\widehat{\mathbf{V}}$ (Figures 2.3(b) and 2.3(c)) for reference.

The high values in the columns of $\mathbf{W}$ from Figure 2.3(a) coincide with the tonal structure of each note, and can be regarded as frequency *templates* for each of the notes of the recording. Matrix $\mathbf{W}$ is hence called the *template matrix*. On the other hand, the high values on the activation matrix $\mathbf{H}$ in Figure 2.3(d) coincide with the time instants where the frequency templates in the columns of $\mathbf{W}$ are active in the spectrogram. Matrix $\mathbf{H}$ is hence called the *activation matrix*. The activation matrix should fairly match the ground truth annotations below it.

For a better visualization of factor matrix plots, the columns $\mathbf{w}_r$ of the resulting matrix $\mathbf{W}$ and their corresponding rows $\mathbf{h}_r$ in $\mathbf{H}$ are scaled in the following way:

$$\mathbf{w}_r = \frac{\mathbf{w}_r}{\max(\mathbf{w}_r)}, \quad \mathbf{h}_r = \mathbf{h}_r \cdot \max(\mathbf{w}_r), \tag{2.14}$$

for $r \in [1 : R]$. This way matrix $\mathbf{H}$ also encodes the dynamics of the different components [2].

A closer look at the resulting template matrix $\mathbf{W}$ in Figure 2.3(a) reveals the frequency content of Note 2 (corresponding to C5 according to the annotations) contains some frequencies that belong to Note 1 (A4) and Note 3 (E5). This interference or leakage between frequency templates is

[2] The same scaling is used in all factor matrix plots in Chapters 2, 3 and A.

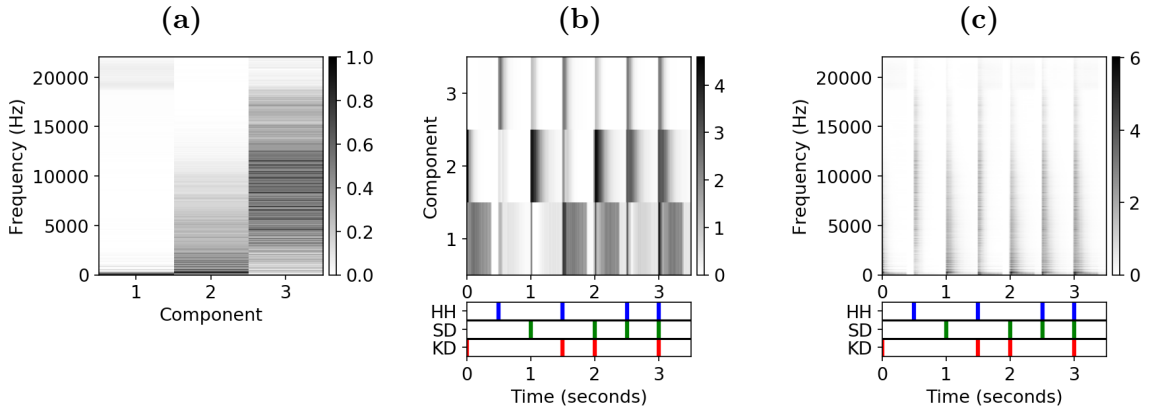

Figure 2.4. Randomly initialized NMF for the drums running example in Figure 2.2. **(a)** Learned template matrix $\mathbf{W}$. **(b)** Learned activations matrix $\mathbf{H}$ with reference ground truth annotations (below). **(c)** Low rank approximation matrix $\widehat{\mathbf{V}} = \mathbf{WH}$ with reference ground truth annotations (below).

also known as cross-talk, and is one of the most common artifacts found in sound decomposition or source separation in general.

Another noticeable artifact can be seen in the frequency content of note 3 (E5), where noise-like frequency components are present outside the harmonic intervals, forming a grey area that surrounds the tonal structure of the note. In the activation matrix, Note 3 presents sharp vertical lines every time any note event occurs. This means that Note 3 is encoding not only the harmonic structure of note E5, but also the onset information of all the notes in the piece.

Turning now to the drums running example, the NMF results using the magnitude spectrogram in Figure 2.2(c) are shown in Figure 2.4. Although in this case the frequency distribution of the components in the template matrix $\mathbf{W}$ (Figure 2.4(a)) looks close to the frequency distribution observed in the input spectrogram, the activations in matrix $\mathbf{H}$ (Figure 2.4(b)) present a more evident case of cross-talk between the components. This has derived in false detections, where components are shown as active at a given time instant but when the ground truth annotations do not. For example, the HH activations (Component 3) show that the HH was present at every half-second interval, when the ground truth annotations show HH is active only every second, except for the last stroke. The same phenomenon is noticeable in KD (Component 1) and SD (Component 2) to a lesser extent.

For both piano and drums examples, the approximated magnitude spectrograms $\widehat{\mathbf{V}}$ in Figures 2.3(c) and 2.4(c) fairly resemble the original spectrograms in Figures 2.1(c) and 2.2(c). However, the amplitude ranges of the $\widehat{\mathbf{V}}$ are considerably higher than in $\mathbf{V}$. In summary, this first NMF results show that, although successfully approximating the input spectrogram, the factor matrices $\mathbf{W}$ and $\mathbf{H}$ still do not provide a clear representation of the frequency content and temporal evolution of the components from the running examples in Figures 2.1(c) and 2.2(c), and do not constitute successful spectral decomposition. Cross-talk artifacts are expected due

to the noticeable frequency overlaps between the different components, specially for the drums example, and are expected become more noticeable if the number of components increases.

2.3.3 Informed Initialization

The music scores in Figures 2.1(a) and 2.2(a) show that the components are present only at certain time intervals during the recording. An ideal activation matrix would consequently have non-zero values only in the frames where a certain component is active. For the piano example, each note triggers only a specific set of frequencies (the fundamental frequency and harmonics of the note). An ideal template matrix columns should therefore have non-zero values only in certain frequency ranges. The work in [12] suggests incorporating score information from the running examples into the factor matrices improves the spectral decomposition results, as it guides the NMF optimization process to more musically meaningful factor matrices.

Score information can be introduced using an important property from the NMF multiplicative updates discussed in Section 2.2: A value in matrices $\mathbf{W}^{(0)}$ or $\mathbf{H}^{(0)}$ set to zero will remain zero throughout the optimization process. Exploiting this property, all values in matrix $\mathbf{H}^{(0)}$ can be set to zero except in time frames when a given component is active, using the ground truth annotations as a reference. In the case of the piano example, the values of matrix $\mathbf{W}^{(0)}$ can be set to zero, except in the regions near the fundamental frequency and harmonics of each note, following the harmonic series [25].

It is important to clarify that setting the values of $\mathbf{W}^{(0)}$ and $\mathbf{H}^{(0)}$ to zero will cause division-by-zero errors when using the multiplicative updates in Equations 2.7 and 2.8. In practice, this can be avoided by replacing zero values by a very small positive number (machine epsilon), or adding a small number to the denominator expressions. As noted by Lin [20], this modification does not affect convergence of the algorithm to a stationary point.

Figures 2.5(a) and 2.5(b) show the score-informed initialization matrices for the piano example. Both matrices were generated using the ground truth annotations in Figure 2.1(d) with the help of the FMP notebooks tool [26]. The informed template initialization matrix $\mathbf{W}^{(0)}$ was generated assuming a piano tuned to a 440 Hz reference, and computing the fundamental frequency from the note number p in Table 2.1, using the following formula:

$$F(p) = 2^{(p-69)/12} \cdot 440[\text{Hz}]. \tag{2.15}$$

Matrix $\mathbf{W}^{(0)}$ in Figure 2.5(a) shows a clear harmonic structure, where each column has non-zero values around the fundamental frequency and harmonics of each of the notes. The informed template matrix columns also exploit the fact that the high frequency harmonics contain less energy, and assigns them a lower initial value. The activation initialization matrix $\mathbf{H}^{(0)}$ in

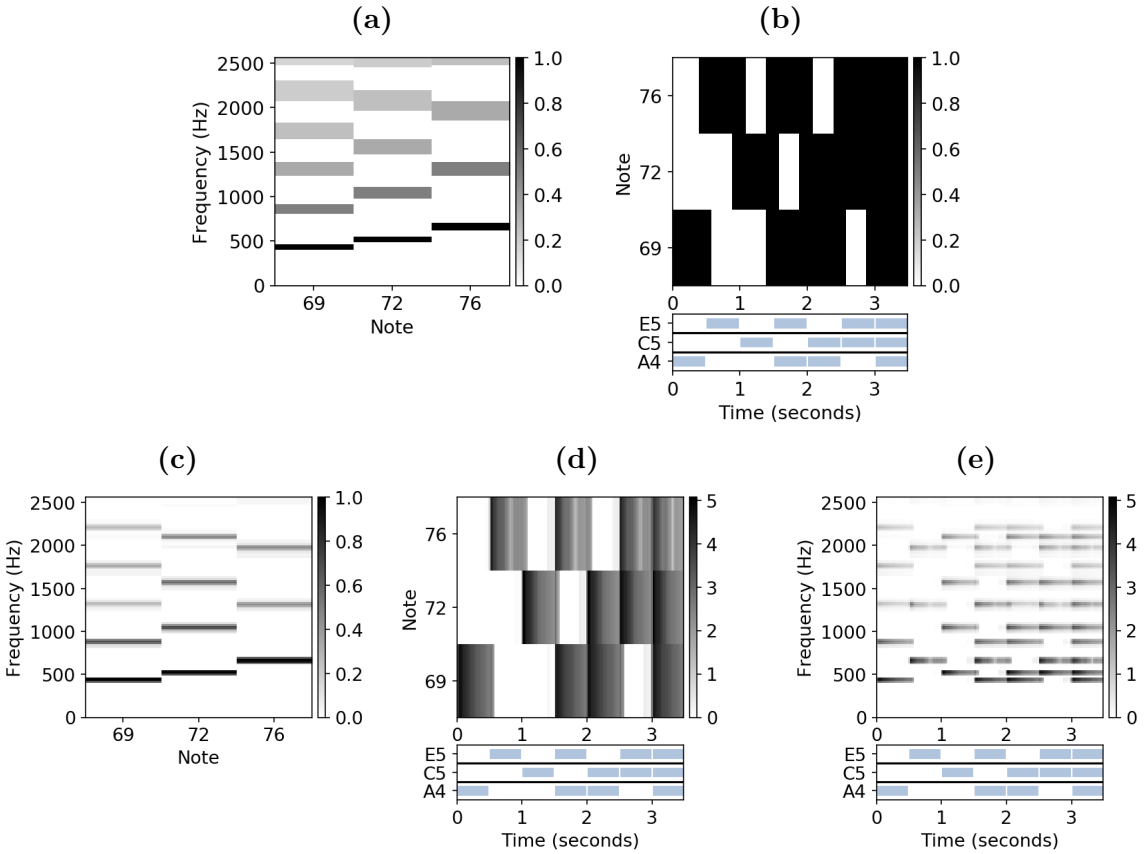

Figure 2.5. Score-informed NMF for the piano running example. **(a)** Pitch-based matrix $\mathbf{W}^{(0)}$. **(b)** Score-based Matrix $\mathbf{H}^{(0)}$ with reference ground truth annotations (below). **(c)** Learned template matrix $\mathbf{W}$. **(d)** Learned activations matrix $\mathbf{H}$ with reference ground truth annotations (below). **(e)** Low rank approximation matrix $\widehat{\mathbf{V}} = \mathbf{WH}$ with reference ground truth annotations (below).

Figure 2.5(b) is a binary matrix, generated using the start and end times of each note. The duration of each note was configured a bit longer as in the annotations, taking into account the overlapped transition observed between the end of each note and the beginning of the next one in the original spectrogram.

The learned matrices in Figure 2.5(c) and 2.5(d) show cleaner and defined templates and activations, as a result of the score-informed initialization. The templates are very effective in guiding the learning process of NMF, with the zero values remaining zero in the output factor matrices. However, the approximation $\widehat{\mathbf{V}}$ in Figure 2.5(e) shows only the horizontal structure of the note harmonics, leaving out the onset information of the notes. This means the informed initialization model has oversimplified the structure of its components, discarding the transients of the note events. To include the onset information, a more refined initialization scheme is required. This initialization scheme particular for piano recordings is called onset models, and is further explained in Appendix A.

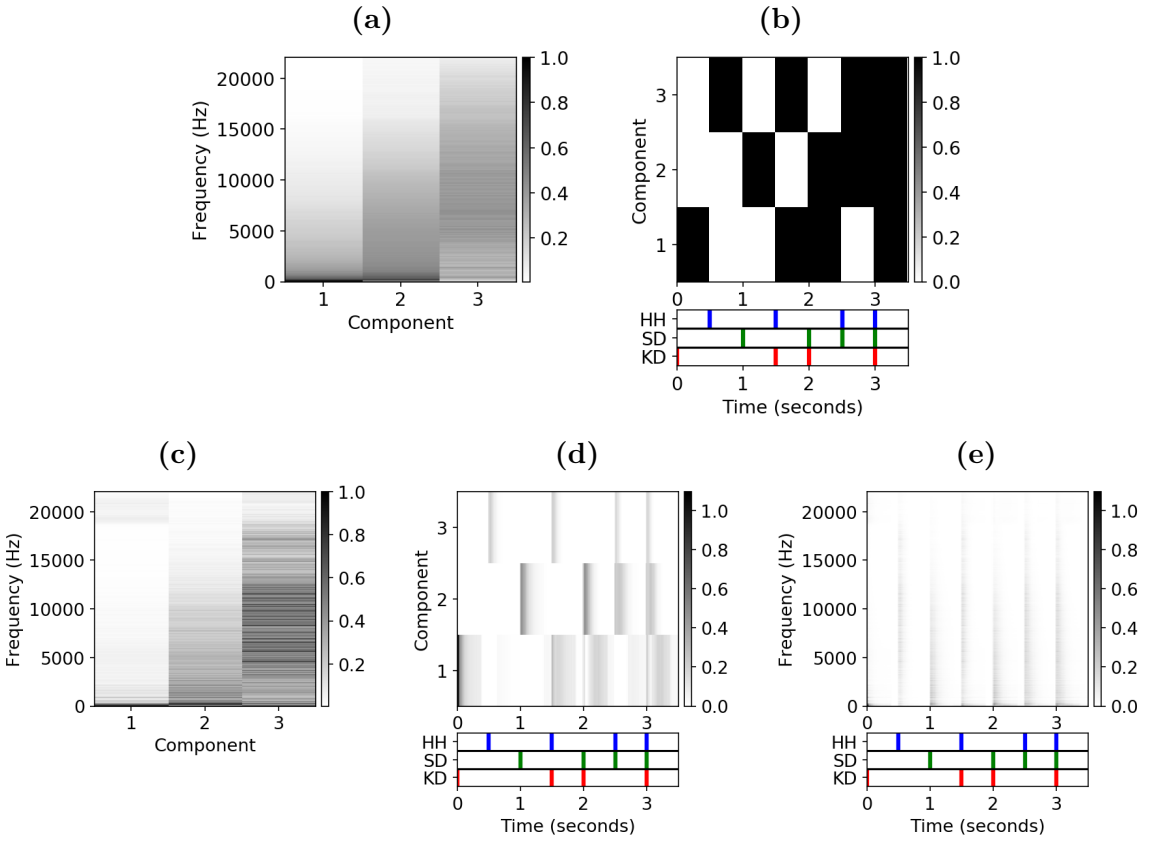

Figure 2.6. Score-informed NMF for the drums running example. **(a)** audio-based matrix $\mathbf{W}^{(0)}$ from the NMF Toolbox. **(b)** Score-based Matrix $\mathbf{H}^{(0)}$ with reference ground truth annotations (below). **(c)** Learned template matrix $\mathbf{W}$. **(d)** Learned activations matrix $\mathbf{H}$ with reference ground truth annotations (below). **(e)** Low rank approximation matrix $\widehat{\mathbf{V}} = \mathbf{WH}$ with reference ground truth annotations (below).

Figures 2.6(a) and 2.6(b) show the score-informed initialization matrices for the drums example. Matrix $\mathbf{H}^{(0)}$ in Figure 2.6(b) is similar to the one generated for the piano example Figure 2.5, but with slightly shorter duration set for each activation. On the other hand, since there is only a vague idea of the frequency content distribution of the drumset components and such distribution has no fixed harmonic structure, matrix $\mathbf{W}^{(0)}$ cannot be set using the same procedure as in the piano example. Matrix $\mathbf{W}^{(0)}$ in Figure 2.6(a) was generated using audio-based templates. These templates were previously obtained from perfectly isolated samples of each instrument, and were taken from the NMF Toolbox [24], successfully implemented in [9] for drum-related applications.

The resulting template matrix $\mathbf{W}$ in Figure 2.6(c) looks fairly similar to the randomly-initialized template matrix in Figure 2.4(a); however, there is a noticeable improvement in the learned activation matrix $\mathbf{H}$ in Figure 2.6(d). There is no longer false activations, meaning no components are active in time instants where they are not supposed to. The value range of Figure 2.4(e) is now much closer to the value range of the input spectrogram in Figure 2.2(c).

Leakage is still present in the decomposition, and can be observed in the component dynamics. All

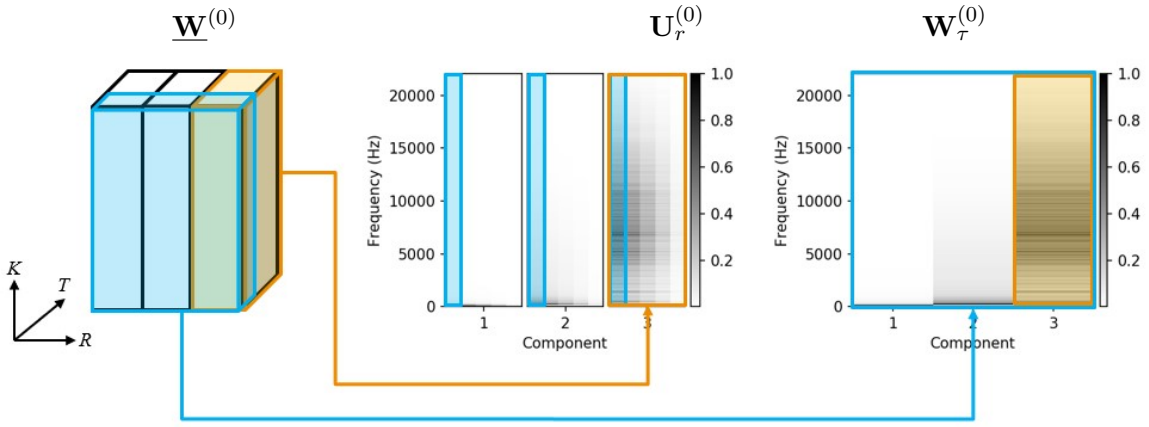

Figure 2.7. NMFD score-informed tensor $\underline{\mathbf{W}}^{(0)}$ for the drums running example with $T = 5$. Matrix $\mathbf{U}_r^{(0)}$ represents the temporal evolution of the frequency content of component r, while $\mathbf{W}_\tau^{(0)}$ is the initial template matrix for the temporal frame τ.

component strokes were played at the same amplitude, but the components in $\mathbf{H}$ show different values in different strokes. For example, in Figure 2.6(d) seconds 1.5,2 or 3 the KD stroke (Component 1) is noticeably weaker than in second 0. This also happens in SD (Component 2). These lower amplitudes coincide with the time instants where HH (Component 3) is active. This implies that, because of the wide frequency spectrum of HH in this example, some of the energy from the KD or SD strokes is being assigned to HH.

From this results it can be concluded that the use of score information from ground truth annotations for NMF is an important step towards obtaining better spectral decomposition results, but it is certainly not definitive, as the factorization model is still simple and prone to component leakage.

2.4 Non-negative Matrix Factor Deconvolution (NMFD)

Given the NMF results for the piano example in Figure 2.3, it is clear that the frequency content of a sound source is not stationary. For the piano example, the frequency content of a note transitions from an onset to a structured harmonic spectrum. The spectral decomposition algorithm should also account for this temporal dimension. Non-negative matrix factor deconvolution (NMFD) [30] was developed precisely as an extension of NMF to include the temporal evolution of the frequency content of each component.

Instead of the single template matrix $\mathbf{W}$ used in NMF, NMFD uses a set of matrices $\mathbf{W}_\tau \in \mathbb{R}_{\geq 0}^{K \times R}$ for $\tau \in [1 : T]$. The parameter $T \in \mathbb{N}$ denotes the total amount of templates desired. Each template matrix $\mathbf{W}_\tau$ is used for a different temporal frame $\tau \in [1 : T]$, having T templates available for a single sound source. An example of an informed initialization $\underline{\mathbf{W}}^{(0)}$ for the drums

example is shown in Figure 2.7. The templates $\mathbf{W}_\tau$ can be grouped in a *tensor* $\underline{\mathbf{W}} \in \mathbb{R}_{\geq 0}^{K \times R \times T}$, where matrices $\mathbf{W}_\tau$ correspond to slices of $\underline{\mathbf{W}}$ along the T dimension. The evolution of the frequency content of each of component can be found in matrices $\mathbf{U}_r \in \mathbb{R}^{K \times T}$ for $r \in [1:R]$, which are slices of $\underline{\mathbf{W}}$ along R. Matrix $\mathbf{W}_\tau^{(0)}$ can be obtained by taking columns of the matrices of $\mathbf{U}_r$ for a given τ. NMF can be hence considered a particular case of NMFD for $T = 1$.

Instead of a matrix product, in NMFD the approximation matrix $\widehat{\mathbf{V}}$ is the result of a convolution operation between $\underline{\mathbf{W}}$ and $\mathbf{H}$, denoted $\underline{\mathbf{W}} * \mathbf{H}$:

$$\widehat{\mathbf{V}} := \sum_{\tau=1}^{T} \mathbf{W}_\tau \overset{\tau\rightarrow}{[\mathbf{H}]}, \tag{2.16}$$

where $\overset{\tau\rightarrow}{[\cdot]}$ denotes the right shift operator, which shifts the columns of a matrix to the right τ times. The shift operator allows the different templates to be activated by a single frame of matrix $\mathbf{H}$ [3]. The approximation matrix $\widehat{\mathbf{V}}$ is in this case the result of the sum of the approximations computed for all $\tau \in [1:T]$.

As in NMF, NMFD seeks to find optimal matrices $\mathbf{W}_\tau, \tau \in [1:T]$ and $\mathbf{H}$ such that the function $D(\mathbf{V}||\widehat{\mathbf{V}})$ is minimized. The optimization problem in NMFD is very similar to the optimization in NMF:

$$\begin{aligned} \min_{\underline{\mathbf{W}},\mathbf{H}} \quad & D\left(\mathbf{V}||\widehat{\mathbf{V}}\right), \\ \text{subject to} \quad & \underline{\mathbf{W}} \in \mathbb{R}_{\geq 0}^{K \times R \times T}, \mathbf{H} \in \mathbb{R}_{\geq 0}^{R \times M}. \end{aligned} \tag{2.17}$$

The multiplicative updates 2.7 and 2.8 are consequently reformulated to fit this new framework by incorporating the shift operator, as follows:

$$\mathbf{W}_\tau^{(\ell+1)} = \mathbf{W}_\tau^{(\ell)} \odot \left(\left(\mathbf{V} \oslash \widehat{\mathbf{V}} \right) \left(\overset{\tau\rightarrow}{\left[\mathbf{H}^{(\ell)}\right]} \right)^\top \right) \oslash \mathbb{1} \left(\overset{\tau\rightarrow}{\left[\mathbf{H}^{(\ell)}\right]} \right)^\top \quad \forall \tau \in [1:T], \tag{2.18}$$

$$\mathbf{H}^{(\ell+1)} = \mathbf{H}^{(\ell)} \odot \left(\left(\mathbf{W}_\tau^{(\ell+1)} \right)^\top \overset{\leftarrow\tau}{\left[\mathbf{V} \oslash \widehat{\mathbf{V}} \right]} \right) \oslash \left(\mathbf{W}_\tau^{(\ell+1)} \right)^\top \mathbb{1}, \tag{2.19}$$

where $\overset{\leftarrow\tau}{[\cdot]}$ denotes the left shift operator. The updates retain the properties of the NMF multiplicative updates, enforcing non-negativity while preserving convergence.

As in NMF, score information can be introduced in the initialization step of NMFD using audio-based initialization templates as the ones shown in Figure 2.7. For matrix $\mathbf{H}^{(0)}$, the same NMF initialization matrix can be used (see Figure 2.6(b)). Algorithm 1 can be used to compute NMFD, replacing the corresponding multiplicative updates. The approximated

[3] For a more detailed explanation on the convolution operation, refer to Appendix B

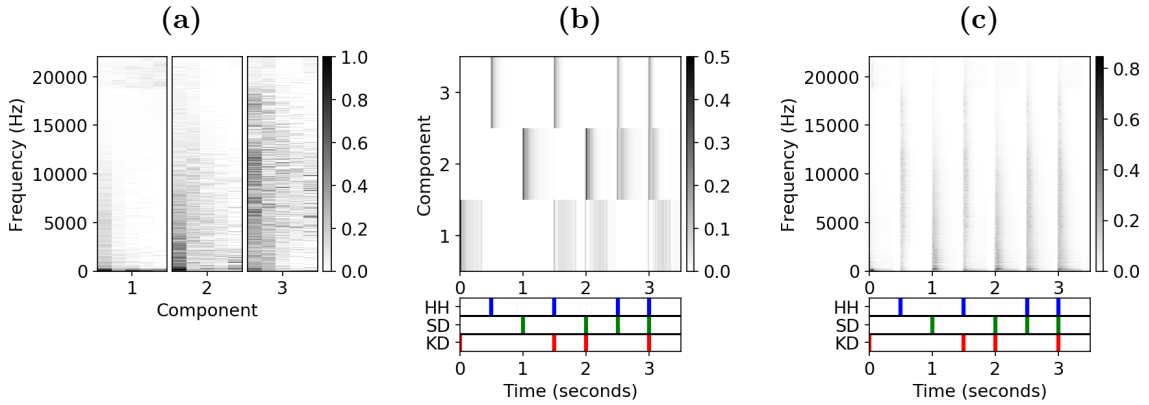

Figure 2.8. Score-informed NMFD for the drums running example. **(a)** Learned template matrices $\underline{\mathbf{W}}$. **(b)** Learned activation matrix $\mathbf{H}$ with reference ground truth annotations (below). **(c)** Low-rank approximation matrix $\widehat{\mathbf{V}} = \sum_{\tau=1}^{T} \mathbf{W}_\tau \overrightarrow{[\mathbf{H}]}^{\tau}$ with reference ground truth annotations (below).

magnitude spectrogram for a particular source $\widehat{\mathbf{V}}_r$ for $r \in [1:R]$ in this case can be calculated by computing the convolution of its component matrix $\mathbf{U}_r$ and its corresponding row from the activation matrix $\mathbf{h}_r$, as follows:

$$\widehat{\mathbf{V}}_r = \sum_{\tau=1}^{T} (\mathbf{U}_r)_\tau \overrightarrow{[\mathbf{h}_r]}^{\tau}, \tag{2.20}$$

where $(\mathbf{U}_r)_\tau$ stands for column τ of matrix $\mathbf{U}_r$.

The resulting NMFD matrices using score-informed initialization matrices for the drums running example is shown in Figure 2.8. Following the implementation by Dittmar et.al in [9], the number of iterations was set to $L = 30$ and the number of temporal frames to $T = 5$.

The obtained templates $\underline{\mathbf{W}}$ in Figure 2.8(a) now include the temporal evolution of the frequency content for each of the drumset components. A faster decay is observed in the amplitude of the frequency components of HH (Component 3) compared to those of KD (Component 1) and SD (Component 2). In the case of SD, the upper frequencies have a lower amplitude and decay faster than the lower ones. Cross-talk effects similar to those observed in Figure 2.6 are also present in the activation matrix $\mathbf{H}$ in Figure 2.8(b). This means that including the temporal dimension in the template matrix of the components is still not enough to cancel cross-talk artifacts, as there is still frequency overlaps in between the component matrices. From the resulting approximation matrix $\widehat{\mathbf{V}}$ it is not clear whether there is an improvement on the reconstruction. For a quantitative analysis on the sound decomposition quality of NMFD, refer to Section 4.3.

Successful implementations in [9] and [23] show that NMFD is a powerful tool for spectral decomposition, in particular for automatic drum transcription [8] and sound decomposition tasks.

Chapter 3

DNN-Based Non-negative Matrix Factorization

This chapter introduces various neural network concepts used in the implementation of deep neural networks, and closely follows the work of Goodfellow [14] and Nielsen [27]. The author suggests referring to this material for a detailed review on neural networks and deep learning concepts.

The field of Artificial Intelligence (AI) has been developed as a way of using computers to tackle problems that humans solve intuitively, but which are difficult to define or describe [14]. Tasks such as identification, classification or prediction are performed by humans in a way that feels automatic, but that can be difficult to emulate from an engineering point of view .

The simplest form of AI comes in the form of an artificial neuron. An artificial neuron is essentially a mapping of a set of input values, described by the following equation:

$$y = g\left(b + \sum_{i=1}^{L} w_i a_i\right),$$

where the neuron output $y \in \mathbb{R}$ represents a sum of its inputs $a_i \in \mathbb{R}$ weighted by a set of coefficients or weights $w_i \in \mathbb{R}$ for $i \in [1 : L]$. The function $g : \mathbb{R} \to \mathbb{R}$ is called the *activation function* establishes a linear or non-linear relation between input and output. Parameter b is called the *bias*, and controls how easy or difficult it is to get a high value at the output. An artificial neuron can work e.g. as a logic gate when g is chosen to be the unit step function and the parameter $b \in \mathbb{R}$ is set to a desired threshold value.

Neurons can be used to compute more complex operations when grouped into *layers*. Neurons in a layer are not connected between them, but to their inputs. Given a set of input values arranged as a vector $\mathbf{a} \in \mathbb{R}^L$, and a matrix $\mathbf{W} \in \mathbb{R}^{N \times L}$ representing the weighted connections between L

inputs and N neurons, the layer output vector $\hat{\mathbf{y}} \in \mathbb{R}^N$ is defined as:

$$\hat{\mathbf{y}} = g\left(\mathbf{b} + \mathbf{W} \cdot \mathbf{a}\right),$$

where function g is applied element-wise. This type of layer, where every neuron is connected to every input value, is called a *fully connected* or dense layer.

In AI problems, it is usual to have only an input vector $\mathbf{a}$ and a desired or reference output $\mathbf{y} \in \mathbb{R}^N$ as given. In this case, the task is to find the layer weights $\mathbf{W}$ and biases $\mathbf{b}$ such that the neuron layer output $\hat{\mathbf{y}}$ coincides with the desired output $\mathbf{y}$. This process is called *training*, and can be summarized by the following optimization problem:

$$\min_{\mathbf{W},\mathbf{b}} \quad \mathcal{L}\left(\mathbf{y}, \hat{\mathbf{y}}\right), \tag{3.1}$$

with $\mathcal{L} : \mathbb{R}^{N^2} \to \mathbb{R}$ being a loss function which is used as an error measure between $\mathbf{y}$ and $\hat{\mathbf{y}}$.

A neural network consists of various layers stacked sequentially, in order to compute multiple operations for more complex problems. The *depth* of a network refers to the amount of layers it contains. A neural network with two or more layers is called a Deep neural network (DNN) [27]. With proper training processes, DNNs can become powerful tools for solving complex AI problems.

This chapter explores the use of neural network models in spectral decomposition. The lessons learned in the implementation of NMF in Chapter 2 will prove to be essential in the configuration of these networks, and in achieving meaningful spectral decomposition results. The chapter is organized as follows: Section 3.1 introduces the reader to the Non-negative autoencoder (NAE), which will be the neural network model used in this thesis to perform spectral decomposition, and establishes its relation with the NMF algorithm. The NAE spectral decomposition results are presented in Section 3.2. Various methods for including score information in the training process of NAEs are explained in Section 3.3. Finally, Section 3.4 describes the use of convolutional layers in NAEs and their effect on spectral decomposition.

3.1 Non-negative Autoencoders (NAE)

An autoencoder is a DNN architecture built to learn data coding models by attempting to reconstruct its input in its output [14]. In an autoencoder topology, the input data is first fed into the encoder stage $\mathcal{E}$, which outputs a set of features that build up a representation of the input signal, also called the *code*. This representation is the input of the decoder stage $\mathcal{D}$, intended to reverse the encoding process to reconstruct the original input data from the code. Autoencoders are hence trained using their own input data as reference, trying to generate the output that better resembles the original input. The difference between the input and output data of the

autoencoder is called the *reconstruction error*.

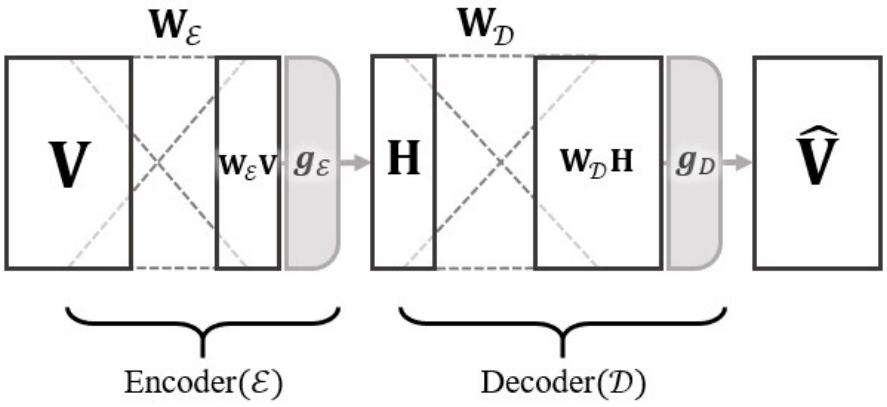

Figure 3.1. Diagram of a shallow autoencoder architecture.

Figure 3.1 illustrates a shallow autoencoder architecture. The name shallow is given because encoder and decoder are each built using a single, fully connected layer. The input matrix $\mathbf{V} \in \mathbb{R}^{K\times M}$ is fed into the encoder layer $\mathcal{E}$ with weights $\mathbf{W}_{\mathcal{E}} \in \mathbb{R}^{K\times M}$ and activation function $g_{\mathcal{E}} : \mathbb{R}^{R\times M} \to \mathbb{R}^{R\times M}$. The encoder layer outputs the code matrix $\mathbf{H} \in \mathbb{R}^{R\times M}$. The reconstructed input matrix $\widehat{\mathbf{V}} \in \mathbb{R}^{K\times M}$ is then computed in the decoder layer $\mathcal{D}$ with weight matrix $\mathbf{W}_{\mathcal{D}} \in \mathbb{R}^{K\times R}$ and activation function $g_{\mathcal{D}} : \mathbb{R}^{K\times M} \to \mathbb{R}^{K\times M}$. The influence of bias parameters will is not taken into account for the derivations in this chapter.

The shallow autoencoder architecture can be summarized as follows:

$$\begin{aligned} &1.\ \text{Encoder}(\mathcal{E}): \quad \mathbf{H} = g_{\mathcal{E}}(\mathbf{W}_{\mathcal{E}} \cdot \mathbf{V}), \\ &2.\ \text{Decoder}(\mathcal{D}): \quad \widehat{\mathbf{V}} = g_{\mathcal{D}}(\mathbf{W}_{\mathcal{D}} \cdot \mathbf{H}). \end{aligned} \tag{3.2}$$

In the training process of autoencoders, the goal is to obtain optimal values of $\mathbf{W}_{\mathcal{E}}$ and $\mathbf{W}_{\mathcal{D}}$ such that the reconstruction error is minimized. This can be formulated as the following optimization problem:

$$\min_{\mathbf{W}_{\mathcal{E}},\mathbf{W}_{\mathcal{D}}} \quad \mathcal{L}\left(\mathbf{V}, \widehat{\mathbf{V}}\right), \tag{3.3}$$

where the function $\mathcal{L} : \mathbb{R}^{(K\times M)^2} \to \mathbb{R}$ is defined as the reconstruction error between input matrix $\mathbf{V}$ reconstructed output $\widehat{\mathbf{V}} = g_{\mathcal{D}}\left(\mathbf{W}_{\mathcal{D}} \cdot g_{\mathcal{E}}(\mathbf{W}_{\mathcal{E}} \cdot \mathbf{V})\right)$. Functions such as the KLD loss function in Equation (2.3) can be used as reconstruction error functions.

As in Section 2.2, the gradient descent method can be used to estimate optimal values of the layer weight matrices. Let $\mathbf{W}_{\mathcal{E}}^{(\ell)}$ and $\mathbf{W}_{\mathcal{D}}^{(\ell)}$ denote the values of the encoder and decoder weight

matrices at iteration ℓ. The gradient descent updates for each weight matrix can be written as:

$$\begin{aligned}\mathbf{W}_{\mathcal{E}}^{(\ell+1)} &:= \mathbf{W}_{\mathcal{E}}^{(\ell)} - \eta_{\mathcal{E}}^{(l)} \odot \left(\nabla_{\mathbf{W}_{\mathcal{E}}} \mathcal{L}\left(\mathbf{V}, \widehat{\mathbf{V}}\right)\right), \\ \mathbf{W}_{\mathcal{D}}^{(\ell+1)} &:= \mathbf{W}_{\mathcal{D}}^{(\ell)} - \eta_{\mathcal{D}}^{(l)} \odot \left(\nabla_{\mathbf{W}_{\mathcal{D}}} \mathcal{L}\left(\mathbf{V}, \widehat{\mathbf{V}}\right)\right).\end{aligned} \tag{3.4}$$

The values of matrices $\mathbf{W}_{\mathcal{E}}^{(0)}$ and $\mathbf{W}_{\mathcal{D}}^{(0)}$ can be initialized randomly using a probability distribution. Typically, a zero-mean normal distribution is used. The computation of the gradient operations $\nabla_{\mathbf{W}_{\mathcal{E}}} \mathcal{L}\left(\mathbf{V}, \widehat{\mathbf{V}}\right)$ and $\nabla_{\mathbf{W}_{\mathcal{D}}} \mathcal{L}\left(\mathbf{V}, \widehat{\mathbf{V}}\right)$ is done efficiently using back propagation.

Matrices $\eta_{\mathcal{E}}^{(l)}$ and $\eta_{\mathcal{D}}^{(l)}$ contain the so-called *learning rates* of the network. As the step sizes in Equation (2.5), the learning rates will determine the convergence of the algorithm and therefore the success of the training process. Due to their importance, multiple algorithms have been developed to effectively choose learning rates that lead to faster convergence. In this work the RMSProp algorithm [14] is used. RMSProp is a commonly used algorithm for neural networks, which scales the learning rates inversely proportional to the square root of a decaying average of its gradients. This scaling lowers the values of matrices $\eta_{\mathcal{E}}^{(l)}$ and $\eta_{\mathcal{D}}^{(l)}$ as the loss function $\mathcal{L}\left(\mathbf{V}, \widehat{\mathbf{V}}\right)$ decreases. The details of the RMSProp and back propagation algorithms and equations are beyond the scope of this thesis; for further information, refer to the works in [27] and [14].

Notice that the decoder layer equation in (3.2) is similar to the NMF approximation of Equation (2.1). In fact, Smaragdis [31] suggests the autoencoder architecture has a direct relation with the NMF algorithm, and it could be thought of as a DNN-based matrix factorization model.

A series of parameters can be set in order to bring the autoencoder closer to a NMF scenario. First, a suitable code dimension R must be chosen. When the code dimension R is chosen such that $R \ll K$, the autoencoder is unable to perfectly reconstruct the input signal. This forces the autoencoder to generate a code with the most characteristic features of the data in order to minimize the reconstruction error. By doing so, the autoencoder is performing dimensionality reduction on the input data [16], and matrix $\widehat{\mathbf{V}}$ becomes a low-rank approximation of $\mathbf{V}$. In this case the autoencoder is said to be *undercomplete*. Additionally, the activation function $g_{\mathcal{E}}$ and $g_{\mathcal{D}}$ should be chosen to be a non-linear function that yields non-negative outputs, such that the code matrix $\mathbf{H}$ and the low rank approximation output matrix $\widehat{\mathbf{V}}$ are non-negative. The Rectified linear unit (ReLU) function $g(x) = \max(x, 0)$ can be used for this purpose, when applied element-wise over the input matrix values.

From an NMF perspective, choosing ReLU for the activation functions $g_{\mathcal{E}}$ and $g_{\mathcal{D}}$, the code matrix $\mathbf{H}$ would play the role of the activation matrix[1], and $\widehat{\mathbf{V}}$ the low-rank approximation of $\mathbf{V}$. The decoder weight matrix $\mathbf{W}_{\mathcal{D}}$ would correspond to the template matrix. Finally, the code dimension R would play the role of the NMF rank. If R is set such that the autoencoder is

[1]The terms *activation matrix* and *code matrix* is hereafter used to refer to matrix $\mathbf{H}$ indistinctly.

undercomplete, this configuration is called a *non-negative autoencoder* (NAE). NAEs are the main focus of this chapter.

3.2 NAE-based Spectral Decomposition

Having established the relation between NAE and NMF from a theoretical perspective, it is pertinent to use NAEs to perform spectral decomposition on the piano and drums running examples of Section 2.3.1, the same way as it was done for NMF in Section 2.3, in order to perform a practical comparison between the two approaches.

A shallow NAE architecture following Equations (3.2) was implemented using the Keras API [4] and the Tensorflow backend [1]. Both $g_{\mathcal{E}}$ and $g_{\mathcal{D}}$ were chosen to be ReLU functions, and the loss function $\mathcal{L}$ was chosen to be the KLD loss function in Equation (2.3) defined in Section 2.1. The bias values for both layers were set to zero to focus our attention solely on the learned weights. The neural network was trained for 1000 epochs using a full-batch approach, following the implementations in [31] and [13]. This means the whole matrix $\mathbf{V}$ was used as a unique training example, instead of training the network using single vectors or groups of vectors of $\mathbf{V}$.

No other neural network parameters were set, such that the NAE configuration was as similar to the NMF configurations as possible, in order to focus on the differences between NMF and NAE results, given that both models are trained following different approaches. A summary of the NAE architecture implementation is shown in Table 3.1.

Layer	**Output Shape**	**Parameters**
Input	$(1, K, M)$	
Encoder ($\mathcal{E}$)		
Dense + ReLU	$(1, R, M)$	$R \times K$
Decoder ($\mathcal{D}$)		
Dense + ReLU	$(1, K, M)$	$K \times R$

Table 3.1. NAE network architecture. The first dimension in the values of the Output Shape column indicates the network is trained using a full-batch training approach.

3.2.1 NAE using Random Initialization

As a first spectral decomposition experiment, the encoder and decoder layer weight values $\mathbf{W}_{\mathcal{D}}^{(0)}$ and $\mathbf{W}_{\mathcal{D}}^{(0)}$ were initialized using a random uniform distribution in the interval $[0, 0.1]$. The resulting decoder weight matrix $\mathbf{W}_{\mathcal{D}}$, the code matrix $\mathbf{H}$ and the low-rank approximation matrix $\widehat{\mathbf{V}}$ using NAE for the piano running example are shown in Figure 3.2.

The code matrix $\mathbf{H}$ in Figure 3.2(b) does not match the ground truth annotations, but rather shows multiple notes active in almost every frame. This behavior can be explained by looking at

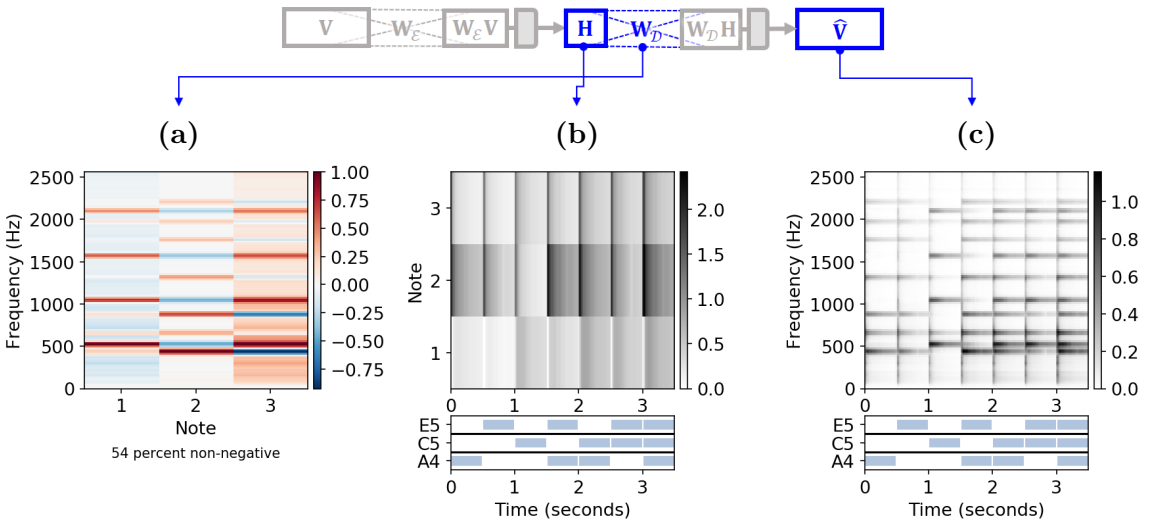

Figure 3.2. randomly-initialized NAE for the piano running example in Figure 2.1. **(a)** Learned decoder weight matrix $\mathbf{W}_\mathcal{D}$. **(b)** Learned code matrix $\mathbf{H}$ with reference ground truth annotations (below). **(c)** Low-rank approximation matrix $\widehat{\mathbf{V}}$ with reference ground truth annotations (below).

the weight matrix $\mathbf{W}_\mathcal{D}$ in Figure 3.2(a). The decoder weight matrix $\mathbf{W}_\mathcal{D}$ is a mixture of positive and negative values (even though it was initialized as non-negative). The matrix is plotted using a red-blue color map to better observe the distribution of positive (red) and negative values (blue), centered around zero (white). In each of the decoder matrix columns, a set of positive values approximates the harmonic structure seen in the NMF template matrix of Figure 2.3. However, for each noticeable positive value in a column of $\mathbf{W}_\mathcal{D}$, a negative counterpart can be found in one of the other two columns. The low rank approximation matrix $\widehat{\mathbf{V}}$ in Figure 3.2(c) resembles the input matrix, but is the result of the cancellation between positive and negative terms of the columns in $\mathbf{W}_\mathcal{D}$, so all components are active in every frame. The NAE output matrices for the drums running example shown in Figure 3.3 present a similar behaviour.

This NAE configuration is not strictly speaking the closest it can be to a NMF scenario; in fact, it is closer to a variant of NMF called *semi NMF* [7], where no constraint is enforced on the template matrix $\mathbf{W}$. Just like the template matrix in NMF, the NAE requires the decoder layer weights $\mathbf{W}_\mathcal{D}$ to be non-negative to avoid the cancellation between its columns.

3.2.2 NAE with Non-negative Decoder

In [31], Smagagdis et.al. propose indirectly enforcing non-negativity on the decoder layers weights by using a penalty term in the loss function, also called *regularizer*. A regularizer is a term added to the loss function which penalizes a certain behavior by making the loss function increase. This way the network will be optimized trying to avoid solutions that increase the value of the regularizer term and affect the total loss.

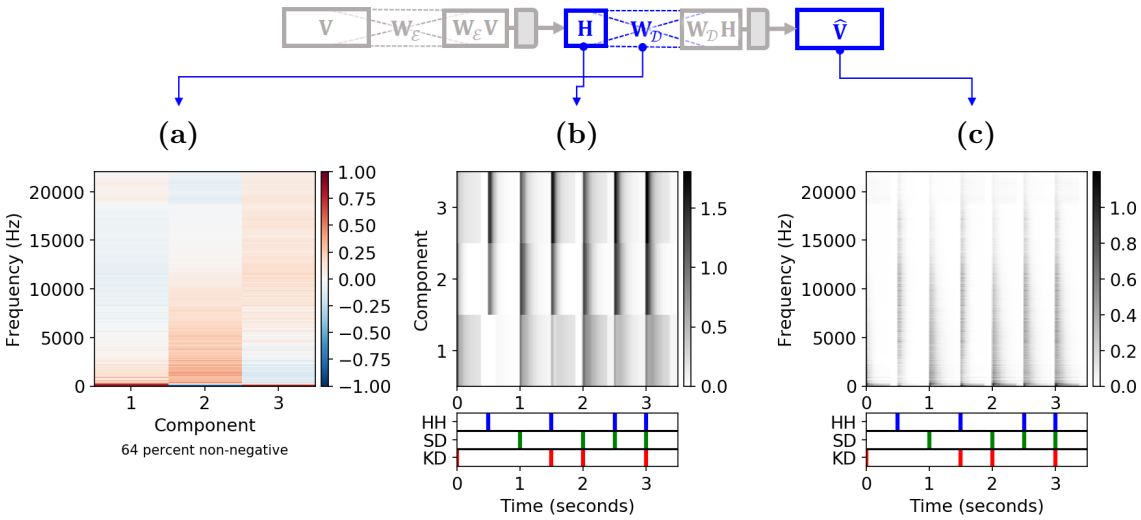

Figure 3.3. Randomly-initialized NAE for the drums running example in Figure 2.2. **(a)** Learned decoder weight matrix $\mathbf{W}_\mathcal{D}$. **(b)** Learned code matrix $\mathbf{H}$ with reference ground truth annotations (below). **(c)** Low-rank approximation matrix $\widehat{\mathbf{V}}$ with reference ground truth annotations (below).

Smaragdis et.al propose a regularizer term which promotes sparsity in matrix $\mathbf{H}$, to indirectly penalize the unnecessary activation of multiple columns of $\mathbf{W}_\mathcal{D}$. The regularizer is added to the original autoencoder loss function as follows:

$$\mathcal{L}'\left(\mathbf{V}, \widehat{\mathbf{V}}\right) = D\left(\mathbf{V}, \widehat{\mathbf{V}}\right) + \lambda \|\mathbf{H}\|_1. \tag{3.5}$$

The loss function $\mathcal{L}'$ increases whenever if the $L1$ norm of $\mathbf{H}$ is too large, therefore promoting solutions where $\mathbf{H}$ is sparse. The regularization parameter λ determines the influence of the regularization term in the total loss.

The resulting decoder weight matrix $\mathbf{W}_\mathcal{D}$, the code matrix $\mathbf{H}$ and the low-rank approximation matrix $\widehat{\mathbf{V}}$ using the regularized NAE model for the drums running example are shown in Figure 3.4. The parameter λ was chosen in order to increase the non-negativity percent of $\mathbf{W}_\mathcal{D}$ (Figure 3.4(a)), which also generates a matrix $\mathbf{H}$ closer to the ground truth annotations. However, when λ is increased further, the activations of Component 2, which still has non-negative values, would be deactivated in all frames.

The regularized NAE in [31] considers only examples where each of the components is isolated in the input spectrogram, and fails to account for the fact that there can be music examples where the three components are simultaneously active, so an sparsity constraint would still not be enough to enforce non-negativity in all elements of $\mathbf{W}_\mathcal{D}$. Additionally, the value of the regularization parameter λ would also become an additional parameter to optimize. Therefore, this method is not suitable for generating non-negative decoder weight matrices.

As an alternative approach, the work in [5] proposes directly constraining layer weights $\mathbf{W}_\mathcal{D}$

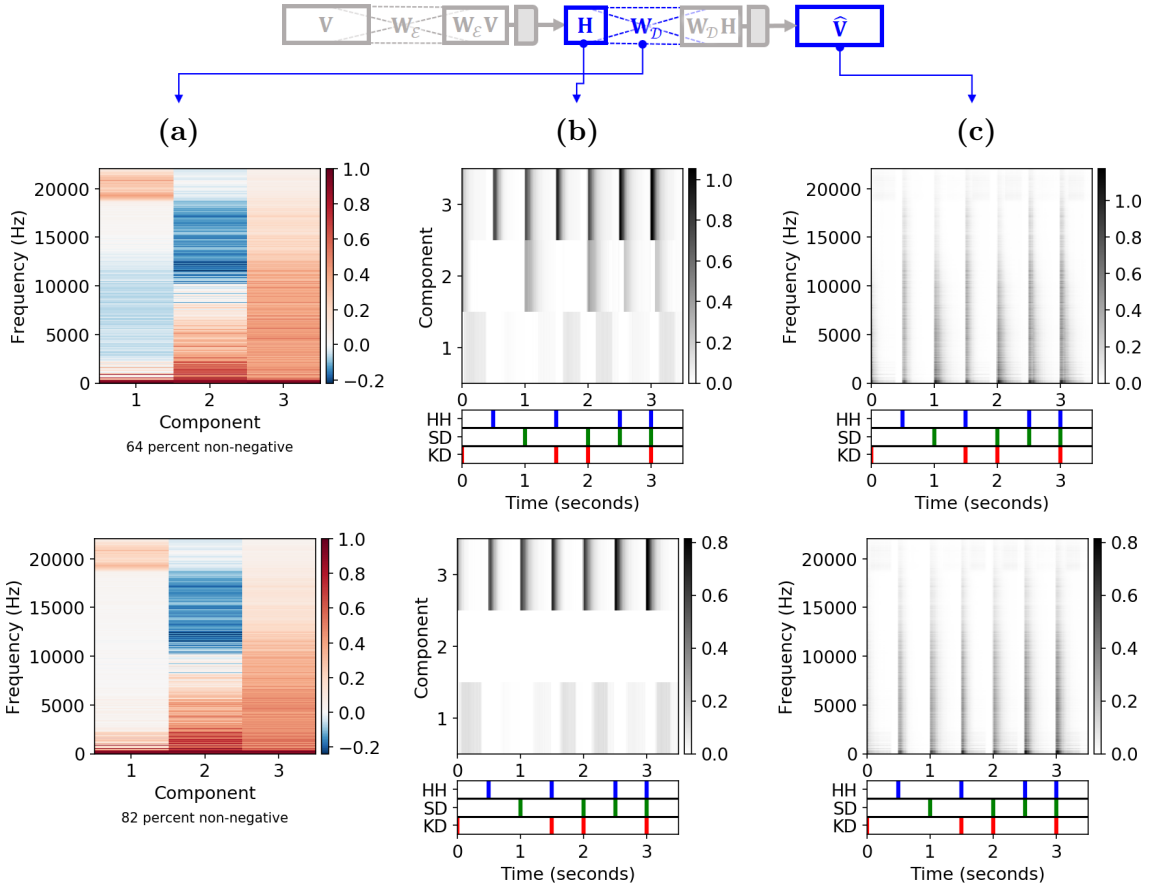

Figure 3.4. Regularized NAE output for the drums running example, with $\lambda = 20$ (above) and $\lambda = 21$ (below). **(a)** Learned decoder weight matrix $\mathbf{W}_{\mathcal{D}}$. **(b)** Learned code matrix $\mathbf{H}$ with reference ground truth annotations (below). **(c)** Low-rank approximation matrix $\widehat{\mathbf{V}}$ with reference ground truth annotations (below).

to be non-negative, claiming it would not only improve the performance of the network (as it reduces the range of values over which it optimizes), but also lead to learning more meaningful parameter values. Given the results in Figures 3.3 and 3.4, constraining the decoder weights matrix $\mathbf{W}_{\mathcal{D}}$ is a strategy worth applying if results closer to NMF are expected.

To directly enforce non-negativity in the decoder weights, it suffices to set all negative weights of $\mathbf{W}_{\mathcal{D}}$ to zero in each epoch during training. For the training process described by Equation (3.4), this would mean adding a step to the computation of $\mathbf{W}_{\mathcal{D}}^{(\ell+1)}$. Let $\mathbf{W}_{\mathcal{D}}^{(\ell+1)}(k,r)$ for $k \in [1:K]$, $r \in [1:R]$ denote the elements of matrix $\mathbf{W}_{\mathcal{D}}^{(\ell+1)}$. The non-negativity constraint can be applied using the following operation:

$$\mathbf{W}_{\mathcal{D}^+}^{(\ell+1)}(k,r) := \max\left(\mathbf{W}_{\mathcal{D}}^{(\ell+1)}(k,r), 0\right). \tag{3.6}$$

Constraining the decoder weights to be non-negative can be seen as using a rectifier function over

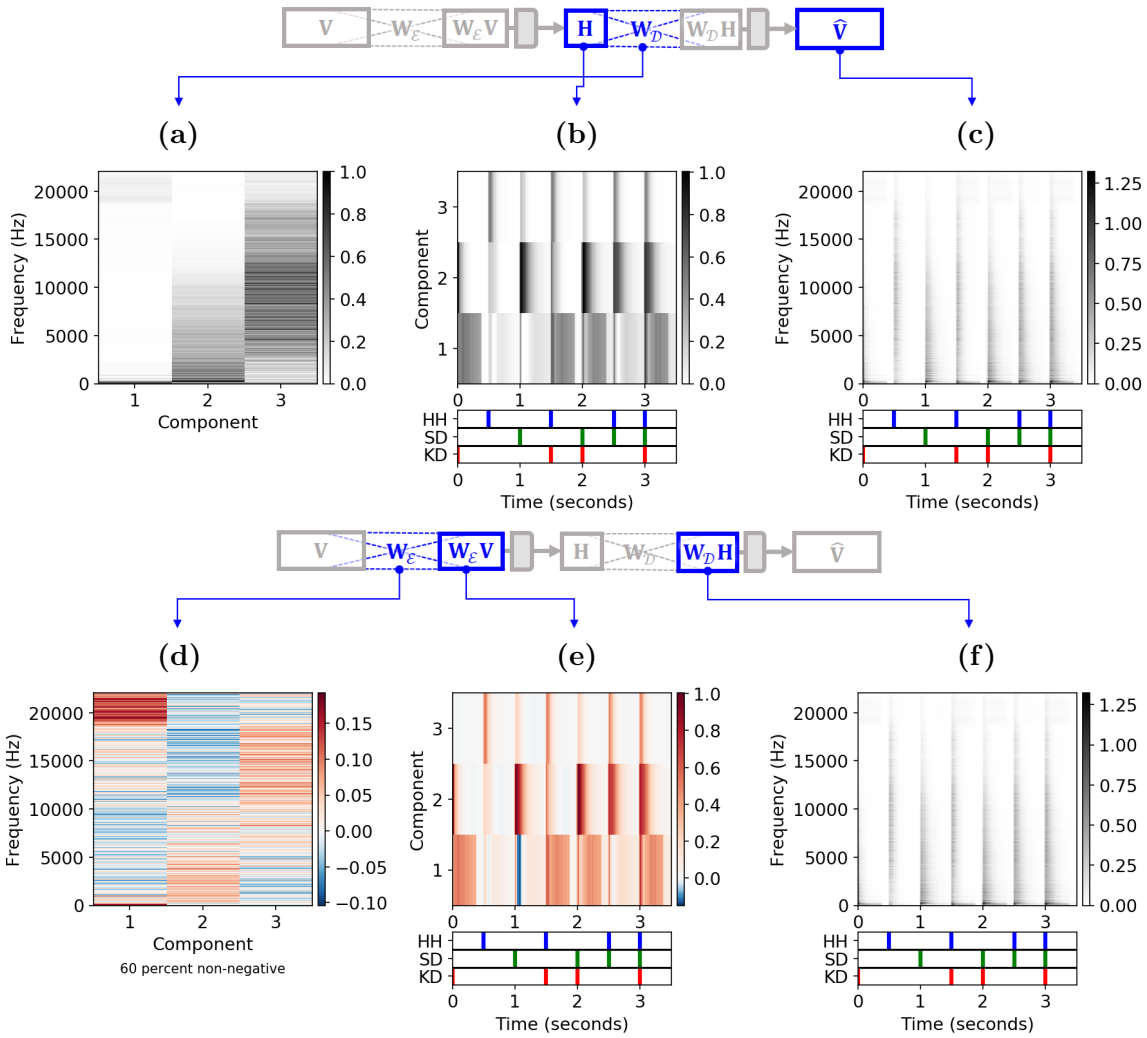

Figure 3.5. Randomly-initialized NAE with non-negative decoder, for the drums running example. **(a)** Learned non-negative decoder weight matrix $\mathbf{W}_{\mathcal{D}^+}$. **(b)** Learned code matrix $\mathbf{H}$, with reference ground truth annotations (below). **(c)** Low-rank approximation matrix $\widehat{\mathbf{V}}$, with reference ground truth annotations (below). **(d)** Learned encoder weight matrix $\mathbf{W}_\mathcal{E}^\top$. **(e)** Matrix $\mathbf{W}_\mathcal{E}\mathbf{V}$ with reference ground truth annotations (below). **(f)** Matrix $\mathbf{W}_{\mathcal{D}^+}\mathbf{H}$ with reference ground truth annotations (below).

its values. It can also be seen as projecting the elements of $\mathbf{W}_\mathcal{D}$ into the non-negative subspace. This modified training process is in fact close to projected gradient methods [21]. The superscript $+$ in $\mathbf{W}_{\mathcal{D}^+}$ will hereafter denote non-negative weight matrices. With this additional constraint, the decoder layer can be written as $\widehat{\mathbf{V}} = g_\mathcal{D}(\mathbf{W}_{\mathcal{D}^+} \cdot \mathbf{H})$. Since $\mathbf{W}_{\mathcal{D}^+}$ and $\mathbf{H}$ are non-negative, choosing a ReLU function for $g_\mathcal{D}$ would have no effect on the output, and a linear activation function could be chosen instead.

As a second experiment, a NAE architecture constraining the decoder weights to be non-negative is used to perform spectral decomposition on the drums running example. The trained decoder parameters $\mathbf{W}_{\mathcal{D}^+}$, the code matrix $\mathbf{H}$, and approximation $\widehat{\mathbf{V}}$ are shown in Figure 3.5. In this

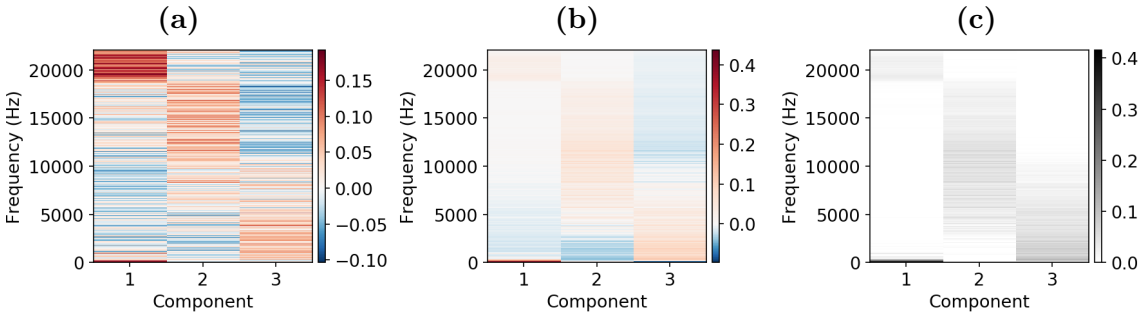

Figure 3.6. Comparison of the Encoder, pseudoinverse and decoder matrices from the randomly-initialized NAE with non-negative decoder for the drums running example. **(a)** Trained encoder layer weight matrix $\mathbf{W}_{\mathcal{E}}^{\top}$. **(b)** Decoder Moore-Penrose inverse $\left(\mathbf{W}_{\mathcal{D}^+}^{\dagger}\right)^{\top}$ **(c)** Trained non-negative decoder weight matrix $\mathbf{W}_{\mathcal{D}^+}$.

second experiment, both the decoder weights $\mathbf{W}_{\mathcal{D}^+}$ and the code matrix $\mathbf{H}$ in Figures 3.5(a) and 3.5(b) resemble the NMF factor matrices from Figure 2.4 more closely, with similar cross-talk artifacts between the components. The value range of matrix $\widehat{\mathbf{V}}$ is also closer to the value range of the original input spectrogram, as opposed to the low-rank approximation matrix in Figure 2.4(c).

To observe the training process of the NAE more closely, additional matrices are plotted in Figures 3.5(e), 3.5(f) and 3.5(g). Figure 3.5(e) shows matrix $\mathbf{W}_{\mathcal{E}}\mathbf{V}$, which corresponds to the code matrix $\mathbf{H}$ before the activation function $g_{\mathcal{E}}$. This matrix is scaled the same way as the code matrix in Figure 3.5(b), using Equation (2.14) according to the values of $\mathbf{W}_{\mathcal{D}^+}$. The matrix is almost entirely non-negative, showing that the autoencoder is compelled to output positive values, as the negative values are filtered out by the activation function. As noted earlier, matrix $\mathbf{W}_{\mathcal{D}}\mathbf{H}$ in Figure 3.5(f) is already non-negative, so a linear function for $g_{\mathcal{D}}$ can be used.

The most interesting matrix is the encoder weight matrix $\mathbf{W}_{\mathcal{E}}^{\top}$ in Figure 3.5(d). In the work by Smaragdis [31], matrix $\mathbf{W}_{\mathcal{E}}$ is regarded as a *pseudoinverse* of $\mathbf{W}_{\mathcal{D}^+}$. Given that $\mathbf{V} \approx \mathbf{W}_{\mathcal{D}^+}\mathbf{H}$, and solving for $\mathbf{H}$, then $\mathbf{H} \approx \mathbf{W}_{\mathcal{D}^+}^{\dagger}\mathbf{V}$, with $(\cdot)^{\dagger}$ being the Moore-Penrose inverse operator. Replacing $\mathbf{W}_{\mathcal{D}^+}^{\dagger}$ by $\mathbf{W}_{\mathcal{E}}$ would result in the encoder layer equation in (3.2), without taking the activation function into account. To take a closer look at the relation between $\mathbf{W}_{\mathcal{E}}$, $\mathbf{W}_{\mathcal{D}^+}^{\dagger}$ and $\mathbf{W}_{\mathcal{D}^+}$, the values for the three matrices are shown Figure 3.6.

The encoder matrix in Figure 3.6(a) resembles the decoder Moore-Penrose inverse in Figure 3.6(b), presenting a similar distribution of positive and negative values. Judging by this result, there is no motivation for enforcing non-negativity in the encoder weight matrix $\mathbf{W}_{\mathcal{E}}$. However, as it will be discussed in Section 3.2.3, the non-negative encoder reveals an interesting property of the NAE training process.

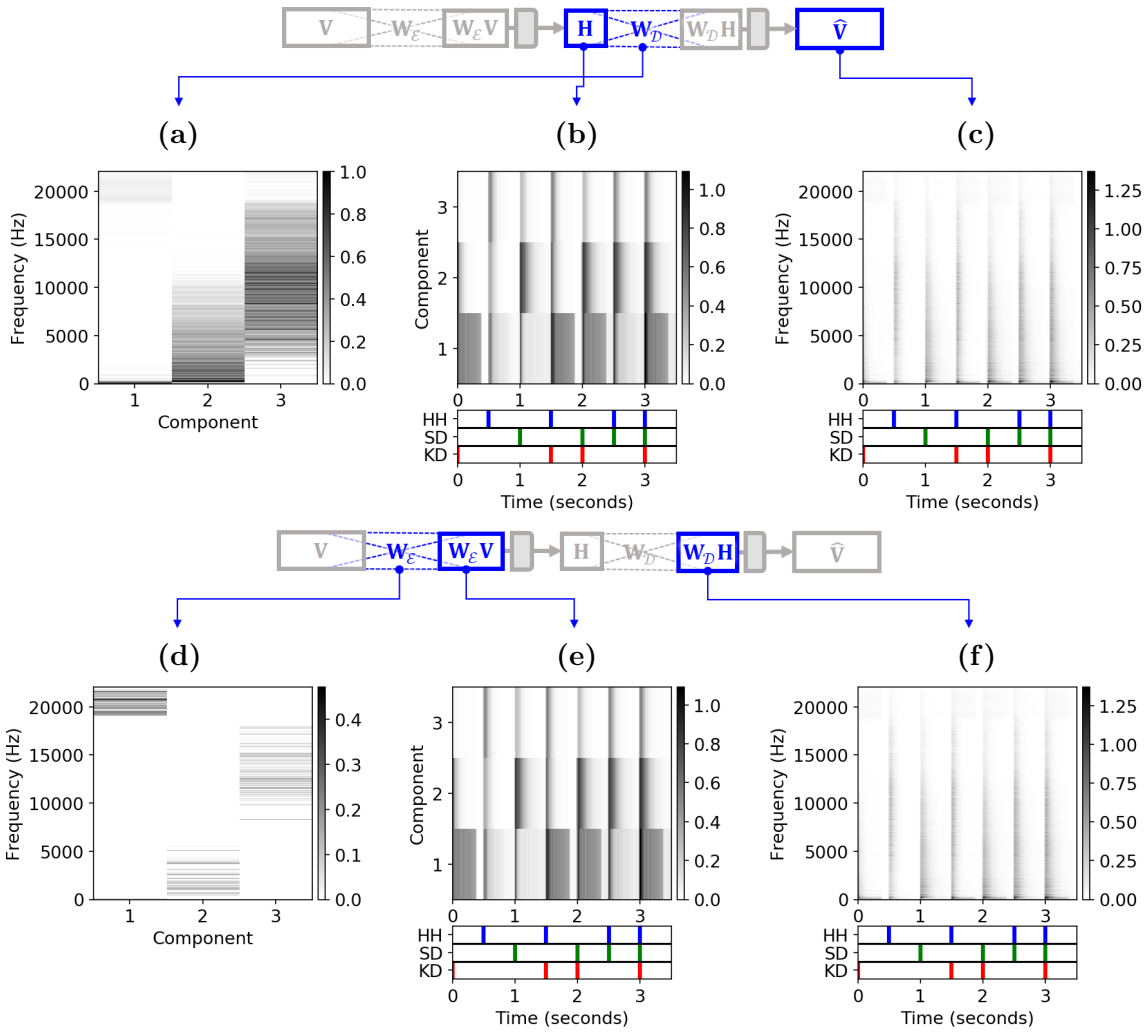

Figure 3.7. Randomly-initialized NAE with non-negative encoder and decoder, for the drums running example. **(a)** Learned non-negative decoder weight matrix $\mathbf{W}_{\mathcal{D}^+}$. **(b)** Learned code matrix $\mathbf{H}$, with reference ground truth annotations (below). **(c)** Low-rank approximation matrix $\widehat{\mathbf{V}}$, with reference ground truth annotations (below). **(d)** Learned non-negative encoder weight matrix $\mathbf{W}_{\mathcal{E}^+}^{\top}$. **(e)** Matrix $\mathbf{W}_{\mathcal{E}^+}\mathbf{V}$ with reference ground truth annotations (below). **(f)** Matrix $\mathbf{W}_{\mathcal{D}^+}\mathbf{H}$ with reference ground truth annotations (below).

3.2.3 NAE with Non-negative Encoder

Figure 3.7 shows the effect of using non-negative encoder and decoder weight matrices in the NAE. Now both matrices $\mathbf{W}_{\mathcal{D}}\mathbf{H}$ in Figure 3.7(f) and matrix $\mathbf{W}_{\mathcal{E}}\mathbf{V}$ in Figure 3.7(e) are now non-negative, meaning a linear function can be chosen for $g_{\mathcal{E}}$.

Matrix $\mathbf{W}_{\mathcal{E}^+}^{\top}$ in Figure 3.7(d) is now naturally sparse. There are no frequency overlaps in between its columns, meaning no element of one column is present in any of the others. This is a remarkable difference between the encoder and decoder weights, as frequency overlaps

should be allowed in the decoder weights $\mathbf{W}_{\mathcal{D}}$ to account for possible frequency overlaps between components.

In fact, the frequencies in each column of $\mathbf{W}_{\mathcal{E}^+}^{\top}$ coincide with the frequency content description of the drumset components back in Section 2.3.1: KD (Component 1) is identified by strong values in the lower part of the spectrum, and the high frequency artifact above 18 kHz; SD (Component 2) and HH (Component 3), by their presence in the low and medium range, respectively. The non-negative constrained encoder contains a set of frequencies that distinguishes a particular component from the rest. This means that each column $\mathbf{W}_{\mathcal{E}}$ is acting as a *matched-filter* [34]: For a given input matrix $\mathbf{V}$, the encoder outputs a high value whenever the frequency spectrum of a given frame matches that of a column in $\mathbf{W}_{\mathcal{E}^+}^{\top}$. This is how the code matrix $\mathbf{H}$ is generated.

Despite the evident differences between $\mathbf{W}_{\mathcal{E}^+}$ and $\mathbf{W}_{\mathcal{E}}$, the decoder weight matrix, the code and low-rank approximation matrices in Figures 3.7(a), 3.7(b) and 3.7(c) resemble those of the non-negative decoder NAE in Figure 3.5(a), 3.5(b) and 3.5(c). Therefore, Having a non-negative matrix $\mathbf{W}_{\mathcal{E}^+}$ does not have a negative impact on the spectral decomposition result, and instead represents an advantage in terms of convergence speed and interpretability, as originally stated in [5].

3.3 Score Information in NAE architectures

Figures 3.5 and 3.7 show that the spectral decomposition of NAEs with random initialization suffers from the same issues as the randomly initialized NMF results in Section 2.3.2. However, Section 2.3.3 has shown that the use of score information can lead to obtain more meaningful factor matrices. In neural networks, this problem is usually addressed by using a labeled data set for training. However, the only data the NAE uses for training is its own input matrix, and the score-informed matrices generated from ground truth annotations only provide coarse information on how the frequency spectrum or the activations of the input should be.

Using the same idea behind the multiplicative updates, introducing score information into the NAE models requires setting some of the values of the code matrix $\mathbf{H}$ and the decoder weight matrix $\mathbf{W}_{\mathcal{D}}$ to zero, and guarantee that they remain zero throughout the training of NAEs. The challenge is doing so for the optimization process of NAEs, which uses no multiplicative updates.

3.3.1 Score-informed Weight Matrices

Introducing score information in the decoder weight matrix $\mathbf{W}_{\mathcal{D}}$ can be done by simply initializing its values with a score-informed $\mathbf{W}_{\mathcal{D}}^{(0)}$ instead of using a random distribution. Figure 3.8 shows the effect of an informed initialization of the NAE for the piano example, using the scored-informed

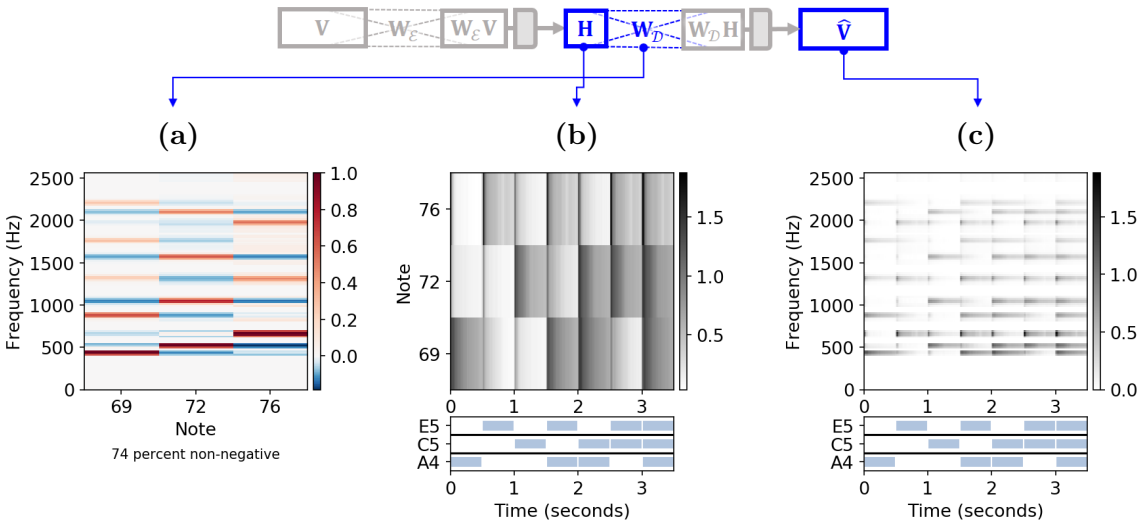

Figure 3.8. NAE with informed initialization, for the drums running example. **(a)** Trained decoder weight matrix $\mathbf{W}_\mathcal{D}$ with informed initialization. **(b)** Learned code matrix $\mathbf{H}$, with reference ground truth annotations (below). **(c)** Low-rank approximation matrix $\widehat{\mathbf{V}}$, with reference ground truth annotations (below).

matrix in Figure 2.5(a) as $\mathbf{W}_\mathcal{D}^{(0)}$. The informed initialization also brings the code matrix $\mathbf{H}$ in Figure 2.5(b) closer to the NMF activation matrix in Figure 2.5(d) and to the ground truth annotations. However, there is nothing in the NAE model that fixes the zero-valued coefficients of $\mathbf{W}_\mathcal{D}$ after being initialized as such. Therefore, the decoder weight matrix $\mathbf{W}_\mathcal{D}$ contains some values (mostly negative) outside the harmonic regions.

When enforcing non-negativity on the decoder weight matrix $\mathbf{W}_\mathcal{D}$ in Section 3.2.2, all the negative values of matrix $\mathbf{W}_\mathcal{D}$ were set to zero *in each iteration*, by adding the non-negative projection step in Equation (3.6) to the gradient descent update in (3.4). Following the same idea, one additional step can be added to the NAE gradient descent update, this time to constrain the values of $\mathbf{W}_\mathcal{D}$ according to the information in $\mathbf{W}_\mathcal{D}^{(0)}$. This can be done by generating a binary mask from the values of $\mathbf{W}_\mathcal{D}^{(0)}$ to filter out the values of $\mathbf{W}_\mathcal{D}$ every iteration.

Let $\mathbf{W}_\mathcal{D}^{(0)}(k,r)$ for $k \in [1:K]$, $r \in [1:R]$ denote the elements of matrix $\mathbf{W}_\mathcal{D}^{(0)}$, and let $\mathbf{M}_\mathbf{W}(k,r)$ be the elements of the binary mask $\mathbf{M}_\mathbf{W} \in \mathbb{R}^{K\times R}$ given by:

$$\mathbf{M}_\mathbf{W}(k,r) = \begin{cases} 1, & \text{if } \mathbf{W}_\mathcal{D}^{(0)}(k,r) > 0, \\ 0, & \text{if } \mathbf{W}_\mathcal{D}^{(0)}(k,r) = 0. \end{cases} \tag{3.7}$$

The generated mask is multiplied element-wise with $\mathbf{W}_\mathcal{D}$ in each iteration, setting to zero all values of $\mathbf{W}_\mathcal{D}$ that are not part of the score-informed frequency template. Note that the masked values $\mathbf{W}'_\mathcal{D} = \mathbf{W}_\mathcal{D} \odot \mathbf{M}_\mathbf{W}$ are not necessarily non-negative. It is possible to enforce both constraints during training by sequentially computing the informed constraint step, followed by

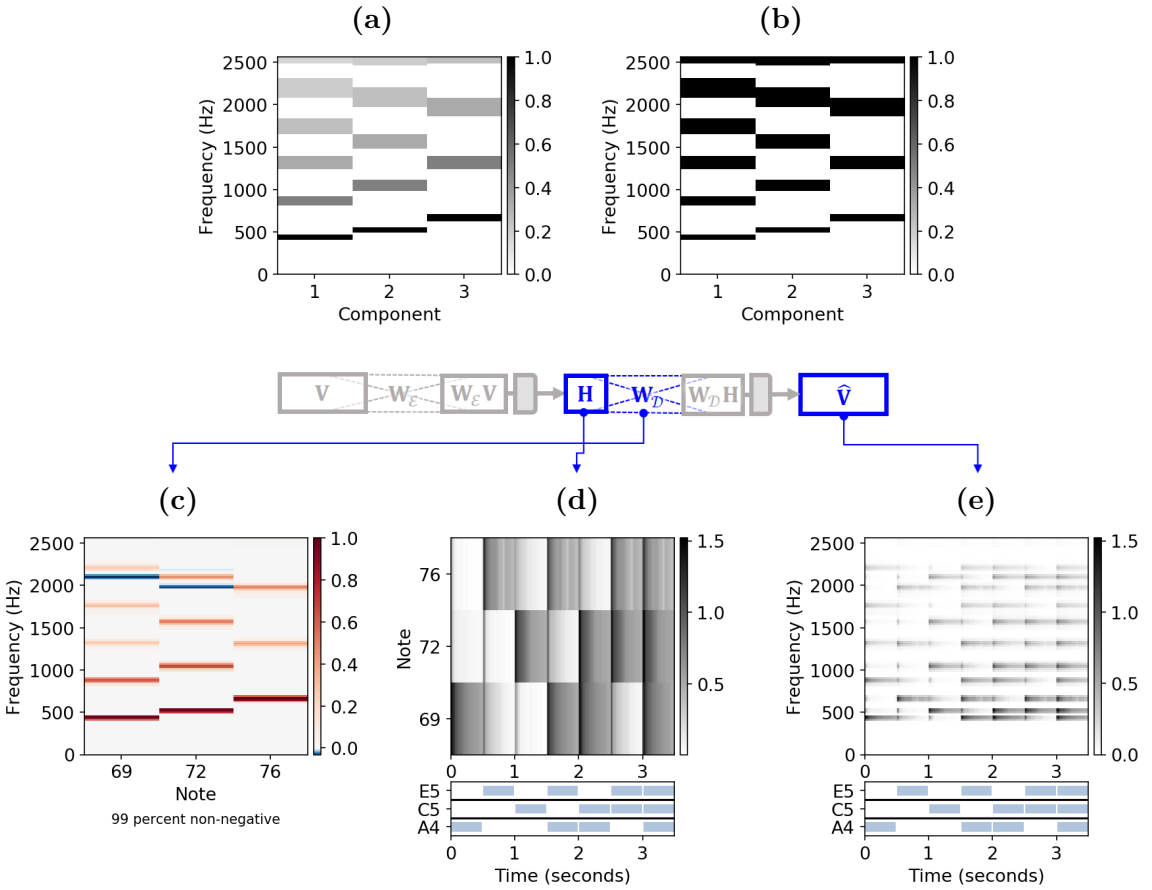

Figure 3.9. NAE with score-informed constrained decoder, for the piano running example. **(a)** Score-informed template matrix $\mathbf{W}^{(0)}$ from Figure 2.5(a). **(b)** Generated binary mask $\mathbf{M}_{\mathbf{W}}$. **(c)** Trained decoder weight matrix $\mathbf{W}'_{\mathcal{D}} = \mathbf{W}_{\mathcal{D}} \odot \mathbf{M}_{\mathbf{W}}$. **(d)** Learned code matrix $\mathbf{H}$, with reference ground truth annotations (below). **(e)** Low-rank approximation matrix $\widehat{\mathbf{V}}$, with reference ground truth annotations (below).

the non-negativity constraint step. To the author's knowledge, this technique has not been used before to introduce musical score information into NAE models.

Figure 3.9 shows the use of informed constraints in the decoder weight matrix $\mathbf{W}_{\mathcal{D}}$ for the piano example. The binary mask $\mathbf{M}_{\mathbf{W}}$ in Figure 3.9(b) is generated from the informed template matrix $\mathbf{W}^{(0)}$ in Figure 3.9(a) using Equation (3.9). The trained matrix $\mathbf{W}'_{\mathcal{D}}$ in Figure 2.5(c) is almost entirely non-negative, and shows high positive values in the harmonic regions similar to the NMF template matrix in Figure 2.5(c), although it still presents negative values in the high frequency harmonics of two of its templates. The use of the non-negative constraint would get rid of these negative values and output the results in Figure 2.5. As discussed in Section 2.3.3, the piano example requires a more sophisticated initialization approach to include onset information. For more information, the reader can refer to Appendix A.

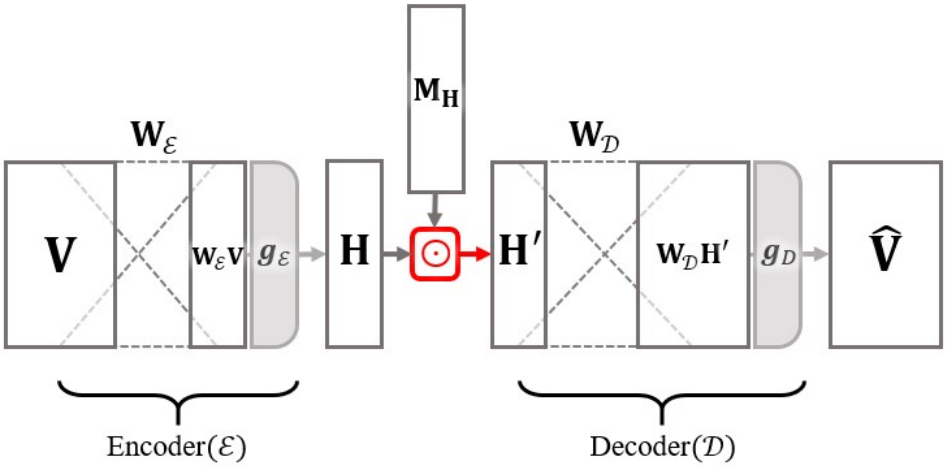

Figure 3.10. Overview of a shallow autoencoder architecture with structured dropout layer, as proposed in [13].

3.3.2 Score-Informed Code Matrices

The code Matrix $\mathbf{H}$ is the output of the encoder stage, but not a proper training parameter. Therefore, the procedure described in Section 3.3.1 cannot be used to generate score-informed activations.

Figures 3.2 and 3.3 show that randomly-initialized, unconstrained NAEs are able to generate a low-rank approximation matrix $\widehat{\mathbf{V}}$ that resembles the input matrix $\mathbf{V}$, but fail at generating a musically meaningful code matrix. This is to be expected, since the network was trained using a single spectrogram and no additional information was introduced in the training process to meet this condition. In this context, regularization is a way of guiding the DNN learning process, by giving preference to a certain solution over another [14]. Taking regularization as a starting point, Ewert et.al [13] propose two strategies to generate score-informed code matrices: Structured dropout and a regularizer. These strategies will be discussed in detail in this section.

As a first strategy, the work in [13] proposes the addition of a non-trainable layer between the encoder and decoder stages. The new layer will disable certain encoder weights by element-wise multiplication of the code matrix $\mathbf{H}$ with a binary mask $\mathbf{M_H} \in \mathbf{R}^{R \times M}$. This binary would contain the score information regarding the activity of the sound sources over time e.g. the score-informed $\mathbf{H}^{(0)}$ in Section 2.3.3. The modified NAE architecture is shown in Figure 3.10.

This strategy can be compared to stochastic dropout, which consists of adding an intermediate, non-trainable layer which sets a percentage of its output values to zero [27]. The disabled outputs are randomly selected at each iteration. In contrast, the proposed modification in [13] works as a form of *structured* or deterministic dropout, since the disabled connections are fixed by the zero values of matrix $\mathbf{M_H}$ throughout the entire training process. During back propagation, the

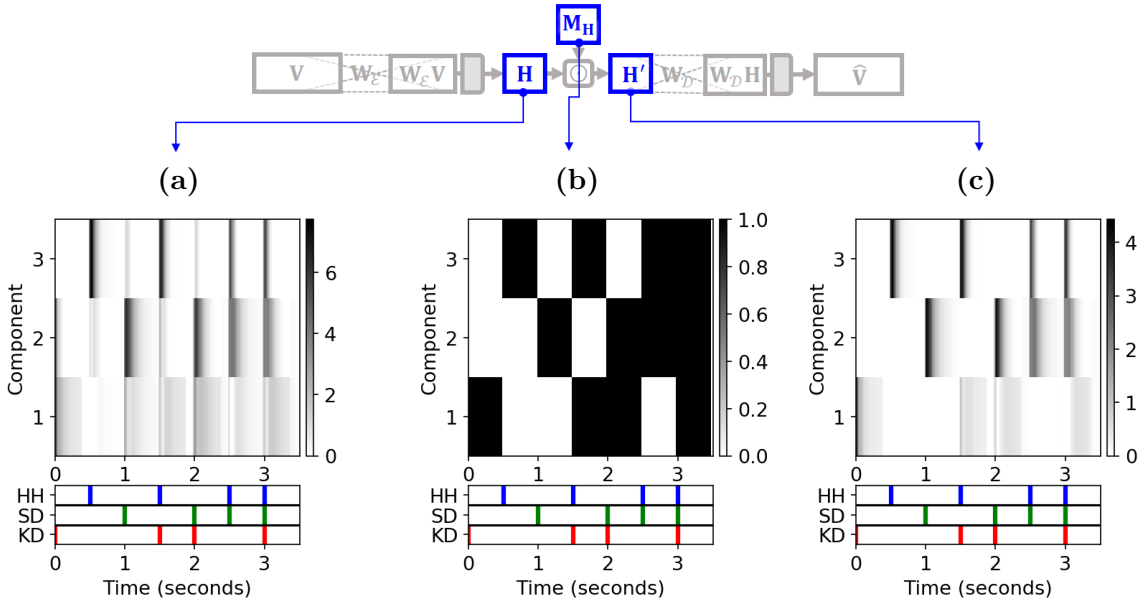

Figure 3.11. Structured dropout layer matrices, with reference ground truth annotations (below). **(a)** Code Matrix $\mathbf{H}$. **(b)** Binary mask $\mathbf{M_H}$ generated from ground truth annotations. **(c)** Matrix $\mathbf{H}' = \mathbf{H} \odot \mathbf{M_H}$.

error information will be focused on the weights of the encoder matrix $\mathbf{W}_\mathcal{E}$, which are directly linked to the non-zero values of $\mathbf{M_H}$. The resulting matrix $\mathbf{H}' = \mathbf{H} \odot \mathbf{M_H}$ can be interpreted as a score-informed activation matrix in the NMF context.

Figure 3.11 shows the effect of the structured dropout layer on the NAE code matrix for the drums example. The mask $\mathbf{M_H}$ is the same score-informed activation matrix $\mathbf{H}^{(0)}$ used to compute the NMF factor matrices in Figure 2.5(b). Matrix $\mathbf{H}'$ in Figure 3.7(c) shows no cross-talk in between the different templates, compared to matrix $\mathbf{H}$ in Figure 3.7(a). As a result, matrix $\mathbf{H}'$ coincides with the ground truth annotations and with the score-informed NMF activation matrix in Figure 2.6, showing similar cross-talk patterns as the ones described in Section 2.3.3.

The second strategy used to introduce score information in NAEs is using a regularizer. Given the score-informed matrix $\mathbf{M_H}$, a regularizer can be added to the original autoencoder loss function in the following way:

$$\mathcal{L}'\left(\mathbf{V}, \widehat{\mathbf{V}}\right) = D\left(\mathbf{V}, \widehat{\mathbf{V}}\right) + \lambda \|(1 - \mathbf{M_H}) \odot \mathbf{H}\|_2^2, \tag{3.8}$$

where the $(1 - \mathbf{M_H}) \odot \mathbf{H}$ term makes the loss function increase if the network presents high values in $\mathbf{H}$ that coincide with the zero-valued regions of the binary mask $\mathbf{M_H}$.

Figure 3.12 shows how including the regularization term in the loss function affects the NAE output matrices for the drums running example. Notice that the weight matrix $\mathbf{W}_\mathcal{D}$ in Figure 3.12(a) is entirely non-negative despite the fact that there is no constraint enforced on its values. The code

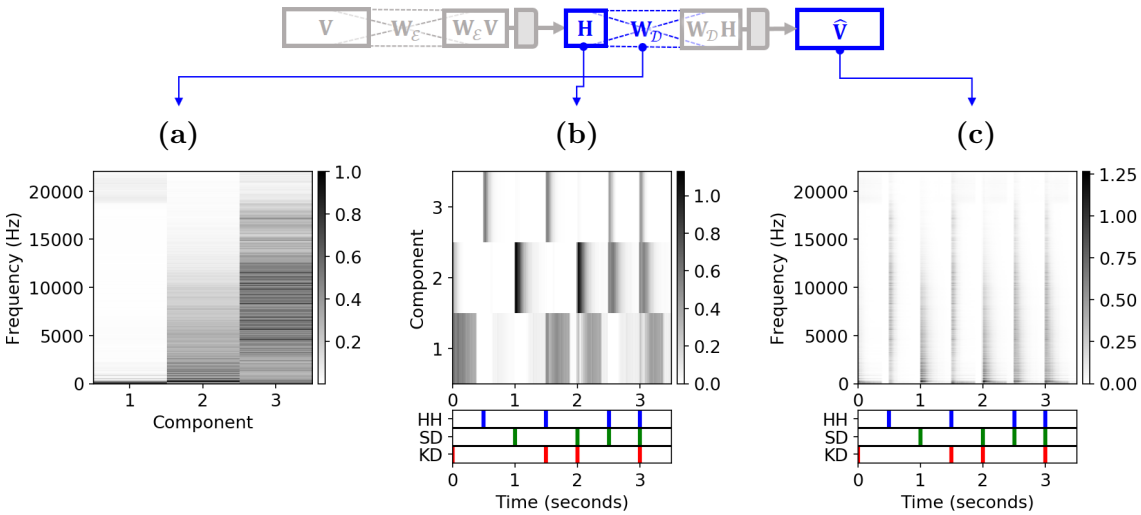

Figure 3.12. NAE with regularized loss function (Eq. 3.8) with $\lambda = 40$, for the drums running example. **(a)** Learned decoder weight matrix $\mathbf{W}_{\mathcal{D}}$. **(b)** Learned code matrix $\mathbf{H}$, with reference ground truth annotations (below). **(c)** Low-rank approximation matrix $\widehat{\mathbf{V}}$, with reference ground truth annotations (below) .

matrix in Figure 3.12(b) is the same as matrix $\mathbf{H}'$ in Figure 3.10(c).

To get such results, the parameter λ must usually be set to a large value, dominating over the divergence term when applying gradient descent, and potentially causing the training process to diverge. This is an important disadvantage, as the values of λ would have to be adapted during training. Nonetheless, according to [13] the combination of the structured dropout layer and the regularization term in Equation (3.8) reduces the number of epochs needed for convergence, while allowing a more flexible setting of λ. This issue is further discussed in Appendix C.

Given the results of both score information strategies, structured dropout seems to be a more effective way of generating score-informed NAEs, as there is no need of tuning an additional regularization parameter.

3.4 Non-negative Convolutional Autoencoders (CAEs)

As discussed in Section 2.4, NMF can be extended to include the temporal evolution of the frequency content for each of the sound sources when performing spectral decomposition. This extension motivated the formulation of NMFD, which uses a convolution operation to allow the use of multiple template matrices for multiple frames. In this section, a NAE approach to the NMFD algorithm is considered.

In [34], Venkataramani et.al. propose a shallow autoencoder model inspired on NMFD, where the fully connected layers of the encoder and decoder are replaced by *convolutional* layers.

Convolutional layers replace the layer weights by a set of convolution filters, also called kernel, which is convolved with the input matrix. Convolution in DNN models is usually performed along both dimensions of the input matrix. When convolution is done along a single matrix dimension, it is called a 1D convolution operation. This operation is a particular case of 2D convolution where one of the dimensions of the filter is the same as the dimension of the input matrix.

Formally, given a convolutional kernel $\underline{\mathbf{W}} \in \mathbb{R}^{K \times R \times T}$, made up of T filters $\mathbf{W}_\tau \in \mathbb{R}^{K \times R}$ for $\tau \in [1, T]$ (abbreviated $T@K \times R$), and an input matrix $\mathbf{H} \in \mathbb{R}^{R \times M}$, a 1D convolutional layer would perform the following operation [2]:

$$\underline{\mathbf{W}} * \mathbf{H} = \sum_{\tau=1}^{T} \mathbf{W}_\tau \overset{\tau\rightarrow}{[\mathbf{H}]}. \tag{3.9}$$

Dimension R is common to both $\mathbf{W}_\tau$ and $\mathbf{H}$, so the convolution is performed along M, outputting a matrix of size $K \times M$. Equation (3.9) is the same as the NMFD convolutive operation in Equation (2.16), so the convolution is performed along the frame dimension, and its output would be the approximation matrix $\widehat{\mathbf{V}}$.

The Convolutional autoencoder (CAE), as proposed in [34], is described by the following equations:

$$\begin{aligned} &1.\ \text{Encoder: } \mathbf{H} = g_{\mathcal{E}}\left(\sum_{\tau=1}^{T} (\mathbf{W}_{\mathcal{E}})_\tau \overset{\tau\rightarrow}{[\mathbf{V}]}\right), \\ &2.\ \text{Decoder: } \widehat{\mathbf{V}} = g_{\mathcal{D}}\left(\sum_{\tau=1}^{T} (\mathbf{W}_{\mathcal{D}})_\tau \overset{\tau\rightarrow}{[\mathbf{H}]}\right). \end{aligned} \tag{3.10}$$

As noted earlier, the decoder layer in Equation (3.10) performs the NMFD low-rank approximation operation, meaning $\underline{\mathbf{W}}_{\mathcal{D}}$ would correspond to the NMFD template matrix tensor, and $\mathbf{H}$ to the activation matrix. For spectral decomposition, the CAE might also benefit from the score information strategies shown in Sections 3.3.1 and 3.3.2.

Additionally, as learned in Section 3.2.3, the encoder filters in $\underline{\mathbf{W}}_{\mathcal{E}}$ would be seen as inverse filters of $\underline{\mathbf{W}}_{\mathcal{D}}$. Their interpretation would be similar to that of the encoder weight matrix $\mathbf{W}_{\mathcal{E}}$ for the NAE case, namely, a matched filter that identifies the time instants where the learned pattern is present in the input matrix. Table 3.4 shows a summary of the CAE architecture, which includes the intermediate structured dropout layer described in Section 3.3.2.

Figures 3.13 and 3.14 show the resulting factor matrices of a score-informed CAE architecture with non-negative decoder, and a CAE with non-negative encoder and decoder, for the drums running example. For the convolutional layers, the number of templates was set to $T = 5$ as in

[2] Refer to Appendix B for more information on the 1D convolution operation.

Layer	**Output Shape**	**Parameters**
Input	$(1, K, M)$	
Encoder ($\mathcal{E}$)		
Conv1D + ReLU	$(1, R, M)$	$T@R \times K$
Structured Dropout	$(1, R, M)$	
Decoder ($\mathcal{D}$)		
Conv1D + ReLU	$(1, K, M)$	$T@K \times R$

Table 3.2. CAE network architecture, with structured dropout layer. The first dimension of the output shape column values indicates the network is trained using a full-batch training approach.

the NMFD implementation, to be able to compare the CAE matrices to the NMFD results in Figure 2.8.

The decoder weight matrix $\underline{\mathbf{W}}_{\mathcal{D}^+}$ of in Figures 3.13(a) and 3.14(a) resemble the NMFD templates $\underline{\mathbf{W}}$ in Figure 2.8(a). However, all $\mathbf{U}$ component matrices present a gap in their frequency content around 17 kHz. The gap is also noticeable in the reconstructed spectrogram of Figure 3.13(c). This reconstruction artifact is not observed in the NMFD decomposition, and its presence might imply that the CAE requires more than $L = 1000$ epochs. The value range of the approximation matrix in Figure 3.13(c) also differs from the one of the original spectrogram (see Figure 2.2(c)).

Matrices $\underline{\mathbf{W}}_{\mathcal{E}} * \mathbf{V}$ (matrix $\mathbf{H}$ before the activation function $g_{\mathbf{E}}$) in Figures 3.13(e) and 3.14(e), illustrate how potential cross-talk values are set to zero or negative values, that are later masked out by the activation function and matrix $\mathbf{M}_{\mathbf{H}}$. In particular, Figure 3.14(e) resembles the uniform NMF activation matrix $\mathbf{H}$ from Figure 2.4(b). After the structured dropout layer, the code matrices $\mathbf{H}$ in Figures 3.13(b) and Figure 3.14(b) coincide with the ground truth annotations and Figure 2.8(d).

The code matrix in Figure 3.14(b) is in fact close to an ideal activation matrix, as it has none of the cross-talk artifacts observed in previous NMFD or NAE results, and shows strokes of uniform amplitude and duration. This result implies that having a non-negative encoder matrix $\underline{\mathbf{W}}_{\mathcal{E}^+}$ such as the one observed in Figure 3.14(d) might contribute to obtaining better sound decomposition results, compared to the unconstrained encoder in Figure 3.13(d). For a quantitative comparison between such approaches, refer to Section 4.3.4.

Finally, it is worth mentioning that the 1D-convolution operation performed by the Tensorflow package [1] does not exactly coincide with Equations in (3.10), so some practical issues were encountered during the implementation of the CAE. A detailed explanation of these issues, among other considerations, can be found in Appendix B.

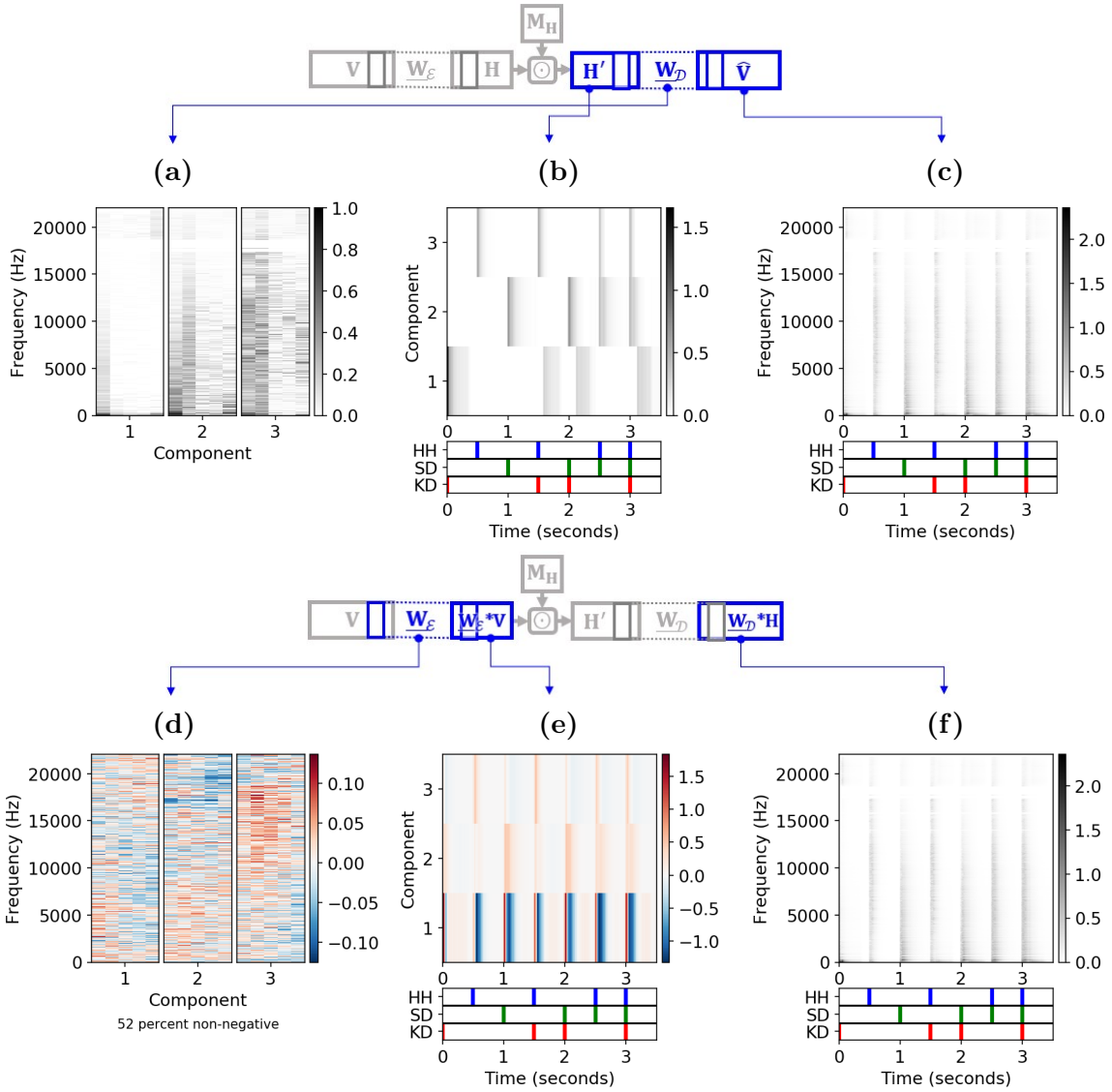

Figure 3.13. Score-informed CAE with non-negative decoder, for the drums running example. **(a)** Learned decoder matrices $\underline{\mathbf{W}}_{\mathcal{D}^+}$. **(b)** Learned code matrix $\mathbf{H}$, with reference ground truth annotations (below). **(c)** Low-rank approximation matrix $\widehat{\mathbf{V}}$, with reference ground truth annotations (below). **(d)** Learned encoder matrices $\underline{\mathbf{W}}_{\mathcal{E}}$. **(e)** Matrix $\underline{\mathbf{W}}_{\mathcal{E}} * \mathbf{V}$ with reference ground truth annotations (below). **(f)** Matrix $\underline{\mathbf{W}}_{\mathcal{D}^+} * \mathbf{H}$ with reference ground truth annotations (below).

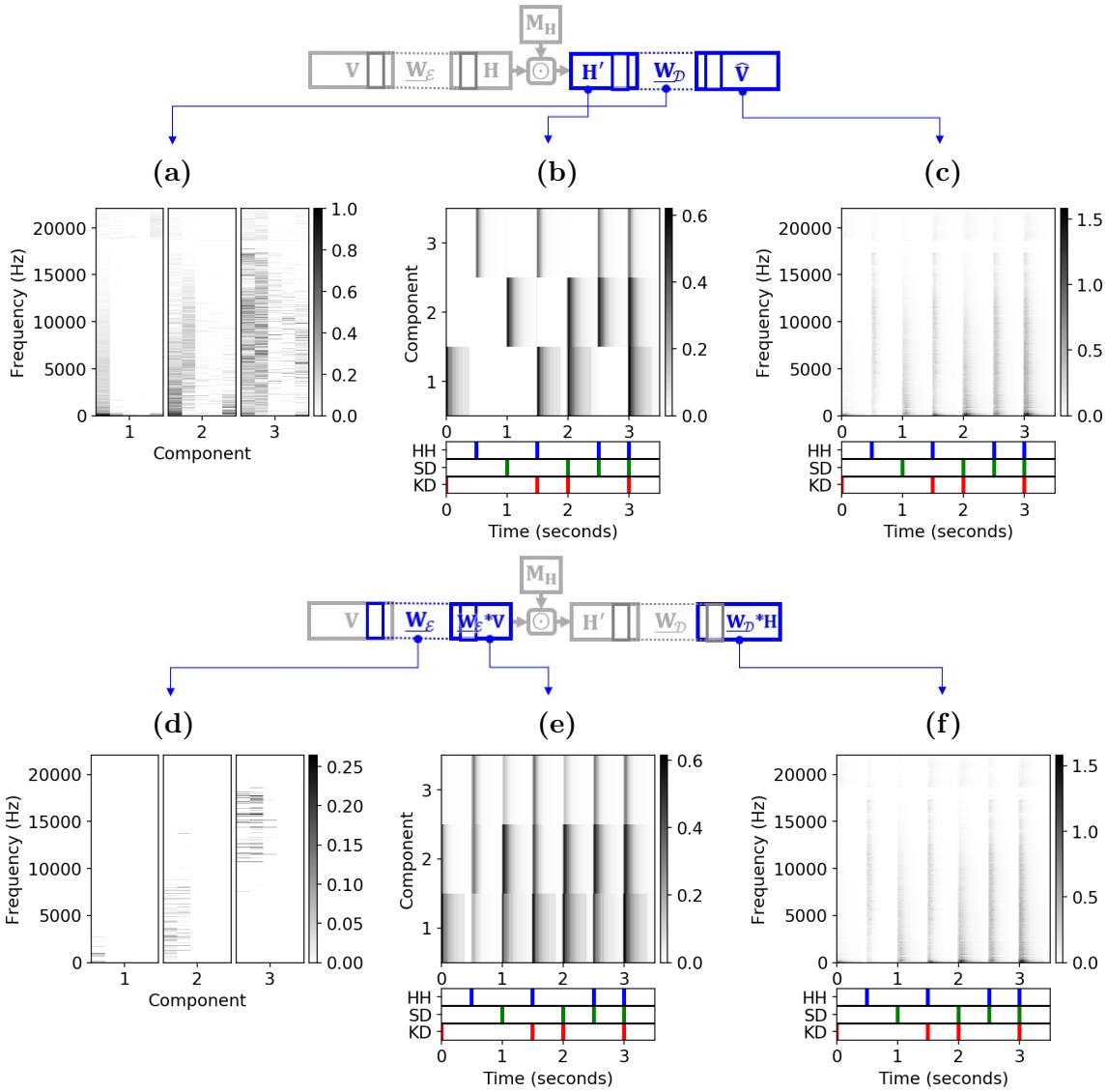

Figure 3.14. Score-informed CAE with non-negative encoder and decoder, for the drums running example.**(a)** Learned decoder matrices $\underline{\mathbf{W}}_{\mathcal{D}^+}$. **(b)** Learned code matrix $\mathbf{H}$, with reference ground truth annotations (below). **(c)** Low-rank approximation matrix $\widehat{\mathbf{V}}$, with reference ground truth annotations (below). **(d)** Learned encoder matrices $\underline{\mathbf{W}}_{\mathcal{E}^+}$. **(e)** Matrix $\underline{\mathbf{W}}_{\mathcal{E}^+} * \mathbf{V}$ with reference ground truth annotations (below). **(f)** Matrix $\underline{\mathbf{W}}_{\mathcal{D}^+} * \mathbf{H}$ with reference ground truth annotations (below).

Chapter 4

Drum Sound Decomposition Evaluation

This chapter presents a quantitative comparison between the several of the the NAE models described in Chapter 3, and their NMF counterparts from Chapter 2, in order to formulate conclusions about their performance and suitability for sound decomposition.

This Chapter is organized as follows: Section 4.1 briefly describes the dataset used for performing sound decomposition evaluation. Section 4.2 describes the overall signal reconstruction pipeline use to generate the estimated sound source signals from their magnitude spectrograms. Finally, Section 4.3 presents and analyses the sound decomposition evaluation results obtained.

4.1 The Dataset

The spectral decomposition evaluation was performed on the publicly available dataset "IDMT-SMT-Drums" [8]. The dataset consists of 64 drum tracks of 14-17 second duration on a 16-bit PCM WAV format, sampled at 44.1 kHz. The dataset contains tracks made up by the three core components (KD, SD and HH) recorded with different types of drumkits and performing different drum patterns. This means there is variety in both the spectral content of the sound sources and their activity through time.

Besides the mixture signal (MIX), Each drum track includes a perfectly isolated KD, SD and HH track. Annotations for all recordings are available for all MIX tracks in a MusicXML format[1]. The MIX audio signal was used as input of the drum sound decomposition algorithm, described in Section 4.2.

[1]More information at https://www.musicxml.com

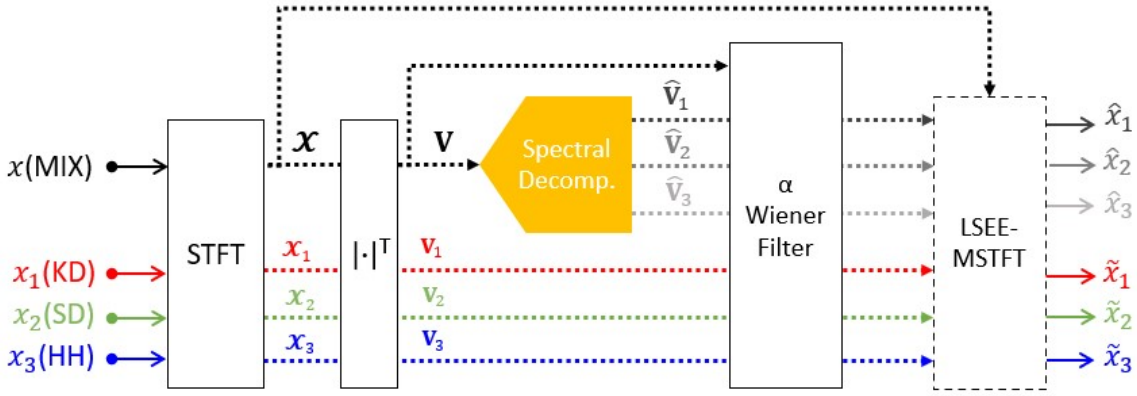

Figure 4.1. Diagram of the sound decomposition pipeline used for the signal reconstruction experiments. The spectral decomposition block is highlighted as the heart of the sound decomposition process.

4.2 The Sound Decomposition Pipeline

As explained in Section 2.3, spectral decomposition is the core part of the sound decomposition process, but not its only step. This section briefly describes the sound decomposition pipeline, closely following the work by Dittmar et.al [9].

Figure 4.1 shows the overall process followed to perform sound decomposition on an input mixture signal x (MIX, presented with a black arrow in Figure 4.1). From the dataset, we know that x is made up by $R = 3$ sound sources (KD, SD , HH). The goal of the sound decomposition pipeline is to generate estimate component signals $\hat{x}_r$ for $r \in [1 : R]$ using only the information from the input signal x.

As done with the running example signals in Chapters 2 and 3, the first step is to generate a time-frequency representation $\mathcal{X}$ from signal x using STFT, following Equation (2.9). This representation can be written in exponential form as $\mathcal{X} = \mathbf{V}^\top \odot \exp(i\mathbf{\Phi})$. Matrix $\mathbf{\Phi}$ corresponds to the phase information or phase spectrogram of $\mathcal{X}$, with elements $\mathbf{\Phi}(m, k) \in [0, 2\pi)$ for $m \in [1 : M], k \in [1 : K]$. The phase information in $\mathbf{\Phi}$ will be used in a later stage of the reconstruction process.

From $\mathcal{X}$, the magnitude spectrogram $\mathbf{V} = |\mathcal{X}|^\top$ is obtained. Matrix $\mathbf{V}$ is the input of the spectral decomposition algorithm (either NMF, NMFD, or a NAE or CAE model). The low rank approximation matrix $\widehat{\mathbf{V}}_r$ for a particular sound source is calculated following Equation (2.11) for NMF and NAEs, and Equation (2.20) for the NMFD and CAE convolutional approaches.

As already discussed in Section 2.3, a time domain signal must be generated from the low-rank approximation matrices. To achieve this, signal reconstruction techniques must be used. However, because the matrices are a low-rank approximations, not all spectral information of the original spectrogram is captured. To cope with this, a mask $\mathbf{M}_r$ can be generated from the low-rank

approximation, to be applied over the original spectrogram $\mathbf{V}$. This mask is computed as follows:

$$\mathbf{M}_r := \left(\widehat{\mathbf{V}}_r\right)^{\alpha} \oslash \left(\sum_{r=1}^{R} \left(\widehat{\mathbf{V}}_r\right)^{\alpha}\right). \tag{4.1}$$

The mask $\mathbf{M}_r$ represents the contribution of the low-rank approximation matrix of component $r \in [1 : R]$ to the input signal $\mathbf{V}$. The parameter $0 \leq \alpha \leq 2$ controls the balance between suppression of other components and reconstruction artifacts. The power operation $(\cdot)^{\alpha}$ is applied element-wise.

The final component magnitude spectrogram $\widehat{\mathbf{V}}_r$ is calculated by setting

$$\widehat{\mathbf{V}}_r = \mathbf{V} \odot \mathbf{M}_r. \tag{4.2}$$

This ratio-masking procedure is also called Alpha Wiener filter [22].

Since the spectral decomposition is performed over the magnitude spectrogram, there is no component phase information to transform the spectrogram back to the time domain. However, it is common practice to use the mixture phase information from $\mathcal{X}$, i.e. $\widehat{\mathcal{X}}_r = \widehat{\mathbf{V}}_r^{\top} \odot \exp(i\mathbf{\Phi})$. Having $\widehat{\mathcal{X}}_r$, the estimated time domain signal $\hat{x}_r$ is computed using the reconstruction method of LSEE-MSTFT [15]. This method computes a time domain signal from a modified spectrogram (MSTFT stands for modified STFT), using the same forward STFT parameters (w,N,H). Signal $\hat{x}_r$ is generated by computing an inverse DFT of spectrogram for each spectral frame. This generates a set of time domain signals $y_m(n)$ for $m \in [1 : M]$ calculated by:

$$y_m(n) := \begin{cases} \frac{1}{N} \sum_{k=0}^{N} \widehat{\mathcal{X}}_r(m,k) \exp(2\pi i k n/N) & \text{if } n \in [1 : N], \\ 0 & \text{otherwise.} \end{cases}$$

The Least squares error estimation (LSEE) to obtain the time domain estimate $\hat{x}_r$ is computed as follows:

$$\hat{x}_r(n) := \frac{\sum_{m\in\mathbb{Z}} y_m(n - mH) w(n - mH)}{\sum_{m\in\mathbb{Z}} w(n - mH)^2},\ n \in \mathbb{Z}.$$

In parallel, the perfectly isolated tracks x_r for KD, SD and HH are also processed in the same way as the MIX signal (STFT, magnitude spectrogram, Wiener Filter and LSEE-MSTFT), except that they do not go through the spectral decomposition process. The resulting signals $\tilde{x}_r$ are reconstructions from the isolated tracks that include the artifacts and distortions introduced during the reconstruction process. These signals are used as Oracle signals for the sound decomposition evaluation experiments in Section 4.3.

4.3 Sound Decomposition Evaluation Experiments

This section presents a quantitative comparison between the NMFD and CAE models. This comparison will give the reader an insight on how close NMFD and CAE are in terms of sound decomposition quality, and of the sound decomposition capabilities of DNN-based sound decomposition models.

4.3.1 Specifications

The test specifications were set following the work by Dittmar et. al. in [9]. The STFT and LSEE-MSTFT parameters were set as in Section 2.3, i.e. using a Hann window function w of block size $N = 2048$, with hop size $H = 512$. The alpha Wiener filter parameter was set to $\alpha = 1$. The frame depth parameter was set to $T = 5$ for all experiments. Because the signal reconstruction operations require that the factor matrices are non-negative, only CAEs with non-negative decoder layer weights were used.

Four different initialization cases were considered, as shown in Table 4.3.1. Case 0 will always correspond to the oracle signals evaluation results.

Test Case	**Templates**	**Activations**
Case 1	Audio-based	Score-informed
Case 2	Audio-based	Random
Case 3	Random	Score-informed
Case 4	Random	Random
Case 0	(Oracle)	

Table 4.1. Score information cases evaluated.

For NMFD, random initialization factor matrix values were generated using a uniform distribution in the range $[0 : 1]$. Score information was introduced as described in Section 2.3.3. The score-informed tensor $\underline{\mathbf{W}}^{(0)}$ was generated using the drumset templates of the NMF Toolbox [24]. The score information was introduced by using a binary initialization matrix $\mathbf{H}^{(0)}$ generated from the ground truth annotations of each track. The algorithm was run for $L = 30$ iterations.

For the CAE models, random initialization weight matrix values were generated using a uniform distribution in the range $[0 : 0.1]$. Score information for decoder weight matrix was introduced through an informed initialization tensor $\underline{\mathbf{W}}_{\mathcal{D}}^{(0)}$. The decoder matrices were constrained using a binary mask $\mathbf{M_W}$, following the method described in Section 3.3.1. The score information for the code matrix $\mathbf{H}$ was introduced by generating a binary mask $\mathbf{M_H}$ from the annotations of each track, and using it in the structured dropout layer, as explained in Section 3.3.2. All CAEs were trained for 500 epochs, disabling bias parameters in their layers.

To make a fair comparison, the score-informed matrices used for matrix factorization and DNN-based approaches were generated in the same way. Matrix $\underline{\mathbf{W}}^{(0)}$ for NMFD is the same as $\underline{\mathbf{W}}_{\mathcal{D}}^{(0)}$ used for CAEs, and matrix $\mathbf{H}^{(0)}$ for NMFD is the same $\mathbf{M_H}$ used for CAEs. Matrices $\mathbf{H}^{(0)}$ and $\mathbf{M_H}$ were generated using a template duration of 500 ms.

4.3.2 Source Separation Evaluation Measure

For source separation evaluation, the performance of the sound decomposition algorithms was evaluated using the Signal-to-distortion ratio (SDR) [36]. Given a discrete, real-valued time domain signal x made up by R sound sources x_r for $r \in [1:R]$, a reconstructed sound source signal $\hat{x}_r$ can be defined as:

$$\hat{x}_r = x_r + x_r^{\text{interference}} + x_r^{\text{artifacts}},$$

meaning that each reconstruction signal $\hat{x}_r$ is assumed as a version of the original sound source signal x_r with added interference (or cross-talk) from the leakage from other components, and artifacts resulting from the imperfect signal reconstruction process. With this definition, the SDR for $\hat{x}_r$ is calculated as:

$$SDR = 10 \cdot \log_{10} \left(\frac{\|x_r\|^2}{\|x_r^{\text{interference}} + x_r^{\text{artifacts}}\|^2} \right). \tag{4.3}$$

For Case 0, the SDR values were computed by comparing the perfectly isolated signal x_r with their respective oracle signal $\tilde{x}_r$. For Cases 1 to 4, x_r the SDR values were computed by comparing x_r with their estimated signal $\hat{x}_r$.

4.3.3 NMFD - CAE audio decomposition

Figure 4.2 shows the sound decomposition evaluation results for NMFD and CAE in terms SDR. SDR was computed for the KD, SD and HH components individually, for each of the 64 tracks in the dataset, for each of the 4 initialization cases. To present these results, The graph displays the component SDR values averaged over all tracks in the dataset, plus a global average. Case 0 (Oracle) is plotted on both NMFD and CAE graphs for comparison.

All cases present an uneven structure, where the SDR of KD is considerably higher than for SD and HH. For Case 0, this is caused partly by the Wiener filter, which introduces artifacts from the other components into the the estimated magnitude spectrograms. In general, as discussed in section 2.3.1, KD has a frequency spectrum concentrated on the lower frequencies, meaning the frequency overlap with the other components is considerably less.

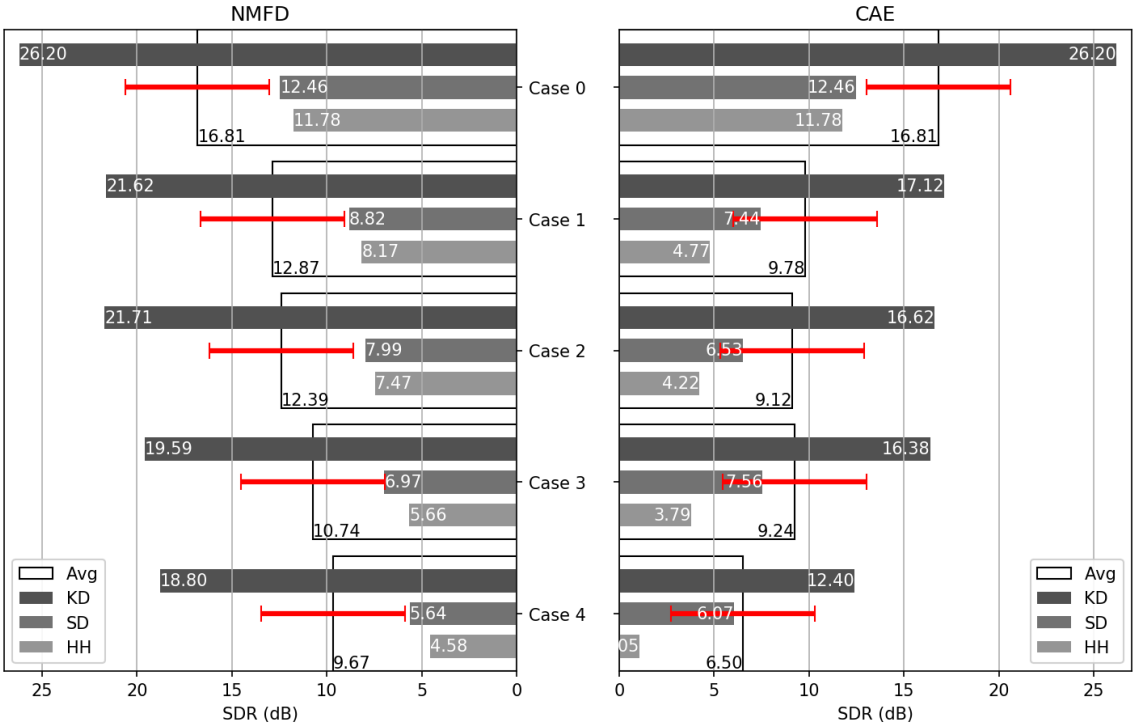

Figure 4.2. SDR evaluation results for NMFD (left) and CAE (right) for the cases described in Table 4.3.1, showing separation quality for KD, SD and HH, plus average (Avg) and standard deviation (in red).

Besides Case 0, the case with the highest SDR values is Case 1, which combines score-informed templates and activation matrices. For NMFD, the use of score-informed templates (Case 2) seems to have a higher impact on the score than the score-informed activations (Case 3). This behavior is opposite to the CAE results, where the SDR values of Case 3 are higher than for Case 2, and only slightly lower than those in Case 1. For CAE, the absence of score information (Case 4) leads to a considerably lower performance compared to NMFD.

It can be observed that the NMFD values are at least 5 dBs apart from the Case 0 values, for all components and cases tested. This means that a difference of a few dBs is considerable in terms of sound decomposition quality. Since the CAE scores are 1 to 4 dBs lower than the NMFD scores, it can be concluded that the performance of the CAE in these tests is lower than the performance of NMFD. As stated in [9], it is pertinent to listen to the reconstructed signals of both methods for multiple tracks to get an idea of what the SDR values mean in terms of perceptive sound quality.

During the tests, it was observed that the maximum template duration parameter, set to generate the score-informed activation matrices, also influenced the results of the evaluations. For times shorter than 500 ms, the CAE scores would increase around 2 to 3 dBs. This means the CAE sound decomposition quality might benefit from score-informed activations with shorter template

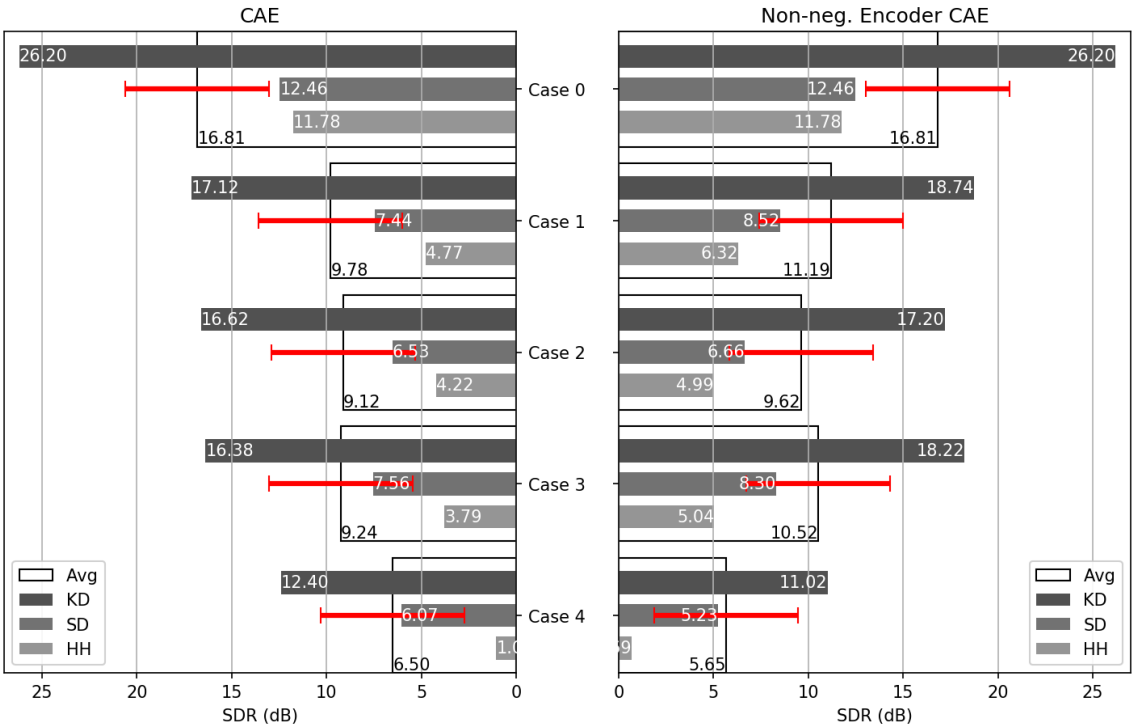

Figure 4.3. SDR evaluation results for a non-negative decoder CAE (left) and a non-negative encoder and decoder CAE (right) for the cases described in Table 4.3.1, showing separation quality for KD, SD and HH, plus average (Avg) and standard deviation (in red).

duration.

The NMFD evaluation results differ from the NMFD baseline experiment results from by Dittmar et.al in [9]. This is mainly because the score-informed activation matrix used in this case is binary, while the score-informed matrix in [9] aims to approximate the decaying envelope of each component.

4.3.4 Non-negative encoder CAE

A second experiment is based on the results observed in Section 3.2.3, where the activation matrix of the score-informed CAE looked cleaner and closer to the ground truth annotations when both encoder and decoder layers were non-negative (Figure 3.14), compared to the results obtained when non-negativity was enforced in the decoder layer only (Figure 3.13).

Figure 4.3 shows the obtained SDR scores for both CAE approaches. The SDR values are considerably higher for the non-negative encoder CAE. For Case 2, the difference is only of approx. 0.5 dBs, but it goes up to almost 2 dBs for Cases 1 and 3, which use score-informed activations. This means that the decomposition quality of the CAE is indeed increased when

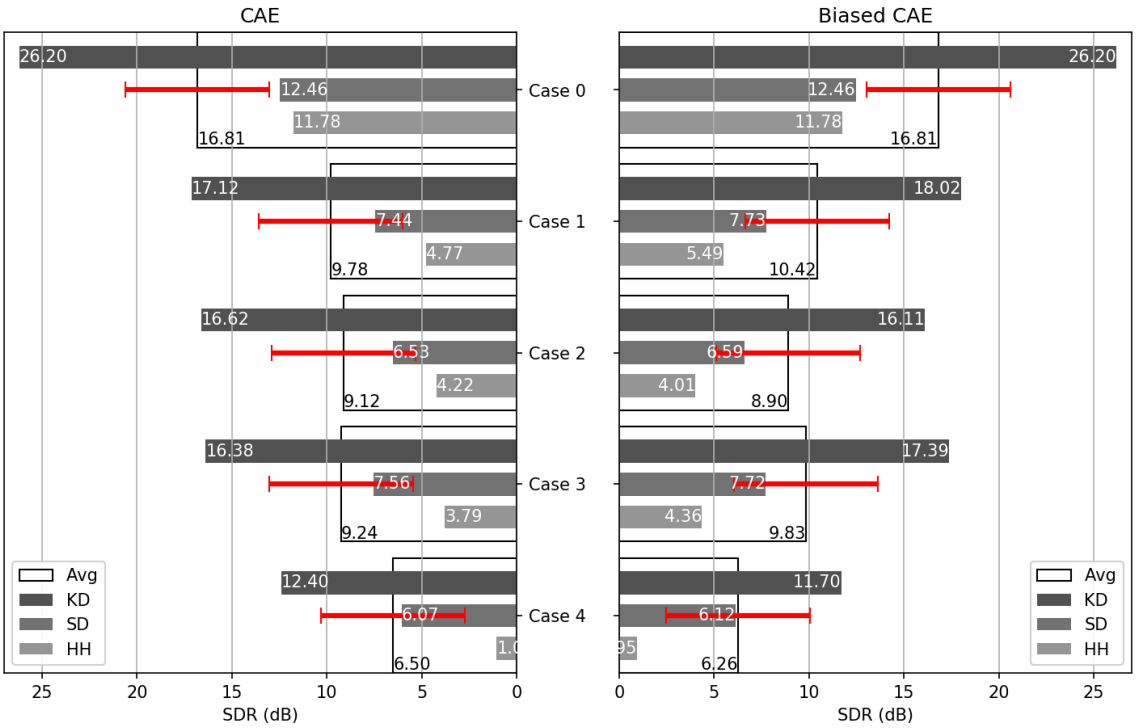

Figure 4.4. SDR evaluation results for a non-negative decoder CAE (left) and a non-negative decoder CAE with activated biases (right) for the cases described in Table 4.3.1, showing separation quality for KD, SD and HH, plus average (Avg) and standard deviation (in red).

both its layers are constrained to be non-negative. The SDR values approach the NMFD results for Case 3, but are still 1 or 2 dBs below the NMFD results for Cases 1 and 2.

4.3.5 Effect of Bias in CAE performance

So far, the bias parameters of the NAE and CAE models have been disabled, to focus solely on the training of encoder and decoder weight matrices. This experiment aims to observe the impact of including the bias parameters in the CAE architecture. All layer biases were initialized with zero values, and trained alongside the encoder and decoder weighs.

Figure 4.4 shows the obtained SDR values for the CAE and the biased CAE, trained with bias values enabled. The results show a 0.6 dB improvement for Cases 1 and 3, but a 0.2 dB decrease for Cases 2 and 4. This could imply the use of biases is enhancing the separation quality when the structured dropout layer is used, although its impact is not as considerable as the one of the non-negative encoder in Section 4.3.

Chapter 5

Conclusions

In this thesis, a seires of DNN-based matrix factorization models were implemented and used to perform sound decomposition in drum recordings. It was observed that the randomly-initialized shallow NAE is not suitable for sound decomposition, as the decoder layer weights are not constrained to be non-negative. The regularization approach suggested by Smaragdis in [31] is not a useful workaround to solve this problem. The implementation of non-negative constraints in the decoder layer weights allows the generation of NMF-like results.

Enforcing non-negativity in the encoder layer of the NAE or CAE models revealed that the encoder layer acts as a matched filter. The non-negative constrained encoder contains a set of frequencies that distinguishes a particular component from the rest, and outputs a high value whenever the input spectrogram frames match the frequency content of one of its columns. Using a non-negative encoder layer also leads to an improvement in the sound decomposition results.

There are various methods that can be used to introduce score information in NAE and CAE models. The use of score-informed NAEs is shown to improve the spectral and sound decomposition results for NAEs and CAEs as it does for NMF and NMFD. The proper configuration of the binary masks $\mathbf{M_W}$ and $\mathbf{M_H}$ plays an important role on the final sound decomposition results.

NMFD has overall better SDR scores than CAEs when applied in the IDMT-SMT-Drums database. The difference is between 0.5 dBs and 3 dBs.

There are several other tests left to perfrom on NAEs, such as evaluating their performance in other datasets, use recordings with a larger number of sound sources, and trying different combinations of constraints, biases, regularizers and DNN features to assess their effect on the sound decomposition quality. The use of 2D convolutional layers, resembling the NMF2D algorithm in [29] is a good example.

The methods shown in this document only comprise the use of single channel inputs. Nevertheless,

stereo mixes also contain valuable information that sound decomposition algorithms can benefit from, as the sound sources in a stereo sound mix are usually *panned* such that the listener has the sensation the sounds come from different directions. NMF and NMFD can be further extended for stereo or multi-channel audio inputs using techniques similar to non-negative tensor factorization, as proposed by Ozerov et.al. in [28]. Cashebeer et.al [2] propose an autoencoder architecture for multi-channel inputs, which is a natural extension to the NAE and CAE architectures, and can be easily implemented with the tools developed in this thesis.

Throughout this work, the magnitude spectrogram computed using STFT has been the starting point of the spectral decomposition analysis, being complemented by alpha Wiener filtering and LSEE-MSTFT techniques for signal reconstruction. These methods are commonly used and rather easy to implement, but also introduce artifacts and affect the sound decomposition quality. However, there are DNN models which are able to learn the transform and that best fits a given audio input, and generate an estimated time-domain signal accordingly. Venkataramani et.al. [35] propose the use of the so-called *end-to-end* systems, where the reconstruction error is computed in the time domain between the input time-domain signal and its DNN-generated estimate, and not between input and output magnitude spectrograms. Besides learning an optimal transform, the use of end-to-end systems also allows the DNN to use the phase information of the input signal for the sound decomposition process. This approach might greatly improve the sound decomposition results obtained in this work.

This thesis is only a few steps away from state of the art sound decomposition algorithms. One of the most well known state of the art end-to-end systems for waveform signal processing is the Wave-U-Net [32] which uses many of the techniques described in this thesis, but for a larger number of layers. The recently developed Demucs system [6], uses the Wave-U-Net architecture to perform source separation of polyphonic music recordings, splitting it into four categories: drums or percussion, bass, voice and others. The architecture of such systems is based on the same principles as NAEs.

Appendix A

Onset Models

The score-informed NMF results in Figure 2.5 show that the model for matrix $\mathbf{W}^{(0)}$ is oversimplified, as it only takes into account the harmonic part of the note spectrum, but it is unable to successfully reconstruct its onsets.

Since each of the piano notes in the example have both onset and harmonic structure, each of the notes can be assigned two components: one for their harmonic frequency content, and one uniform, noise-like template, that will encode the frequency information of the onsets. This implies the rank of the approximation must be doubled. For the piano example in Figure 2.1 the rank would be set to $R = 6$, to include the new onset information into the NMF algorithm. This implies double the amount of columns and rows for the template and activation matrices, respectively.

Figure A.1 (a) and (b) show the proposed onset model templates. The columns of $\mathbf{W}^{(0)}$ are a combination of horizontal, harmonic structures, and flat, noise-like spectra. Each note is assigned a noise template and a harmonic template. In the activation matrix $\mathbf{H}^{(0)}$, the onset templates are active only at the beginning of each note event, with their harmonic templates activated a few frames later. A reasonable overlap is left between both onset and harmonic activations to ensure a smooth transition between both types of components in the final approximation.

Figure A.1 (c) and A.1(d) show the leaned NMF matrices using the onset model templates. It is observed that the noise-like templates effectively capture the onset information and maintain the flat spectrum assigned in $\mathbf{W}^{(0)}$, although some also show high values in harmonic frequencies, as in Note 69. This derives into a lower amplitude in the values of the harmonic template. This has no effect in the approximation matrix $\widehat{\mathbf{V}}$ in Figure A.1(e), which is now considerably more similar to the input magnitude spectrogram than the one obtained Figure 2.5(e).

The onset models can be extended to an NMFD scenario, by incorporating note templates $\mathbf{U}_r^{(0)}$ that contain both onset and harmonic structures. Figure A.2(a) shows an example of a possible

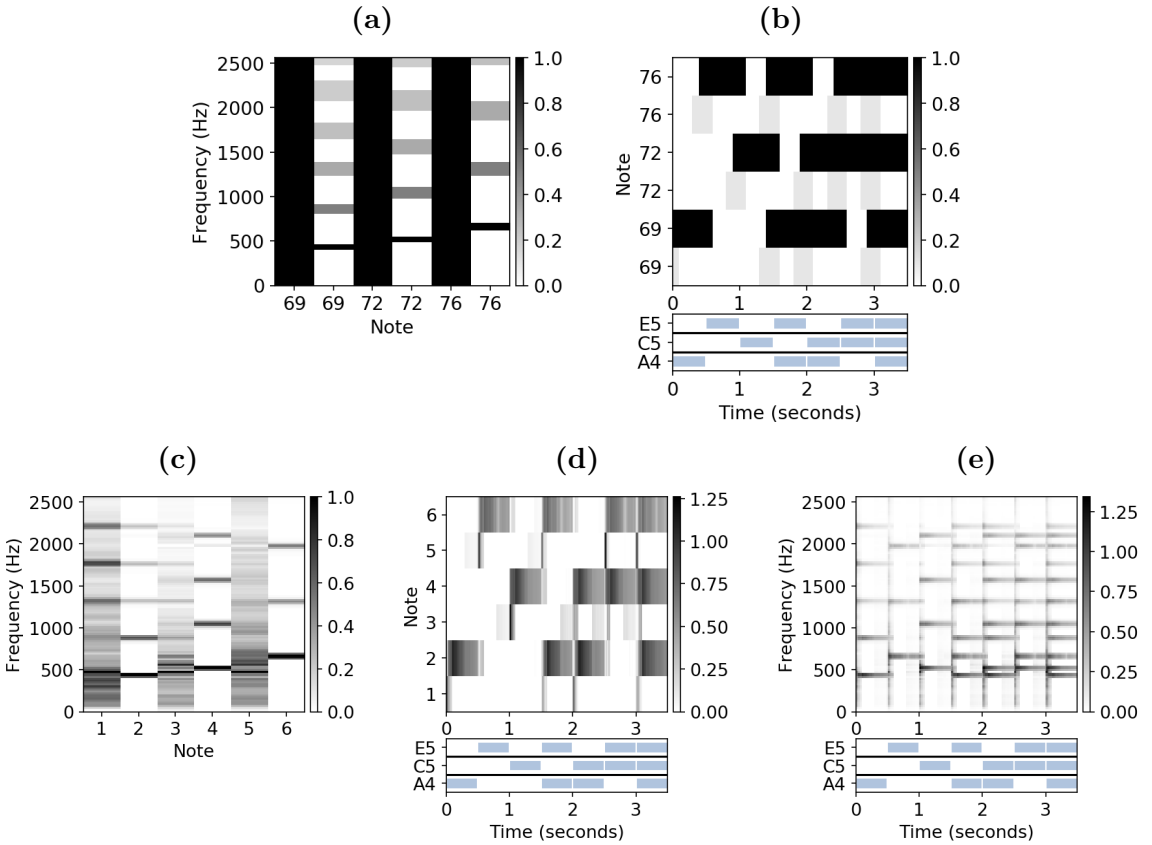

Figure A.1. Score-informed NMF for the piano running example on Figure 2.1. **(a)** Matrix $\mathbf{W}^{(0)}$ generated using frequency score information, with onset templates. **(b)** Matrix $\mathbf{H}^{(0)}$ generated using time score information, adding onset activations, with ground truth annotations (below). **(c)** Learned template matrix $\mathbf{W}$. **(d)** Learned activations matrix $\mathbf{H}$ with ground truth annotations (below). **(e)** Low rank approximation matrix $\widehat{\mathbf{V}} = \mathbf{WH}$ with ground truth annotations (below).

template initialization of $\underline{\mathbf{W}}^{(0)}$ combining both onset and harmonic templates for each of the notes. This initialization makes the learned $\underline{\mathbf{W}}$ in Figure A.2(c) encode the frequency content transition from onset to harmonic. The activation matrix $\mathbf{H}$ in Figure A.2(d) shows a high value at the beginning of each note event (the onset), followed by a more uniform area that corresponds to the slow decay of the fundamental frequency and the harmonics. The overlap between note events is also noticeable.

The score-informed matrix $\mathbf{W}^{(0)}$ in Figure A.1(a) can be seen as a particular case of the NMFD $\underline{\mathbf{W}}^{(0)}$ for $T = 2$, but in the NMFD case no additional rows in the activation matrix $\mathbf{H}$ are needed, and the parameter R still coincides with the number of components present in the piano piece.

Onset models can also be incorporated into NAE models, using the score-informed NAE strategies presented in 3.3, generating binary masks $\mathbf{M_W}$ and $\mathbf{M_H}$ from matrices $\mathbf{W}^{(0)}$ and $\mathbf{H}^{(0)}$. Figure A.3(a) and A.3(b) show how the NAE is able to learn a non-negative decoder weight matrix $\mathbf{W}_{\mathcal{D}}$ and a code matrix $\mathbf{H}$ that resemble the score-informed NMF template and activation

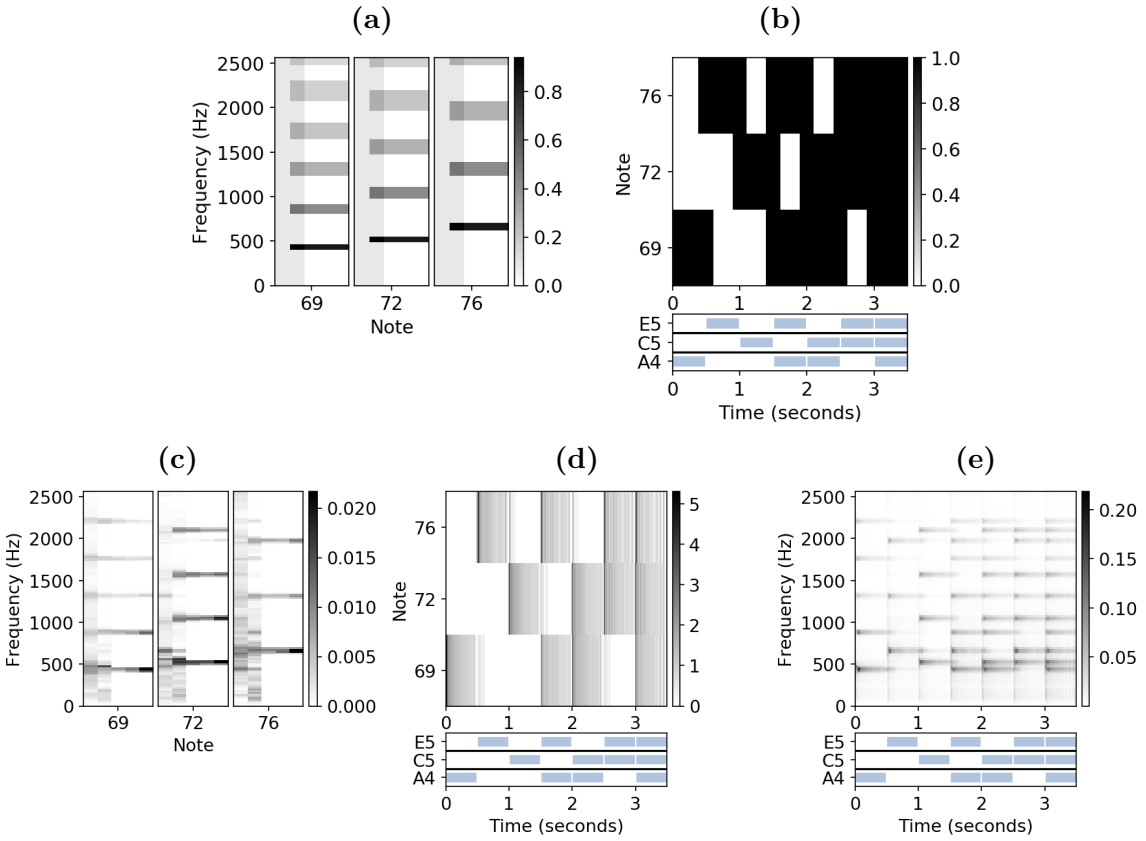

Figure A.2. Score-informed NMF for the piano running example on Figure 2.1. **(a)** Matrix $\underline{\mathbf{W}}^{(0)}$ generated using frequency score information, with onset templates. **(b)** Matrix $\mathbf{H}^{(0)}$ generated using score information, adding onset activations, with reference ground truth annotations (below). **(c)** Learned template matrices $\underline{\mathbf{W}}$. **(d)** Learned activations matrix $\mathbf{H}$. **(e)** Low rank approximation matrix $\widehat{\mathbf{V}}$ with reference ground truth annotations (below).

matrices, with the decoder matrix $\mathbf{W}_{\mathcal{D}}$ being almost entirely non-negative. It is also observed that the use of a non-negative decoder $\mathbf{W}_{\mathcal{D}^+}$ generates cleaner and more defined templates. The value range of the low rank approximation matrix $\widehat{\mathbf{V}}$ in Figure A.3(c) is closer to that of the original spectrogram, as the NMF low-rank approximation in Figure A.1(e).

The CAE matrices in Figure A.4 show how the CAE onset models can also be adapted to a DNN model. However, the learned code matrix $\mathbf{H}$ in Figure A.4(b) shows the onsets are not present in all notes, but are instead concentrated in note 76 (E5), which supposes a disadvantage in the spectrogram decomposition result with respect to NMFD.

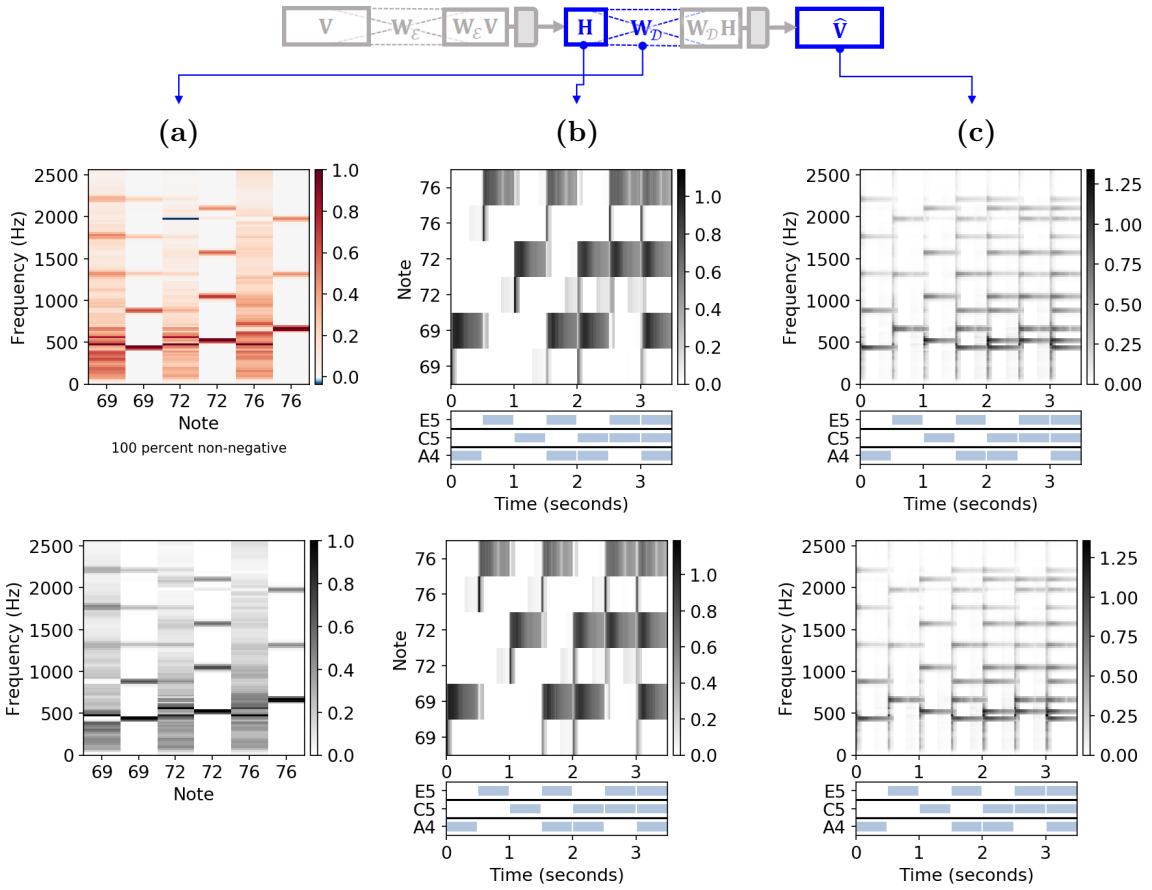

Figure A.3. Scored-informed NAE (above) and score informed NAE with non-negative decoder (below), for the piano running example.**(a)** Trained decoder matrices $\underline{\mathbf{W}}_{\mathcal{D}}$ (above) $\underline{\mathbf{W}}_{\mathcal{D}^+}$ (below). **(b)** Learned code matrix $\mathbf{H}$, with reference ground truth annotations (below). **(c)** Reconstructed input matrix $\widehat{\mathbf{V}}$, with reference ground truth annotations (below).

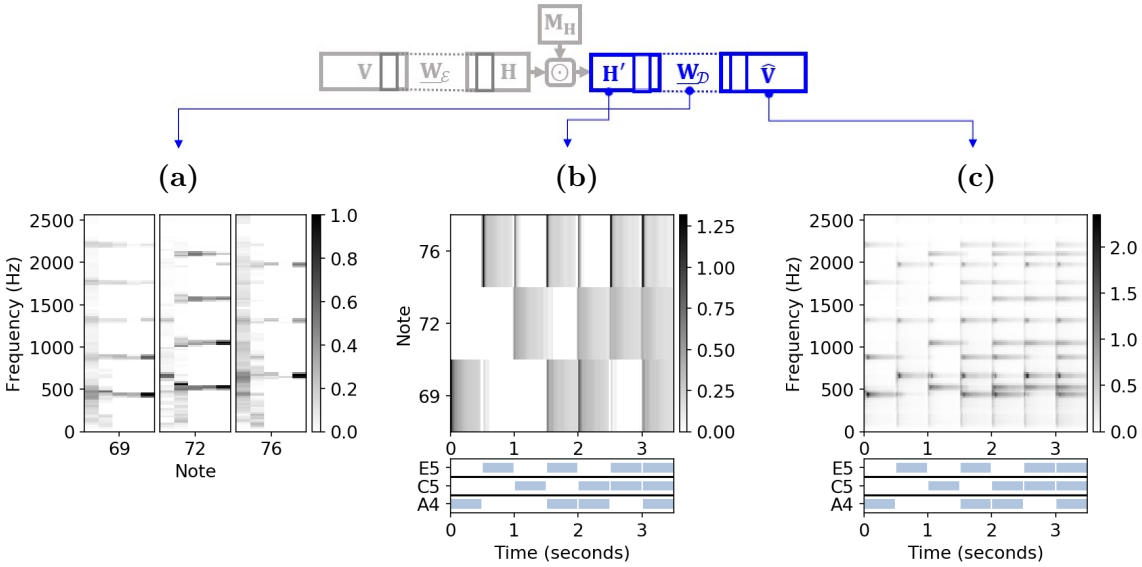

Figure A.4. Score-informed CAE with non-negative decoder, for the piano running example. **(a)** Learned non-negative decoder matrices $\underline{\mathbf{W}}_{\mathcal{D}^+}$. **(b)** Learned code matrix $\mathbf{H}$, with reference ground truth annotations (below). **(c)** Reconstructed input matrix $\widehat{\mathbf{V}}$, with reference ground truth annotations (below).

Appendix B

Convolution schemes

The NMFD convolution operation is described by the following equation (already presented in Section 2.4):

$$\widehat{\mathbf{V}}_{\text{NMFD}} := \sum_{\tau=1}^{T} \mathbf{W}_\tau \overset{\tau\rightarrow}{[\mathbf{H}]}, \tag{B.1}$$

To better understand this operation, we will use the toy example shown in Figure B.1 (below):

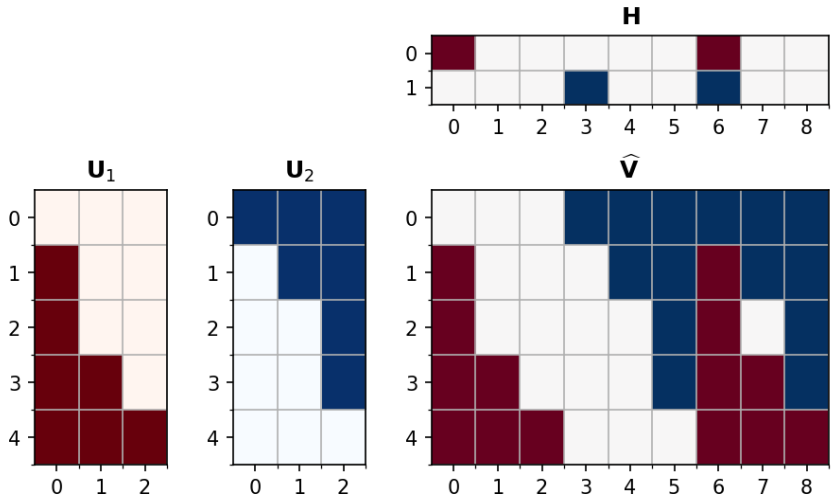

Figure B.1. Convolution operation example. The component matrices (left) are convoluted with the activation matrix (above) according to Equation (B.1) to generate the convolution matrix (center).

In this example there are two components, identified by the colors red and blue i.e. $R = 2$. Each component is made up by a matrix $\mathbf{U}_r \in \mathbb{R}^{5\times3}$, with $T = 3$ being the number of temporal frames chosen to describe each component. The templates $\mathbf{W}_\tau \in \mathbb{R}^{5\times2}$ are built by stacking one column from each component matrix together, for $\tau \in [1:3]$. The activation matrix $\mathbf{H}$ is a 2×9 matrix,

where the rows correspond to the components R, and the columns to the length of the sequence. The activation matrix sequence is also color-coded, and starts with the red component being active at frame 0, followed by the blue component at frame 3, and finally both components active at the same time at frame 6.

To understand how the resulting matrix $\widehat{\mathbf{V}}$ is calculated, Figure B.2 shows the process of using Equation (B.1) for each τ:

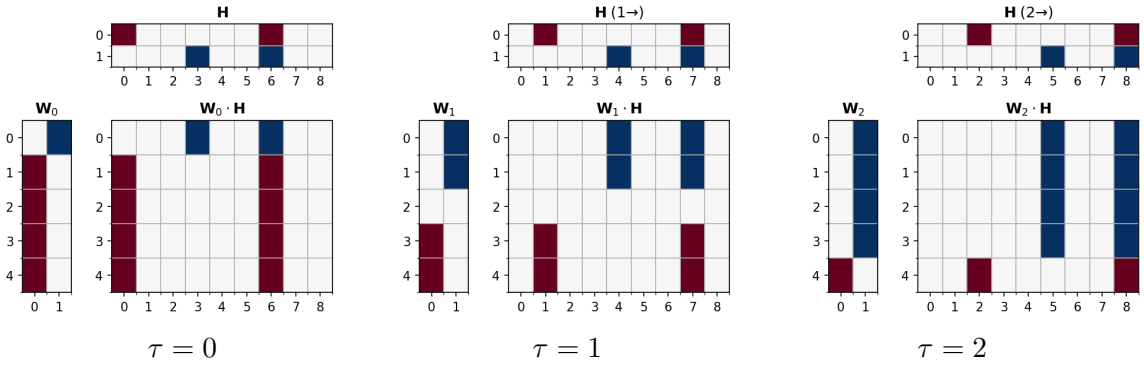

Figure B.2. Detailed convolution operation process for example in Figure B.1

.

Figure B.2 shows how matrix $\mathbf{H}$ is shifted one frame to the right for each τ, filling the remaining values with zeros. Matrix $\widehat{\mathbf{V}}$ is the sum of each of the resulting matrix products shown in Figure B.2. Notice that only one activation every $T = 3$ frames is needed to make the entire $\mathbf{U}_r$ component matrix appear in the final convolution matrix $\widehat{\mathbf{V}}$.

Unfortunately, when computing convolution using the Tensorflow `tf.nn.convolution` function (embedded in the 1D convolutional layer implemented in Section 3.4) the following result is obtained:

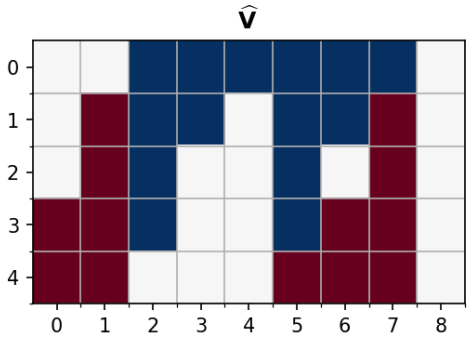

Figure B.3. Convolution operation result using the Tensorflow convolution function.

Figure B.3 shows a matrix $\widehat{\mathbf{V}}$ with the component matrices *flipped* along the vertical axis, and shifted to the left. As a consequence, when using convolutional layers in a CAE (see Chapter 3,

Section 3.4) to learn weight matrices $\mathbf{U}_{\mathcal{D}}$ given the input matrix $\mathbf{V}$ and the code matrix $\mathbf{H}$, the output filters would be flipped and shifted in the same way.

This phenomenon might not represent a problem in most DNN models, where location the filter values is not relevant as long as the network performs accurately. However, when using CAEs for sound decomposition, two main problems were found: First, when using audio-based layer weight constraints (see Section 3.3.1) the audio-based mask $\mathbf{M}_{\mathbf{W}}$ would not match the weight matrices $\mathbf{W}_\tau$, affecting the network training. Second, learning flipped or truncated weight matrices generates counterproductive results when constructing the component spectrograms for sound decomposition evaluation (Section 4.2), since the reconstruction is computed outside the neural network, using the convolution operation in Equation (B.1).

Two important aspects of the convolution operation are responsible of the result in Figure B.3: the shift operation and the padding configuration. These two configurations can be better observed in the step-by-step convolution process for the Tensorflow convolution operation:

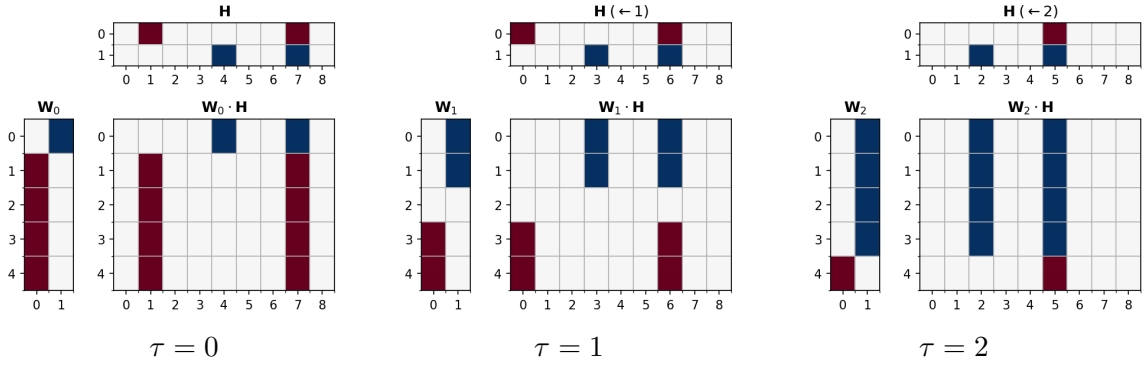

Figure B.4. Tensorflow convolution operation step by step. **(a)** $\tau = 0$. **(b)** $\tau = 1$. **(c)** $\tau = 2$.

Figure B.4 shows the shift operator is working in the opposite direction of the NMFD convolution operation, but the activation matrix $\mathbf{H}$ was not flipped, and neither was the order of the templates $\mathbf{W}_\tau$. This is the reason why matrices $\mathbf{U}_r$ appear flipped in matrix $\widehat{\mathbf{V}}$.

Chapter 9 of [14] points out that many machine learning libraries implement a type of *cross-correlation* operation, but call it convolution. This is the case for the Tensorflow convolution. Cross-correlation and convolution are similar operations, but they differ in the direction of the shift operator.

The other important parameter to configure in the Tensorflow operation is padding. For the NMFD convolution computation in Figure B.2, the shift operator is implicitly using a form of padding in matrix $\mathbf{H}$, adding adding zero-valued columns to the left side of $\mathbf{H}$ every time the matrix is shifted to the right. This allows the convolution operation result $\widehat{\mathbf{V}}$ to have the same number of columns as the activations matrix $\mathbf{H}$, adding a total of $T - 1$ columns to the original matrix.

However, the padding in Figure B.4 is different to that of the shift operator. For $\tau = 0$ a zero-valued column can be observed to the *left* of matrix $\mathbf{H}$. A brief look into the Tensorflow convolution source code shows it has two configurable padding options: `valid` and `same`. The `valid` padding means no zero-valued columns are added to either end of matrix $\mathbf{H}$. Using this scheme, the shift operator would discard the leftmost column of $\mathbf{H}$ without adding any columns to the right to compensate. This prevents the output $\widehat{\mathbf{V}}$ from having the same number of columns as $\mathbf{H}$. On the other hand, the `same` padding option guarantees that the input and output will have equal dimensions by adding $\lceil (T-1)/2 \rceil$ zero-valued columns *on both sides* of matrix $\mathbf{H}$.

Luckily, in the Keras 1D convolutional layer class, a third padding scheme option called `causal` is also available. This padding scheme, implemented through the `keras.backend.temporal_padding` function, adds $(T-1)$ columns *to the left* of matrix $\mathbf{H}$. The use of causal padding can be observed in the following Figure:

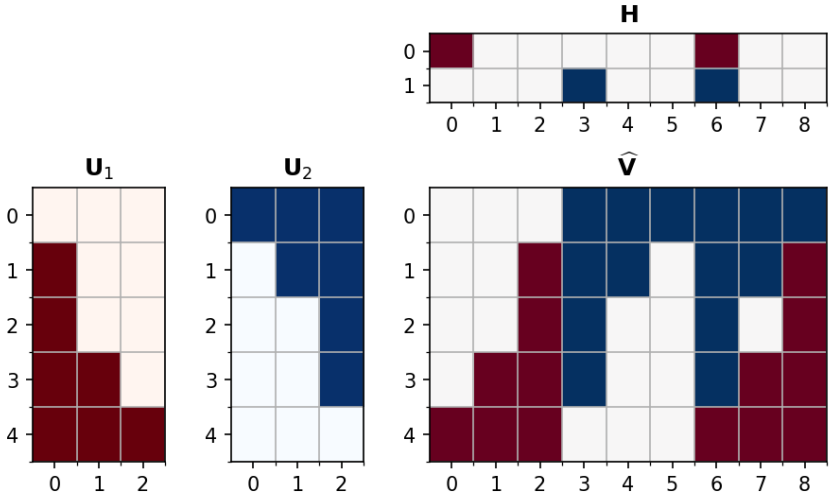

Figure B.5. Convolution operation result using the Tensorflow convolution function, using causal padding.

The 'causal' padding scheme outputs a matrix $\widehat{\mathbf{V}}$ with component matrices $\mathbf{U}_r$ which no longer truncated, but still flipped. A workaround solution implemented was to use the Keras 1D convolutional layer with causal padding, but taking into account the learned templates would be flipped. To apply the informed weight constraints described in Section 3.3.1 for the CAE, it was necessary to flip the informed initialization tensors $\underline{\mathbf{W}}_{\mathcal{D}}^{(0)}$ and the binary masks $\mathbf{M}_{\mathbf{W}}$ along the vertical axis. After training the CAE the resulting weight matrices $\mathbf{U}_r$ had to be flipped back before using Equation 2.20 to compute the low-rank approximation matrices $\widehat{\mathbf{V}}_r$ used for signal reconstruction.

Appendix C

NAE Performance comparison

This chapter provides a numerical comparison on the loss function values of several NMF and NAE implementations, to observe their convergence speed and to evaluate their performance in terms of reconstruction error. It is important to note that the performance of the algorithms is not a direct indicator of their spectral decomposition quality, but it gives some insight on how the different models are working. All performance tests were made using the drums running example from Section 2.3.1 as an input. Noth NMF and NAE algorithms were run for $L = 1000$ iterations / epochs.

C.1 NAE Performance Comparison

As a measure of performance, the loss function $\mathcal{L}$ was computed during the training of different NAE models, and compared with the divergence in NMF. Table C.1 shows the initial and final loss values for NMF and the three main NAE models described in Chapter 3: The randomly initialized, unconstrained NAE; the NAE with non-negative encoder; and the NAE with both non-negative encoder and decoder. The table shows that the NMF initial loss value is higher, mainly because its values were randomly initialized in the $[0 : 1]$ interval, while the NAEs were initialized in the $[0 : 0.1]$ interval. However, the NMF loss value ends up being lower than the final NAE loss. All models reach a similar final loss value.

Figure C.1 shows the behavior of the loss function for each model on the first 200 iterations (in the case of NMF), and for the first 200 epochs for the NAE models. Although the NMF loss starts at a higher value, it drops drastically after very few iterations. For the NAEs, the loss decays in a much smoother way, and require more iterations to approach the same loss values after 1000 epochs. These results most likely derive from the fact that NAEs must train a higher number of parameters. In particular, the loss of the NAE with non-negative encoder and decoder

Model	Initial $\mathcal{L}$ value	Final $\mathcal{L}$ value
NMF	194271.860	1040.121
NAE	76058.477	1123.889
NAE$[\mathcal{D}^+]$	76058.477	1073.836
NAE$[\mathcal{E}^+, \mathcal{D}^+]$	76058.477	1046.907

Table C.1. Initial and final loss function values for NMF and the NAE models described. **NAE**$[\mathcal{D}^+]$: NAE with non-negative decoder. NAE, **NAE**$[\mathcal{E}^+, \mathcal{D}^+]$: NAE with non-negative encoder and decoder.

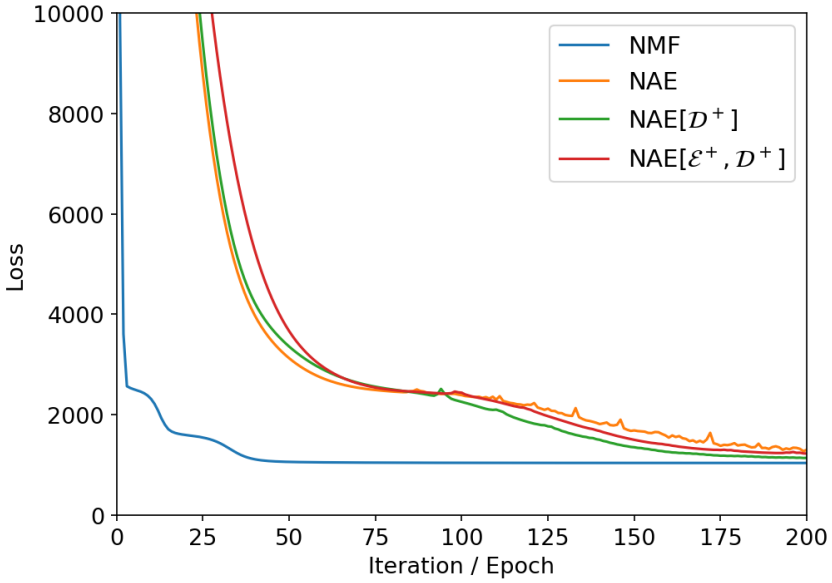

Figure C.1. Loss function values for the first 200 iterations (NMF) / Epochs (NAEs). **NAE**$[\mathcal{D}^+]$: NAE with non-negative decoder. NAE, **NAE**$[\mathcal{E}^+, \mathcal{D}^+]$: NAE with non-negative encoder and decoder.

is the one that decays more slowly, but its final value is the closest to NMF.

C.2 Score-Information Strategies Performance Comparison

Due to the large amount of combinations between NAE constraints and score information strategies, only NAE models with non-negative encoder were used for computing the performance results in this section.

The initial values of both NMF and NAE models are considerably higher when score-informed methods are used, mainly because the values of the initialization matrices are not uniformly distributed and not scaled to fit the random initialization intervals. For this example, the models with audio-based templates or weight matrices converge faster and have the lowers reconstruction error values, showing the effectiveness of the NMF Toolbox templates for this example. In contrast, the algorithms which use informed activations strategies or dropout do not reach the low values of the randomly initialized algorithms. The randomly-initialized algorithms are able

Model	**Initial $\mathcal{L}$ value**	Final $\mathcal{L}$ value
Random Initialization	190741.242	1040.291
Informed Templates	4423192.127	1040.114
Informed Activations	221834.147	1510.541
Informed Templates and Activations	4004946.754	1510.142

Table C.2. Initial and final loss function values of various NMF initialization schemes, for the drums running example.

Model	**Initial $\mathcal{L}$ value**	**Final $\mathcal{L}$ value**
Random Initialization	76058.477	1073.836
Informed decoder weights $\mathbf{W}'_{\mathcal{D}^+}$	22009.172	1074.242
Dropout	43113.035	1820.819
Informed $\mathbf{W}'_{\mathcal{D}^+}$ + Dropout	30133.668	1821.814
Dropout + Regularization ($\lambda = 10$)	97777.140	1852.251

Table C.3. Initial and final loss function values of various NAE initialization schemes, for the drums running example.

to reconstruct the input signal with low error, but as shown in Sections 2.3.2 and 3.2.1, they do no generate meaningful factor matrices.

As in Figure C.1, Figure C.2 shows that the score-informed NMF algorithms start at higher values, but converge in significantly fewer iterations. The converge of the NAE models is again slower, but reaches similar reconstruction error values after $L = 1000$ epochs.

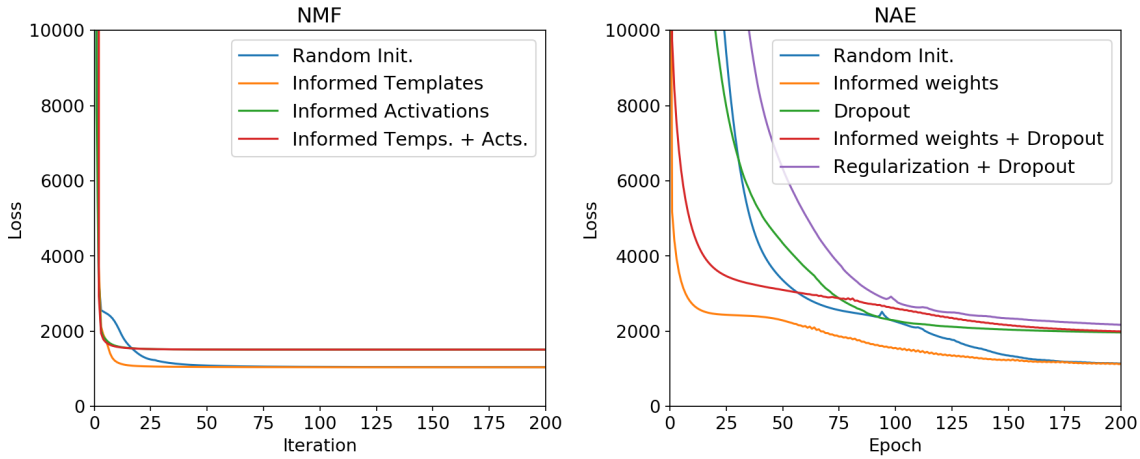

Figure C.2. Loss function values for the first 200 iterations (NMF, left) / epochs (NAE, right).

The NAE which combines the dropout and regularization strategies is the one with slowest convergence and the highest reconstruction error value. This result contradicts the work in [13], which states that the combination of both strategies drastically improves performance and makes the algorithm converge is less epochs, although it is not claimed that this is the case in general. For this particular example, setting the dropout parameter λ to zero, the reconstruction error

curve would be the same as the dropout NAE curve. A parameter λ bigger than zero increases the total loss function and negatively affects the performance of the NAE.

Appendix D

Source Code

In this chapter, the headers of selected python functions created during the writing of this thesis are reproduced. The headers contain information about the name of the described function and its input/output behavior.

```
def autoencoder(V, R, mode='basic', loss_function=custom_loss, activation='relu', l1_reg=0.0,
                epochs=1000):
    """Shallow autoencoder model, following the proposed architecture in [1].

        References
        ----------
        [1] P. Smaragdis and S. Venkataramani, A neural network alternative to non-negative
         audio models,in Proceedings of the IEEE International Conference on Acoustics,
         Speech, and Signal Processing(ICASSP), New Orleans, LA, USA, 2017, pp. 86{90.

        Parameters
        ----------
        V: array-like
            Input matrix (typically a magnitude spectrogram of dimension K x M)

        R : int
            code dimension, indicates the rank of the code matrix.

        mode: str
            Indicates which autoencoder mode to run.
            "normalized" sets norm of the columns of the decoder weights to 1.
            "non-negative" constraints the encoder and decoder matrices to be non-negative.
            "nn_dec" constraints only the decoder matrix to be non-negative.
            "nn_norm" constraints the encoder and decoder matrices to be non-negative,
            and the norm of the decoder matrix columns to 1.

        loss_function:  function or class
            The loss function used to train the autoencoder.

        activation: str
            A Keras activation function used in the encoder and decoder layers.
```

```
        l1_reg: float >= 0
            L1 regularizer parameter for the encoder output, following the implementation in [1].

        epochs: int
            Number of epochs for training.

        Returns
        -------
        V_hat: array-like
            Low-rank approximation matrix

        W_D: array-like
            Learned decoder weight matrix

        W_E: array-like
            Learned encoder weight matrix

        H: array-like
            Learned code matrix

        history: array-like
            A keras-generated vector, with the computation of the loss and mean squared error (MSE)
            of the autoencoder at each epoch.


def dropout_autoencoder(V, R, T_W=None, T_H=None, nn=False, temp_const=False,
                        act_const=False, l2_reg=0.,
                        loss_function=custom_loss, activation='relu', epochs=1000):
    """Shallow autoencoder with structured dropout layer, following the proposed architecture in [2].

    References
        ----------
        [2] S. Ewert and M. B. Sandler,Structured  dropout  for  weak  label  and
        multi-instance learning and its application to score-informed source separation,
        in Proceedings of the IEEE International Conference on Acoustics, Speech,
        and Signal Processing (ICASSP),
        New Orleans, LA, USA, 2017,pp. 2277{2281.

        Parameters
        ----------
        V: array-like
            Input matrix (typically a magnitude spectrogram of dimension K x M)

        R : int
            code dimension, indicates the rank of the code matrix.

        T_W: array-like
            Score informed template matrix from NMF

        T_H: array-like
            score informed activation matrix from NMF

        nn: bool
            When True, constrains the decoder weight matrix to be non-negative
```

```
        temp_const: bool
            When True, enables the use of informed constraints in the decoder matrix

        act_const: bool
            When True, enables the structured dropout layer

        l2_reg: float >= 0
            L2 regularizer parameter for the encoder output, following the
            implementation in [2].

        loss_function:  function or class
            The loss function used to train the autoencoder.

        activation: str
            A Keras activation function used in the encoder and decoder layers.

        epochs: int
            Number of epochs for training.

        Returns
        -------
        V_hat: array-like
            Low-rank approximation matrix

        W_D: array-like
            Learned decoder weight matrix

        W_E: array-like
            Learned encoder weight matrix

        H: array-like
            Learned code matrix

        history: array-like
             A keras-generated vector, with the computation of the loss and mean squared error (MSE)
             of the autoencoder at each epoch.
        """

def cae(V, R, T, T_W=None, T_H=None, cae_params=dict()):
    """Shallow convolutional autoencoder with structured dropout layer, following the
    proposed architecture in [3].

            References
                ----------
                [3] S. Venkataramani, C. Subakan, and P. Smaragdis,Neural network
                alternatives to convolutive audio models for source separation, in
                Proceedings of the IEEE International Workshop on Machine
                Learning for Signal Processing (MLSP), Tokyo, Japan, 2017, pp. 1{6.

                Parameters
                ----------
                V: array-like
                    Input matrix (typically a magnitude spectrogram of dimension K x M)

                R : int
```

```
                    code dimension, indicates the rank of the code matrix.

                T: int
                    Template temporal dimension / length of the layer filters

                T_W: array-like
                    Score informed template tensor from NMFD

                T_H: array-like
                    score informed activation matrix from NMF

                cae_params: dict
                    loss_function           The loss function used to train the cae
                    epochs                  Number of epochs to train the cae
                    reg                     l2 regularizer parameter, folloing strategy on [2]
                    act_const               Enables the structured dropout layer
                    enc_activation          Encoder activation function
                    enc_use_bias            Enables / disables bias in encoder layer
                    enc_non_neg             Enforces non-negativity in the encoder layer weights
                    enc_temp_init           Enables informed initialization of encoder weights
                    enc_temp_const          Enables informed constraints of the encoder weights
                    dec_activation          Decoder activation function
                    dec_use_bias            Enables / disables bias in decoder layer
                    dec_non_neg             Enforces non-negativity in the decoder layer weights
                    dec_temp_init           Enables informed initialization of decoder weights
                    dec_temp_const          Enables informed constraints of the decoder weights

                Returns
                -------
                V_hat: array-like
                    Low-rank approximation matrix

                W_D: array-like
                    Learned decoder weight tensor

                W_E: array-like
                    Learned encoder weight tensor

                H: array-like
                    Learned code matrix

                history: array-like
                     A keras-generated vector, with the computation of the loss and
                     mean squared error (MSE) of the autoencoder at each epoch.
                """

def plot_autoencoder(V, W, H, fs, N_fft, H_fft, labels=['', '', ''], ann=[], label_keys=[],
                        input_type='drums', pitch_set=[], order=[0,1,2], maxnorm=True,
                        subplot_size=(4, 4)):

    """Autoencoder plotting function, inspired by the FMP notebooks

            Parameters
            ----------
            V: array-like
```

```
            Low rank approximation matrix to plot

        W: array-like
            Template matrix / tensor to plot

        H: array-like
            Activation matrix to plot

        fs: int
            Sampling frequency

        N_fft:
            STFT block size

        H_fft: int
            STFT hop size

        labels: list
            List of strings, with labels for the matrices

        ann: array-like
            Annotation file from the FMP notebooks.

        label_keys: array-like
            Annotation parameter for drum annotation plots

        input_type: str
            "piano" or "drums", selects the type of plot desired

        pitch_set: list
            List of note numbers to plot

        order: array-like
            Array of size R which sorts the default order
            of columns / rows of the template / activation matrices

        maxnorm: Bool or Array
            Enables max-normalization using the template matrix, or an input matrix

        subplot_size: list
            Sets individual subplot figure sizes
    """
```

Bibliography

[1] M. Abadi et al., *TensorFlow: Large-scale machine learning on heterogeneous systems*, 2015. Software available from `https://www.tensorflow.org/`.

[2] J. Casebeer, M. Colomb, and P. Smaragdis, *Deep tensor factorization for spatially-aware scene decomposition*, in 2019 IEEE Workshop on Applications of Signal Processing to Audio and Acoustics, WASPAA 2019, New Paltz, NY, USA, October 20-23, 2019, IEEE, 2019, pp. 180–184.

[3] K. Choi, G. Fazekas, K. Cho, and M. B. Sandler, *A tutorial on deep learning for music information retrieval*, CoRR, abs/1709.04396 (2017).

[4] F. Chollet et al., *Keras*. `https://keras.io`, 2015.

[5] J. Chorowski and J. M. Zurada, *Learning understandable neural networks with nonnegative weight constraints*, IEEE Transactions on Neural Networks and Learning Systems, 26 (2015), pp. 62–69.

[6] A. Défossez, N. Usunier, L. Bottou, and F. Bach, *Music Source Separation in the Waveform Domain*, Tech. Rep. 02379796v1, HAL, 2019.

[7] C. H. Q. Ding, T. Li, and M. I. Jordan, *Convex and semi-nonnegative matrix factorizations*, IEEE Transactions on Pattern Analysis and Machine Intelligence, 32 (2010), pp. 45–55.

[8] C. Dittmar and D. Gärtner, *Real-time transcription and separation of drum recordings based on NMF decomposition*, in Proceedings of the International Conference on Digital Audio Effects (DAFx), Erlangen, Germany, September 2014, pp. 187–194.

[9] C. Dittmar and M. Müller, *Reverse engineering the Amen break – score-informed separation and restoration applied to drum recordings*, IEEE/ACM Transactions on Audio, Speech, and Language Processing, 24 (2016), pp. 1531–1543.

[10] J. Driedger and M. Müller, *Extracting singing voice from music recordings by cascading audio decomposition techniques*, in Proceedings of the IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP), Brisbane, Australia, 2015, pp. 126–130.

[11] J. Driedger, T. Prätzlich, and M. Müller, *Let It Bee – Towards NMF-inspired audio mosaicing*, in Proceedings of the International Society for Music Information Retrieval Conference (ISMIR), Málaga, Spain, 2015, pp. 350–356.

[12] S. Ewert and M. Müller, *Using score-informed constraints for NMF-based source separation*, in Proceedings of the IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP), Kyoto, Japan, March 2012, pp. 129–132.

[13] S. Ewert and M. B. Sandler, *Structured dropout for weak label and multi-instance learning and its application to score-informed source separation*, in Proceedings of the IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP), New Orleans, LA, USA, 2017, pp. 2277–2281.

[14] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*, MIT Press, Cambridge and London, 2016. `http://www.deeplearningbook.org`.

[15] D. W. Griffin and J. S. Lim, *Signal estimation from modified short-time Fourier transform*, IEEE Transactions on Acoustics, Speech, and Signal Processing, 32 (1984), pp. 236–243.

[16] G. Hinton and R. Salakhutdinov, *Reducing the dimensionality of data with neural networks*, Science, 313 (2006), pp. 504–507.

[17] D. D. Lee and H. S. Seung, *Learning the parts of objects by non-negative matrix factorization*, Nature, 401 (1999), pp. 788–791.

[18] ——, *Algorithms for non-negative matrix factorization*, in Proceedings of the Neural Information Processing Systems (NIPS), Denver, Colorado, USA, November 2000, pp. 556–562.

[19] A. Lefevre, F. Bach, and C. Févotte, *Semi-supervised NMF with time-frequency annotations for single-channel source separation*, in Proceedings of the International Society for Music Information Retrieval Conference (ISMIR), Porto, Portugal, 2012, pp. 115–120.

[20] C. Lin, *On the convergence of multiplicative update algorithms for nonnegative matrix factorization*, IEEE Transactions on Neural Networks, 18 (2007), pp. 1589–1596.

[21] ——, *Projected gradient methods for nonnegative matrix factorization*, Neural Computation, 19 (2007), pp. 2756–2779.

[22] A. Liutkus and R. Badeau, *Generalized Wiener filtering with fractional power spectrograms*, in Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), Brisbane, Australia, April 2015, pp. 266–270.

[23] P. López-Serrano, C. Dittmar, and M. Müller, *Finding drum breaks in digital music recordings*, in Proceedings of the International Symposium on Computer Music Multidisciplinary Research (CMMR), Porto, Portugal, September 2017, pp. 68–79.

[24] P. López-Serrano, C. Dittmar, Y. Özer, and M. Müller, *Nmf toolbox: Music processing applications of nonnegative matrix factorization*, in Proceedings of the International Conference on Digital Audio Effects (DAFx), Birmingham, UK, September 2019.

[25] M. Müller, *Fundamentals of Music Processing*, Springer Verlag, 2015.

[26] M. Müller and F. Zalkow, *FMP notebooks: Educational material for teaching and learning fundamentals of music processing*, in Proceedings of the International Conference on Music Information Retrieval (ISMIR), Delft, The Netherlands, 2019, pp. 573–580.

[27] M. A. Nielsen, *Neural Networks and Deep Learning*, Determination Press, 2015. `http://neuralnetworksanddeeplearning.com`.

[28] A. Ozerov and C. Févotte, *Multichannel nonnegative matrix factorization in convolutive mixtures. with application to blind audio source separation*, in Proceedings of the IEEE International Conference on Acoustics, Speech, and Signal Processing, ICASSP 2009, 19-24 April 2009, Taipei, Taiwan, IEEE, 2009, pp. 3137–3140.

[29] M. N. Schmidt and M. Mørup, *Nonnegative matrix factor 2-d deconvolution for blind single channel source separation*, in Independent Component Analysis and Blind Signal Separation, 6th International Conference, ICA 2006, Charleston, SC, USA, March 5-8, 2006, Proceedings, J. P. Rosca, D. Erdogmus, J. C. Príncipe, and S. Haykin, eds., vol. 3889 of Lecture Notes in Computer Science, Springer, 2006, pp. 700–707.

[30] P. Smaragdis, *Non-negative matrix factor deconvolution; extraction of multiple sound sources from monophonic inputs*, in Proceedings of the International Conference on Independent Component Analysis and Blind Signal Separation ICA, Grenada, Spain, September 2004, pp. 494–499.

[31] P. Smaragdis and S. Venkataramani, *A neural network alternative to non-negative audio models*, in Proceedings of the IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP), New Orleans, LA, USA, 2017, pp. 86–90.

[32] D. Stoller, S. Ewert, and S. Dixon, *Wave-U-net: A multi-scale neural network for end-to-end audio source separation*, in Proceedings of the 19th International Society for Music Information Retrieval Conference (ISMIR), Paris, France, 2018, pp. 334–340.

[33] N. Vasiloglou, A. G. Gray, and D. V. Anderson, *Non-negative matrix factorization, convexity and isometry*, CoRR, abs/0810.2311 (2008).

[34] S. Venkataramani, C. Subakan, and P. Smaragdis, *Neural network alternatives to convolutive audio models for source separation*, in Proceedings of the IEEE International Workshop on Machine Learning for Signal Processing (MLSP), Tokyo, Japan, 2017, pp. 1–6.

[35] S. Venkataramani, E. Tzinis, and P. Smaragdis, *End-to-end non-negative autoencoders for sound source separation*, CoRR, abs/1911.00102 (2019).

[36] E. Vincent, N. Bertin, R. Gribonval, and F. Bimbot, *From blind to guided audio source separation: How models and side information can improve the separation of sound*, IEEE Signal Processing Magazine, 31 (2014), pp. 107–115.

[37] E. Vincent, R. Gribonval, and C. Févotte, *Performance measurement in blind audio source separation*, IEEE Transactions on Audio, Speech, and Language Processing, 14 (2006), pp. 1462–1469.

[38] E. Vincent and X. Rodet, *Music transcription with ISA and HMM*, in Proceedings of the International Conference on Independent Component Analysis and Blind Signal Separation (ICA), 2004, pp. 1197–1204.

[39] Y.-X. Wang and Y.-J. Zhang, *Nonnegative matrix factorization: A comprehensive review*, Knowledge and Data Engineering, IEEE Transactions on, 25 (2013), pp. 1336–1353.

[40] C.-W. Wu, C. Dittmar, C. Southall, R. Vogl, G. Widmer, J. Hockman, M. Müller, and A. Lerch, *A review of automatic drum transcription*, IEEE/ACM Transactions on Audio, Speech, and Language Processing, 26 (2018), pp. 1457–1483.

[41] K. Yoshii, R. Tomioka, D. Mochihashi, and M. Goto, *Beyond nmf: Time-domain audio source separation without phase reconstruction*, in in Proceedings of the International Society for Music Information Retrieval Conference (ISMIR), 2013. DAFX-7, 2013.