Friedrich-Alexander-Universität Erlangen-Nürnberg

Master Thesis

# Neural Networks with Nonnegativity Constraints for Decomposing Music Recordings

submitted by

Tim Zunner

submitted

January 14, 2021

Supervisor / Advisor

M. Sc. Michael Krause
Prof. Dr. Meinard Müller

Reviewers

Prof. Dr. Meinard Müller

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

*To Marina*

*"How do you write like you're running out of time?"*
*Aaron Burr, Hamilton*

# 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, January 14, 2021

Tim Zunner

# Acknowledgements

I would like to express my gratitude to the people who supported me throughout this thesis.

To Prof. Dr. Meinard Müller, who has not only been my mentor since the beginning of my master studies but also agreed to supervise this thesis. Thank you for helping me find the topics I am most interested in and enabling me to work on exactly these in a warm and welcoming research environment. I appreciate your support and your efforts to always motivate students to give the best they can.

Thank you to my supervisor Michael Krause, who always helped me with whatever problem I encountered and gave me inspiring ideas that made the results of this thesis possible. It was a pleasure to work with you and I wish you the best of luck for your remaining PhD studies.

To Yiğitcan Özer, who did not only supervise me during my job at Fraunhofer IIS but also showed great interest in my research topic. Thank you for your support and input during our fruitful discussions. I wish you all the best for your upcoming PhD studies.

I would also like to thank Edgar Suárez whose inspiring work was the foundation of this thesis. Without your detailed documentation and your advice I would not have been able to get into the depth of this thesis so quickly.

A special thank you to Christian Dittmar who gave me the possibility to work at Fraunhofer IIS. I enjoyed working in your research group and it gave me some pleasant variety during the work on this thesis. I look forward to also work with you in the future.

To Frank Zalkow, who helped me with technical problems throughout my experiments and with the presentation of my thesis in the ASC forum.

Furthermore, I would like to thank all my friends from the ASC study program. It was a life changing experience to study with a group of so many nice and open-minded people. I especially want to thank Fabian and Ángel who helped me with proofreading this thesis.

I also want to thank my family for always supporting me and giving me the possibility to pursue my studies. I especially want to thank my grandpa who passed away shortly before I started working on this thesis. Thank you for your support throughout all phases of my life, you will forever be in my heart.

Finally, I want to express my gratitude to my girlfriend, soulmate and best friend Marina. Thank you for supporting me in whatever I want to achieve and helping me become the best version of myself. You always know how to keep me motivated.

# Abstract

Source separation is a common task in audio signal processing. In this thesis, we deal with a special case of source separation in the field of music information retrieval, namely music decomposition. Music recordings often consist of a superposition of different sound sources such as different instruments or components of one instrument. The task in music decomposition is to split a music recording into separated signals for the desired sources. In this thesis, we decompose solo piano recordings into two source signals: the left hand signal typically containing the accompaniment and the right hand signal typically containing the melody. That is, we want to obtain separate audio tracks for the left and right hand from the same piano recording.

A long-known machine learning algorithm used for audio decomposition is nonnegative matrix factorization (NMF). The basic idea behind this algorithm is to approximate a nonnegative matrix by a product of two nonnegative matrices. In audio processing it is used to factorize a spectrogram into a template matrix containing the frequency templates for the approximation and an activation matrix containing the corresponding activation patterns over time. Using multiplicative update rules for this algorithm speeds up the convergence and enables us to guide the solution of the algorithm to be interpretable from a musical point of view.

In recent years, neural networks have been used to achieve state-of-the-art results in many tasks in the areas of image and audio signal processing. This also holds for music information retrieval. One possibility to perform spectrogram factorization using neural networks are nonnegative autoencoders (NAEs), which are a neural network-based alternative to nonnegative audio models. In previous work it has been shown that this algorithm can also be guided similar to NMF in order to achieve comparable decomposition results which are musically meaningful.

In this thesis, we compare different NMF and NAE approaches by applying them to the piano scenario and studying their convergence behavior. Additionally, we study the problem of a rising loss value in the NMF approach with multiplicative updates. We give our own contribution to the NAE approach by deriving multiplicative update rules for the network weights of the NAE and show that these new update rules speed up the convergence of the network. Furthermore, they simplify imposing constraints on the network weights. Finally, we show ideas for the initialization of the encoder weights of the NAE and apply the NMF and NAE approaches to a bigger dataset.

# Contents

# Chapter 1

# Introduction

Recordings of music performances often contain a mixture of different sources. For example, if we consider the performance of a band, we have the singer as one source and the different instruments like guitar, drums or bass as multiple additional sources. Other examples could be the different voices in a choir recording or even the different components of one single instrument. While the mixing of different sources plays an important role in music performances, the task of audio decomposition is to revert this mixing process and create separated signals for the different sources. This is needed in order to analyze or process the sources independent of one another. As mentioned in [30], audio decomposition is the baseline for various music information retrieval tasks like music transcription [32], singing voice extraction [7] or audio mosaicing [8]. In [30] the decomposition of drum recordings into the different drum components is discussed. In this thesis, we cover the application to piano recordings, where we want to separate the left hand typically playing the accompaniment from the right hand typically playing the melody. This means we want to extract separate signals for the left and the right hand from the same piano recording.

One algorithm which has been widely used in audio decomposition is nonnegative matrix factorization (NMF). It tries to produce a low-rank approximation of a nonnegative matrix by means of a product of two nonnegative matrices. Using this algorithm for music decomposition, we can factorize a spectrogram into a set of frequency templates and the according activation patterns over time. When using multiplicative updates [18] for the computation of the matrices we speed up the convergence of the algorithm and can easily introduce score information into the algorithm in order to achieve musically meaningful results, as shown for our scenario in [9]. We can then use these results in combination with the annotation data to decompose the recording into the desired separated signals for the two hands.

In recent years, neural networks produced outstanding results for applications as image recognition [15] or automatic speech recognition [13]. This is also the case in the area of music information retrieval [3]. As shown in [28], we can use nonnegative autoencoders (NAEs) to learn nonnegative audio models comparable to matrix factorization. NAEs are autoencoder networks with additional nonnegativity constraints. If we additionally introduce musically informed constraints into the network, as shown in [10] and [30], we can produce results similar to the ones resulting from musically informed NMF. These results can then similarly by applied for the separation of left and right hand in piano recordings. Advantages of these networks compared to NMF are, e. g., the easier applicability to bigger datasets due to the smaller number of parameters or the possibility to increase the complexity of the models by increasing the number of layers. One disadvantage however is the slower convergence of the networks compared to NMF.

In this thesis, we apply NMF and NAEs to a music decomposition scenario, building upon the results by Suárez in [30]. We apply their approach to a piano dataset and contribute our own extensions to their work on this technique. The main contributions of this thesis are as follows: we apply NMF to the piano scenario and investigate the convergence behavior of different approaches. From this, we conclude that both the multiplicative updates and the musically informed constraints speed up the convergence of the algorithm. We also observe that the impact of the update rules on this speed-up is significantly higher than for the constraints. Furthermore, we study the practically occurring problem of a rising loss value for NMF with multiplicative updates and conclude that the problem is caused by values of zero or close to zero in the initialization. Afterwards, we apply NAEs with additional musically informed constraints to the piano scenario and study the convergence behavior. As our most important contribution, we derive multiplicative update rules for the weights of the neural network inspired by the multiplicative updates for NMF. We show that these new update rules have effects comparable to the ones for NMF: they speed up the convergence of the network by a factor of ten and simplify imposing constraints on the network weights. Building upon these new update rules, we also propose possible informed initializations for the encoder weights. Furthermore, we apply the approaches to a bigger dataset and see that using our multiplicative update rules yields promising results compared to other NAE-based approaches. When jointly processing the whole dataset at once, we experience some problems regarding the quality of the decomposition results for all NMF and NAE approaches. We assume this to be caused by the differences in piano model and tuning between the different examples of the dataset.

## 1.1 Thesis Organization

This thesis is organized as follows:

In Chapter 2, we cover various aspects of NMF. We introduce the NMF algorithm and explain the theory behind it in Section 2.1. Subsequently, we show how to apply it to the music decomposition scenario in Section 2.2. Our own contributions are shown in Sections 2.3 and 2.4. In Section 2.3, we discuss the convergence behavior of NMF and what components of the algorithm speed up the convergence the most. In Section 2.4, we investigate the problem of rising loss values which occurred during our experiments.

We talk about the core topic of this thesis, namely neural networks, in Chapter 3. In Section 3.1, we explain the theory behind NAEs and introduce the used network structure. Subsequently, we show how to apply NAEs to music decomposition in Section 3.2. Sections 3.3 and 3.4 contain our own contributions regarding this technique. In Section 3.3, we derive multiplicative update rules for the neural network and discuss their properties and behavior. Building upon this, we compare different initializations for the encoder weights in Section 3.4.

We cover the application of the algorithms to a bigger dataset in Chapter 4. We introduce the used dataset in Section 4.1. Afterwards, we discuss the results for the different algorithms in Section 4.2. Furthermore, we show the structure and contents of Jupyter notebooks provided complementary to this report and talk about the quality of the audio results in Section 4.3.

Finally, we draw the conclusions of this thesis and give an outlook for future work in Chapter 5 and give details on the implementations of the different algorithms in Appendix A.

# Chapter 2

# Nonnegative Matrix Factorization (NMF)

Nonnegative matrix factorization (NMF) is a long-known technique for decomposing signals of various kinds. In this thesis, we focus on audio signals, more specifically on music recordings. However, NMF has also been applied to various other types of signals, such as images [12], infrared [33] or astrophysical data [1].
This chapter is organized as follows: In Section 2.1, we explain the basic theory behind NMF as well as the most common NMF algorithm. Subsequently, we show how NMF can be applied to music decomposition in Section 2.2. Furthermore, we cover various aspects of the convergence behavior of NMF in Section 2.3. Finally, we discuss the problem of the rising loss in Section 2.4.

## 2.1 Theoretic Foundations

In this section, we introduce the theoretic foundations of NMF, closely following Section 8.3.1 of [24].
The main idea behind NMF is to approximate a nonnegative matrix $V$ by a product of two matrices, $W$ and $H$, which are also nonnegative. A matrix is called nonnegative, when all of its entries are nonnegative, i. e., they are either zero or positive. $W$ is commonly called the template matrix, as it contains the basis vectors to represent the columns of $V$, and $H$ the activation matrix, as it contains the activations for the corresponding basis vectors. Usually, these matrices are of smaller size than the original matrix. Therefore, NMF can in this case also be seen as a compressed representation of the original input matrix.

### 2.1.1 Mathematical Definition

We want to approximate a nonnegative matrix $V \in \mathbb{R}_{\geq 0}^{K\times N}$, consisting of $N$ $K$-dimensional columns, by a factorization into $W \in \mathbb{R}_{\geq 0}^{K\times R}$ and $H \in \mathbb{R}_{\geq 0}^{R\times N}$:

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

The value $R$ defines the rank of the approximation and is typically lower than both $K$ and $N$. This rank defines the number of templates (i. e. columns of $W$) to be trained during NMF. To get an intuition for the mathematical formulation of NMF, we can look at the column-wise definition:

$$v \approx Wh. \tag{2.2}$$

Here, $v$ denotes the $n$-th column of $V$ and $h$ the $n$-th column of $H$. We see, that $v$ is approximated by a weighted sum over the columns of $W$, where the weights are given by the respective entries in $h$. Due to the nonnegativity constraints for the entries of $W$ and $h$, there are no cross-cancellation effects between the templates when performing the sum. Therefore, the superposition of them is purely constructive. The approximation of the matrix $V$ is from now on also referred to as $\hat{V}$ in this thesis. In the case of NMF, it is defined as:

$$\hat{V} := WH. \tag{2.3}$$

After introducing the NMF-based factorization of the input matrix $V$, we are now going to formulate NMF as an optimization problem. First of all, we have to define a cost function, which we want to optimize:

$$\mathcal{L}(V, \hat{V}) = \|V - \hat{V}\|^2 := \sum_{k=1}^{K} \sum_{n=1}^{N} (V_{kn} - \hat{V}_{kn})^2. \tag{2.4}$$

This function is also known as the squared Euclidean distance in the $K \times N$-dimensional space. Thus, we measure the quality of the approximation by means of the distance with respect to the true matrix $V$. When the distance between $V$ and $\hat{V}$ is small, the approximation $\hat{V}$ is more similar to the original $V$. This function is referred to as the cost or the loss function in this thesis. Note that other popular choices are the Kullback-Leibler divergence [6], the Itakura-Saito divergence [19] or the more generalized $\beta$-divergence [11]. However, we only focus on the squared Euclidean distance in this thesis.

To get an approximation close to the original $V$ we need to minimize this cost function. We can rewrite the cost function as

$$\mathcal{L}(W, H) = \|V - WH\|^2. \tag{2.5}$$

Here, $V$ is our fixed input and $W$ and $H$ are our variables to optimize. Thus, we can write the NMF problem as

$$\min_{W \in \mathbb{R}_{\geq 0}^{K \times R}, H \in \mathbb{R}_{\geq 0}^{R \times N}} \mathcal{L}(W, H). \tag{2.6}$$

### 2.1.2 Gradient Descent

One possibility to find a (locally) optimal solution to Problem (2.6) is the use of the gradient descent algorithm. The idea behind this is to tweak the optimizable parameters towards the direction of the steepest descent of the loss function. This direction is found by taking the negative gradient of said function w.r.t. the weights we want to change. Furthermore, the step size is usually regulated by a parameter $\gamma \geq 0$. This parameter will be referred to as the learning rate throughout this thesis. The update rules for $W$ and $H$ are therefore given by

$$W^{(\ell+1)} = W^{(\ell)} - \gamma \cdot \frac{\partial \mathcal{L}(W, H)}{\partial W} \tag{2.7}$$

and

$$H^{(\ell+1)} = H^{(\ell)} - \gamma \cdot \frac{\partial \mathcal{L}(W, H)}{\partial H}, \tag{2.8}$$

where $\ell$ denotes the index of the iteration. In order to calculate the gradient, we first rewrite the loss function as an explicit function of the coefficients of the matrices $W$ and $H$:

$$\mathcal{L}(W, H) = \|V - \hat{V}\|^2 = \sum_{k=1}^{K} \sum_{n=1}^{N} (V_{kn} - \hat{V}_{kn})^2 = \sum_{k=1}^{K} \sum_{n=1}^{N} \left( V_{kn} - \sum_{r=1}^{R} W_{kr} H_{rn} \right)^2 \tag{2.9}$$

For the calculation of the gradient for a specific coefficient $H_{\rho\nu}$, we calculate the derivative of the loss function with respect to this coefficient:

$$\begin{aligned}
\frac{\partial \mathcal{L}}{\partial H_{\rho\nu}} &= \frac{\partial \left( \sum_{k=1}^{K} \sum_{n=1}^{N} \left( V_{kn} - \sum_{r=1}^{R} W_{kr} H_{rn} \right)^2 \right)}{\partial H_{\rho\nu}} && (2.10) \\
&= \frac{\partial \left( \sum_{k=1}^{K} \left( V_{k\nu} - \sum_{r=1}^{R} W_{kr} H_{r\nu} \right)^2 \right)}{\partial H_{\rho\nu}} && (2.11) \\
&= \sum_{k=1}^{K} 2 \Big( V_{k\nu} - \sum_{r=1}^{R} W_{kr} H_{r\nu} \Big) \cdot (-W_{k\rho}) && (2.12) \\
&= 2 \Big( \sum_{r=1}^{R} \sum_{k=1}^{K} W_{k\rho} W_{kr} H_{r\nu} - \sum_{k=1}^{K} W_{k\rho} V_{k\nu} \Big) && (2.13) \\
&= 2 \Big( \sum_{r=1}^{R} \Big( \sum_{k=1}^{K} W_{\rho k}^{\top} W_{kr} \Big) H_{r\nu} - \sum_{k=1}^{K} W_{\rho k}^{\top} V_{k\nu} \Big) && (2.14) \\
&= 2 \big( (W^{\top} W H)_{\rho\nu} - (W^{\top} V)_{\rho\nu} \big) && (2.15)
\end{aligned}$$

We start with (2.10), based on (2.9). Next, we derive (2.11) by using the fact that the derivative is zero for all summands not depending on $\nu$ (and therefore also not on $H_{\rho\nu}$). Then, we apply the chain rule for derivatives to obtain (2.12). Subsequently, we use the transpose matrix $W^{\top}$ and rearrangements of the sums to arrive at the matrix-based formulation of the gradient in (2.15). Finally, for a fixed value of matrix $W$, we can formulate the gradient descent update rule for the elements of matrix $H$ as

$$H_{rn}^{(\ell+1)} = H_{rn}^{(\ell)} - \gamma_{rn}^{(\ell)} \cdot \Big( \big( W^{\top} W H^{(\ell)} \big)_{rn} - \big( W^{\top} V \big)_{rn} \Big), \qquad (2.16)$$

where $H^{(0)} \in \mathbb{R}^{R \times N}$ denotes the initialization of $H$. Note that we choose the step size $\gamma_{rn}^{(\ell)}$ to be dependent on the iteration $\ell$ and the indices $r$ and $n$ of the matrix entry. Therefore, (2.16) is a more generalized formulation of the gradient descent update rule. Furthermore, we omit the factor 2 occurring in (2.15) in the following as it can be removed by either including it in the learning rate or adding a factor of $\frac{1}{2}$ to the loss function.
By performing similar derivations for the entries of matrix $W$, which we omit for the sake of brevity, we can analogously formulate the update rule for $W$ as

$$W_{kr}^{(\ell+1)} = W_{kr}^{(\ell)} - \gamma_{kr}^{(\ell)} \cdot \Big( \big( W^{(\ell)} H H^{\top} \big)_{kr} - \big( V H^{\top} \big)_{kr} \Big), \qquad (2.17)$$

where $W^{(0)} \in \mathbb{R}^{K \times R}$ is the initialization of matrix $W$.
In order to update both matrices $W$ and $H$ with these update rules, we choose to use an alternating procedure. For each iteration $\ell$, we first fix matrix $W$ and update matrix $H$ using (2.16). Then, we switch the roles and update $W$ using (2.17). We repeat these steps for all iterations $\ell = 0, 1, 2, ..., L-1$, where $L \in \mathbb{N}$ is the total number of iterations. Our goal is that

the matrices converge to a globally optimal solution.
There are still some problems with this procedure. First, we have to experimentally find suitable learning rates $\gamma_{rn}^{(\ell)}$ and $\gamma_{kr}^{(\ell)}$. This is typically done by choosing small positive values which are constant for all iterations and matrix entries.
Second, even when choosing nonnegative initializations $H^{(0)}$ and $W^{(0)}$, we cannot guarantee that our updates will not result in negative entries for the matrices $W^{(\ell)}$ and $H^{(\ell)}$. This is a problem as $W$ and $H$ in the original NMF Problem (2.6) are constrained to be nonnegative. In this thesis, we solve this problem by using a projected gradient method [20]. In each iteration, after calculating the updated matrices, we set all negative entries to zero. By doing so, we project the entries back to the nonnegative space.
Third, we cannot ensure to find the globally optimal solution. The cost function is not jointly convex and therefore, the algorithm may only end up in a local minimum. These locally optimal solutions are not unique as the approximation $\hat{V}$ is invariant to scaling of the decomposition factors. Furthermore, the estimated solution heavily depends on the initialization.

### 2.1.3 Multiplicative Update Rules

In [18], Lee and Seung proposed choices for the learning rates $\gamma_{rn}^{(\ell)}$ and $\gamma_{kr}^{(\ell)}$. The idea is to set the learning rates such that the additive updates become multiplicative ones. For this, the learning rate for matrix $H$ is set to

$$\gamma_{rn}^{(\ell)} = \frac{H_{rn}^{(\ell)}}{\left(W^\top W H^{(\ell)}\right)_{rn}}, \tag{2.18}$$

which results, when inserted into (2.16), in the new update rule

$$H_{rn}^{(\ell+1)} = H_{rn}^{(\ell)} \cdot \frac{\left(W^\top V\right)_{rn}}{\left(W^\top W H^{(\ell)}\right)_{rn}}. \tag{2.19}$$

Analogously, we can define a learning rate

$$\gamma_{kr}^{(\ell)} = \frac{W_{kr}^{(\ell)}}{\left(W^{(\ell)} H H^\top\right)_{kr}} \tag{2.20}$$

for matrix $W$, insert it into (2.17) and arrive at the update rule

$$W_{kr}^{(\ell+1)} = W_{kr}^{(\ell)} \cdot \frac{\left(V H^\top\right)_{kr}}{\left(W^{(\ell)} H H^\top\right)_{kr}}. \tag{2.21}$$

```
Algorithm: NMF ($V \approx WH$)

Input:   Nonnegative matrix $V$ of size $K \times N$
         Rank parameter $R \in \mathbb{N}$
         Number of iterations $L \in \mathbb{N}$
Output:  Nonnegative template matrix $W$ of size $K \times R$
         Nonnegative activation matrix $H$ of size $R \times N$

Procedure: Define nonnegative matrices $W^{(0)}$ and $H^{(0)}$ by some random
or informed initialization. Furthermore set $\ell = 0$. Apply the following
update rules (written in matrix notation):

    (1)  $H^{(\ell+1)} = H^{(\ell)} \odot \big(((W^{(\ell)})^\top V) \oslash ((W^{(\ell)})^\top W^{(\ell)} H^{(\ell)} + \varepsilon)\big)$
    (2)  $W^{(\ell+1)} = W^{(\ell)} \odot \big((V(H^{(\ell+1)})^\top) \oslash (W^{(\ell)} H^{(\ell+1)} (H^{(\ell+1)})^\top + \varepsilon)\big)$
    (3)  Increase $\ell$ by one.

Repeat the steps (1) to (3) until $\ell = L$. Finally, set $H = H^{(L)}$ and
$W = W^{(L)}$.
```

Table 2.1: Iterative algorithm for learning an NMF decomposition. The multiplicative update rules are given in matrix notation, where the operator $\odot$ denotes pointwise multiplication and the operator $\oslash$ pointwise division.

These new update rules solve two problems mentioned in Section 2.1.2. First, we do not have to experimentally choose learning rates, as we implicitly choose them to be (2.18) and (2.20) by using the multiplicative update rules. Second, when initializing the matrices $W$ and $H$ with nonnegative values only, the multiplicative update rules ensure that the nonnegativity constraint will not be violated, as the product of nonnegative numbers always results in nonnegative values. The third problem however is not solved by these new update rules as the solution still does not need to be globally optimal.

Even though these new update rules may at first not seem very intuitive, Lee and Seung show in their paper [18] that the loss function (2.5) is nonincreasing for each update of $W$ and $H$ when using the update rules (2.19) and (2.21). They typically converge to a stationary point and show a faster convergence speed than other approaches like the additive updates in (2.16) and (2.17). We study the convergence behavior of NMF in more detail in Section 2.3.

The NMF algorithm which we use the most in this thesis is described in Table 2.1. We use the multiplicative update rules (2.19) and (2.21) to update the matrix weights. Note that due to the special form of the update equations, the updates of the weights for one matrix can be computed by one single computation. We add a machine epsilon $\varepsilon$ of $10^{-7}$ in the denominators of the update rules in order to avoid divisions by zero. Furthermore, we use a fixed number of iterations $L$ as the stopping criterion.

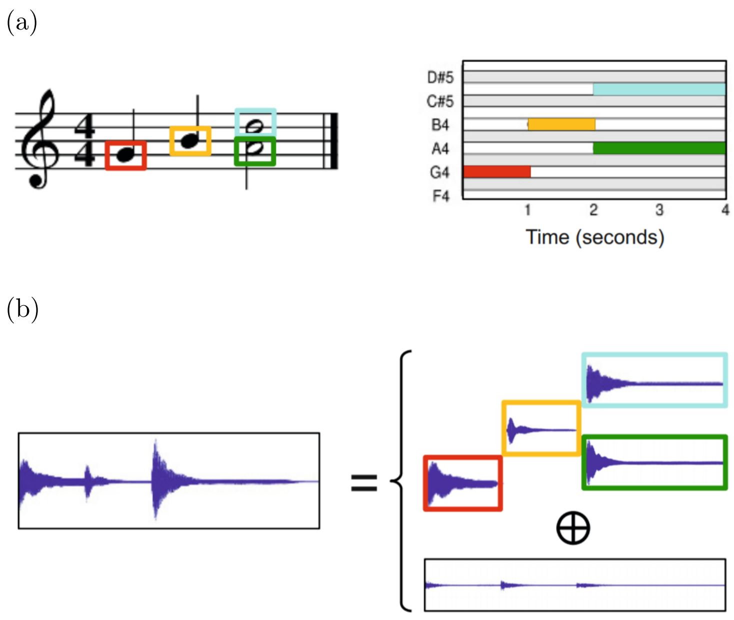


Figure 2.1: Piano decomposition example: (a) Symbolic representations, (b) Audio representation and decomposition into isolated note signals (red, yellow, blue and green) and residual signal (black) (Figure taken from [24]).

## 2.2 Application to Music Decomposition

In this section, we discuss the application of NMF to the scenario covered in this thesis and explain important musical and mathematical basics for later sections. We closely follow [9] as well as Sections 8.3.2 and 8.3.3 of [24].
The task in this thesis is the decomposition of music recordings. Music recordings and many other audio signals often contain a superposition of multiple sound sources. For the musical scenario, these sound sources may be different singers, instruments or even different components of one instrument. In this thesis, we work with piano recordings, where the sources are the left and the right hand of the pianist.
One common assumption is the applicability of a linear signal model. This means that we assume a digital audio signal $x \in \mathbb{R}^M$ of length $M \in \mathbb{N}$ consisting of $S \in \mathbb{N}$ sources can be written as

$$x = \sum_{s=1}^{S} x_s + e. \tag{2.22}$$

The isolated signal for source $s$ is denoted by $x_s \in \mathbb{R}^M$ and $e \in \mathbb{R}^M$ denotes the residual component typically capturing undesired sources like background noise. This principle is illustrated in Figure 2.1. The example shows a short excerpt of a piano piece containing four notes. Figure 2.1a shows

the symbolic representations, given by the score and the corresponding piano-roll representation. Figure 2.1b shows the according audio signal and its decomposition into the isolated signals for the four notes and the remaining residual signal. Note that this example has one key difference compared to the task we try to solve in this thesis: this example models every note as a separate source, whereas we model all notes played by the left hand as one and the notes played by the right hand as the other source. Therefore, we have two sources in total in our scenario instead of one source per note like we have in this example.
In order to use NMF for the decomposition of audio recordings, we need to convert the audio signal $x$ into a nonnegative matrix-based representation. Therefore, we represent our signals by their magnitude spectrograms in this thesis. We compute the Short-Time Fourier Transform (STFT) $\mathcal{X}$ of $x$ and subsequently compute the magnitude spectrogram $V$ as

$$V := |\mathcal{X}|^{\top}. \tag{2.23}$$

For further details on the STFT we refer to [24]. The size $K \times N$ of matrix $V$ depends on the parameters chosen for the computation of the STFT. In our experiments we use a sampling rate of 22050 Hz, a window length of 4096 and a hop size of 1024, thus resulting in a frequency resolution of 5.38 Hz and a time resolution of 46.44 msec. As the upper half of the frequency bins contains redundant information, we only use the lower half for our representation. This results in our parameter $K$ to be fixed to $K = 2049$. The parameter $N$ is dependent on the example as it depends on the length of the audio signal. The rank parameter $R$ is typically chosen such that it corresponds to the number of pitches present in the recording. For the example from Figure 2.1, this means that $R$ would be chosen to $R = 4$. The idea behind this is to have one frequency template for each pitch. However, with random initializations for $W$ and $H$, the learned decomposition is typically not that interpretable. A solution to this is shown in Section 2.2.1.

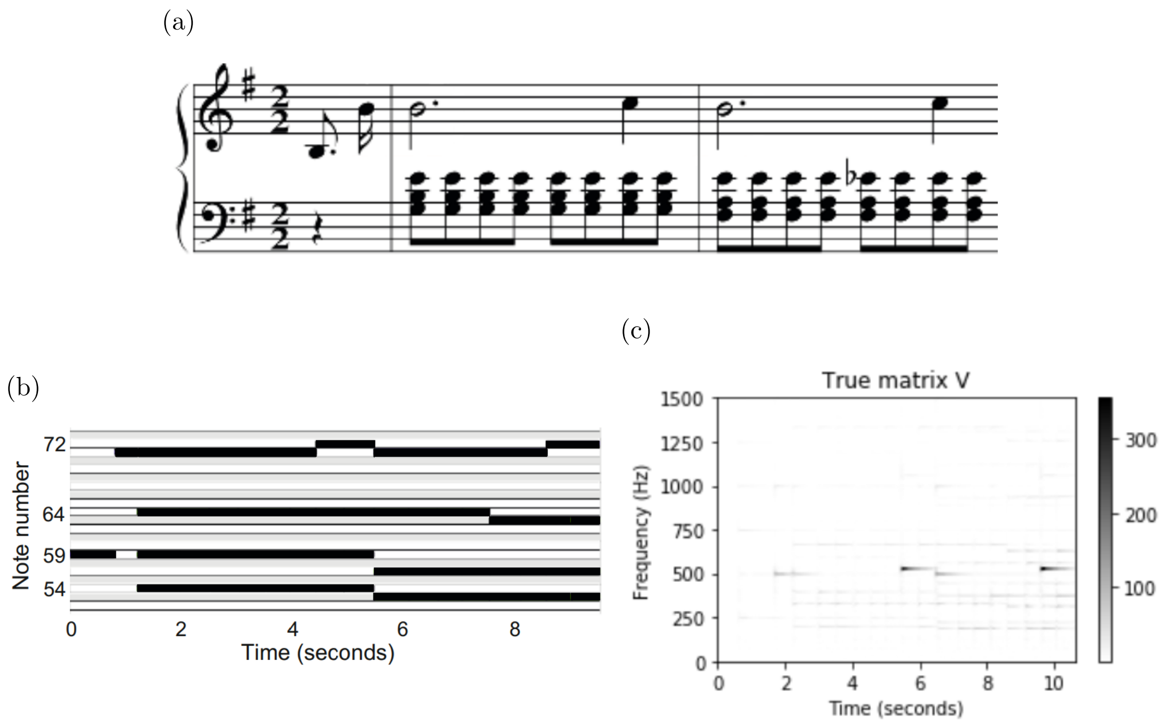
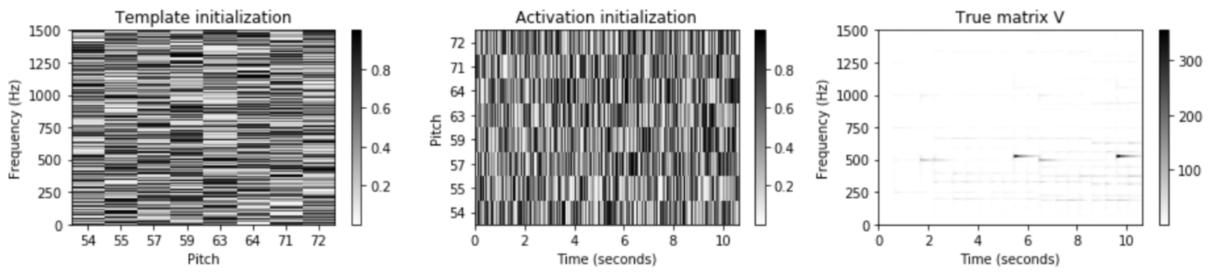


Figure 2.2: Representations of the running example: (a) Score, (b) Piano roll, (c) Magnitude spectrogram (Figures (a) and (b) taken from [24]).

In the following, we introduce the running example that we use throughout this thesis. It is a 10 second excerpt of the beginning of a performance of Chopin's Prélude Op. 28, No. 4. The according representations are shown in Figure 2.2. Figure 2.2a shows the score information for this example and Figure 2.2b the symbolic representation after converting it to a piano roll. Figure 2.2c shows the magnitude spectrogram resulting from the parameters mentioned before. Note that we use the whole frequency range with $K = 2049$ for our computations but only show the frequency range up to 1500 Hz for most of the figures as this range contains the most significant frequency components. This short example will be used as the standard application for the remainder of this chapter as well as for Chapters 3 and 4.

As can be seen, there are eight different pitches active in this example. Therefore, we choose the rank of our approximation to be $R = 8$ and initialize the matrices $W$ and $H$ as random nonnegative matrices of the according dimensions. We fix the number of iterations to be $L = 100$. The resulting matrices are shown in Figure 2.3 and we see that both the templates and the activations show a rather random behavior. These results yield a good approximation of matrix $V$ but are not easily interpretable. We would rather desire some harmonic structures in matrix $W$ and piano roll-like behavior comparable to Figure 2.2b for matrix $H$.

(a)



(b)

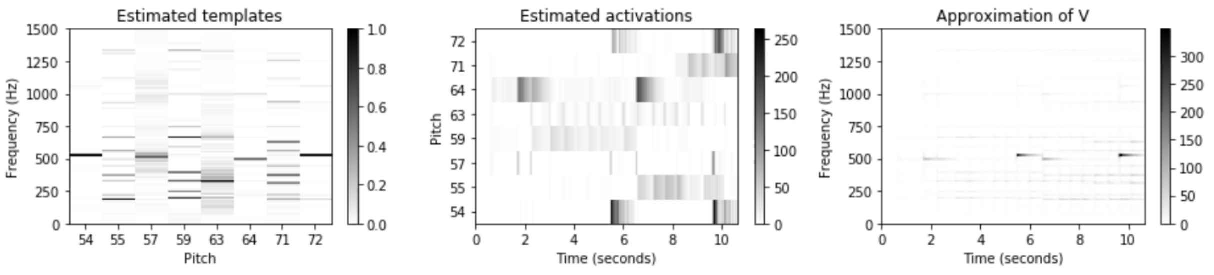


Figure 2.3: NMF-based decomposition with random initialization: (a) Initializations and input matrix $V$, (b) Resulting approximation.

### 2.2.1 Musically Informed NMF

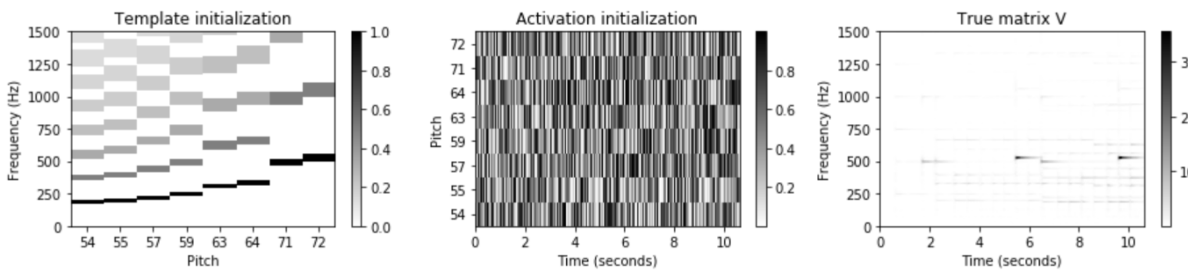
As we just showed, randomly initializing the matrices of the NMF factorization does not yield interpretable templates. One solution to this was introduced in [9]. The basic idea is to use the score information to impose additional musically informed constraints on $W$ and $H$. For both matrices, the constraints are based on enforcing certain areas of the matrices to be zero. This can easily be done through the initialization when using the multiplicative update rules (2.19) and (2.21). Initializing an entry of $W$ or $H$ by zero results in this entry remaining at zero throughout the entire optimization process as even multiplying an entry of zero by a positive value will still result in zero.

#### 2.2.1.1 Template Constraints

Harmonic signals have a very specific and sparse representation in the frequency domain. Let us consider a single MIDI pitch $p$. The frequency for this pitch for a 440 Hz tuning is given by

$$F_{\text{pitch}}(p) = 2^{(p-69)/12} \cdot 440 \text{ Hz}. \tag{2.24}$$

(a)



(b)

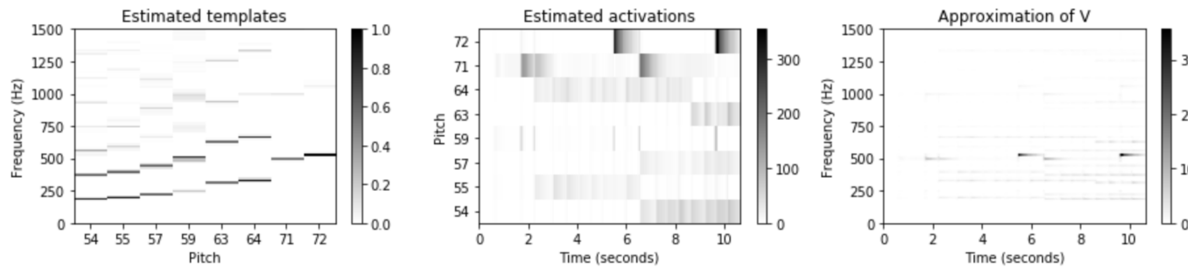


Figure 2.4: NMF-based decomposition with informed initialization for $W$: (a) Initializations and input matrix $V$, (b) Resulting approximation.

A piano recording containing only this pitch should typically only contain positive entries in its spectrogram at the harmonics $h \cdot F_{\text{pitch}}(p)$ for $h \in \mathbb{N}$. Therefore, the approach used in this thesis is to only initialize these frequency bins (plus some tolerance area to account for tuning deviations) of the template for pitch $p$ in matrix $W$ by a value $1/h$. All remaining entries are initialized by zero.

The initialization matrix for imposing informed constraints on $W$ as well as the resulting approximation after 100 iterations can be seen in Figure 2.4. It can be seen that the templates in matrix $W$ behave as desired, i. e., each template corresponds to the harmonics for one single pitch. However, there still are problems concerning the activations in $H$: there are positive activation values for time frames, when the notes are not played according to the annotations. One example for this can be seen for MIDI pitch 59, where we have undesired activations after the time stamp of seven seconds.

#### 2.2.1.2 Activation Constraints

In order to overcome the problem of undesired activity of templates, we also impose constraints on the activation matrix $H$. Once again, we use the information from the annotation data. By using time-aligned score information, we know when which note is played in the recording. Therefore,

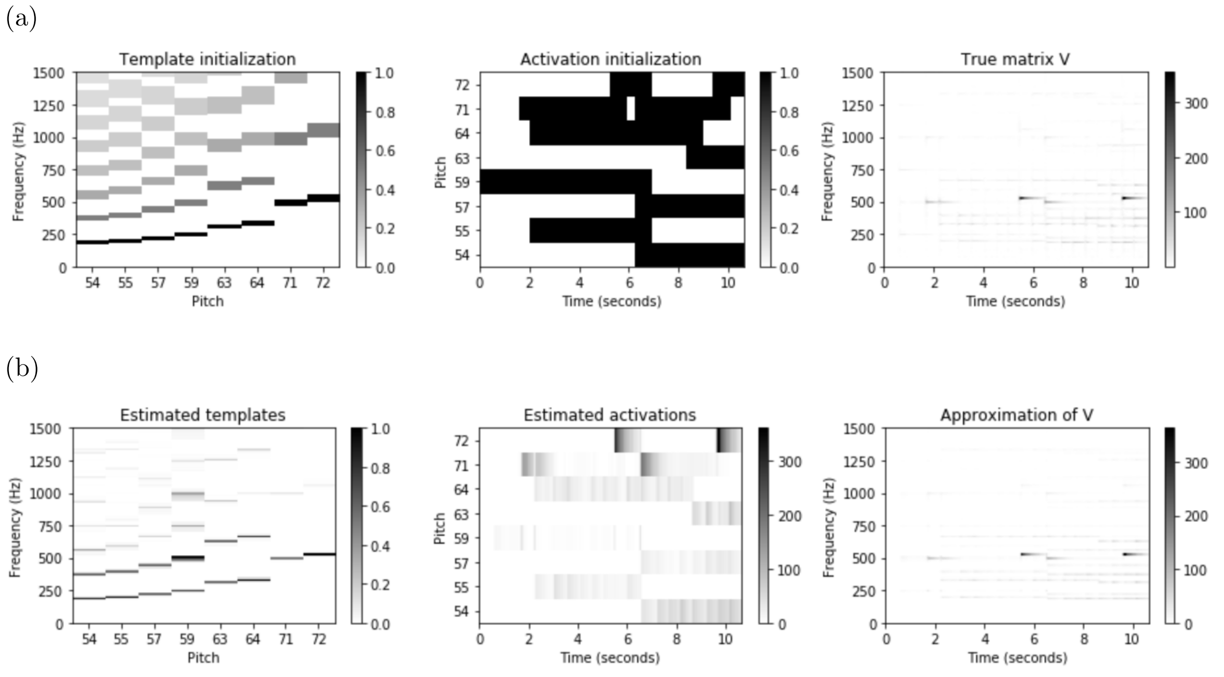


Figure 2.5: NMF-based decomposition with informed initialization for $W$ and $H$: (a) Initializations and input matrix $V$, (b) Resulting approximation.

we can constrain when which pitch template should be active. Accordingly, we initialize those entries of $H$ with one which we expect to contain activity according to the annotation (plus some tolerance to account for synchronization errors). All other entries are initialized by zero to avoid activities in undesired time frames.

The according initialization and resulting factorization, once again after 100 iterations, are plotted in Figure 2.5. It can be seen that we successfully removed the unwanted activations, e. g. for MIDI pitch 59 after the time stamp of seven seconds, compared to Figure 2.4. However, there are some time frames where several templates show unexpectedly high simultaneous activations. An example for this can again be seen for MIDI pitch 59. After the time stamp of six seconds, there are no more onsets for this pitch according to the annotation. However, there is still a high peak at the end of the unconstrained area of the activation pattern for this pitch template. This effect can be explained by the fact that playing a note on the piano does not only result in harmonic frequency components but also a noise-like transient component at the beginning of the note. This is also called onset and results from the hammers hitting the strings. As all the templates only contain sparse harmonic frequency content, there is no model for the transient component which activates a wide frequency range. Therefore, NMF tries to cover this by activating as many frequencies as possible through simultaneous activation of various harmonic templates.

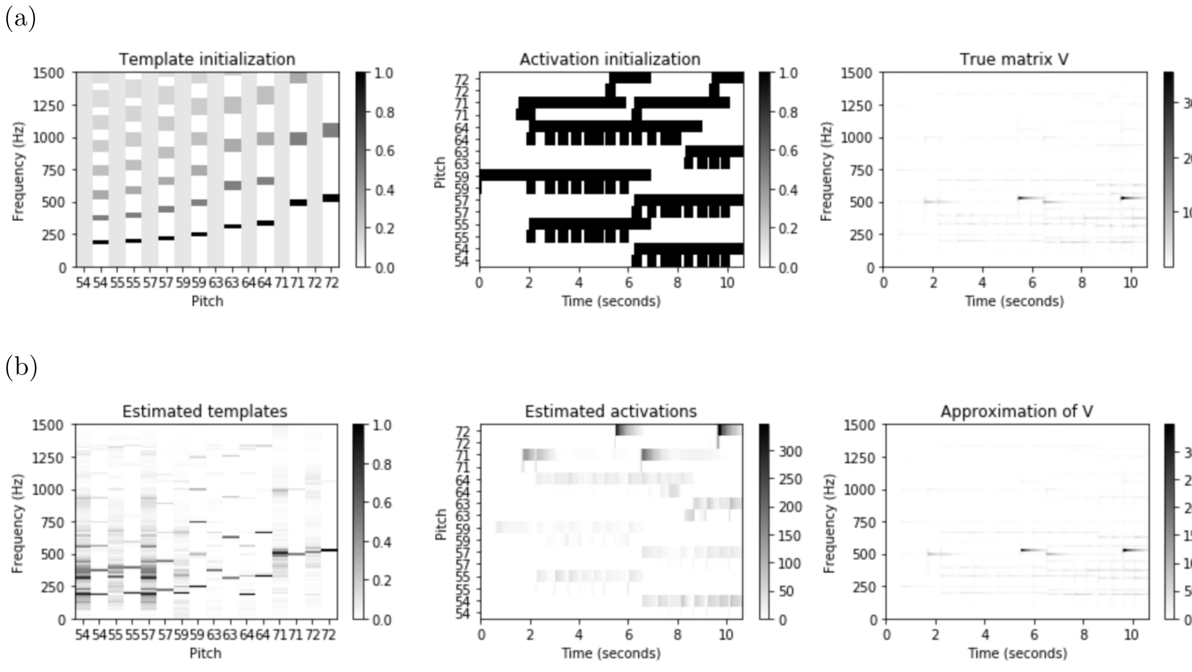


Figure 2.6: NMF-based decomposition including onset information: (a) Initializations and input matrix $V$, (b) Resulting approximation.

#### 2.2.1.3 Onset Models

A way to avoid the coverage of onsets by the simultaneous activity of multiple templates is to use additional models for the onset components of the pitches. Therefore, we have to slightly change our matrices $W$ and $H$. We adjust the rank of the approximation to now be two times the number of pitches present in the audio recording. The template matrix now contains two columns per pitch: one for the harmonic component, that we introduced in Section 2.2.1.1, and one additional column for the noise-like onset model. As we do not expect any specific harmonic behavior from these templates, we initialize all frequency bins by a constant value. We use a value of 0.1 in this thesis. The activation matrix now similarly contains two rows per pitch. The rows corresponding to the harmonic templates remain unchanged compared to Section 2.2.1.2. For the rows corresponding to the onset templates, we only initialize a tolerance area around the beginnings of note events by one, as these are the time frames where we want the onset templates to be active.

The resulting initialization and approximation matrices are shown in Figure 2.6. The results are as expected, with the activation matrix now only showing decaying activities at the desired time frames. The high activation peak after the time stamp of six seconds for the harmonic template of MIDI pitch 59 is gone as the onset for the other MIDI pitches is now modeled by their respective onset templates. Furthermore, we see that the onset templates do not show any harmonic behavior, but do still differ for different pitches regarding their frequency content.

As this initialization scheme shows the most desirable and interpretable results, we use it as the standard initialization for NMF in the remainder of this thesis, unless stated otherwise. We refer to it as the musically informed initialization.

### 2.2.2 NMF-based Audio Decomposition

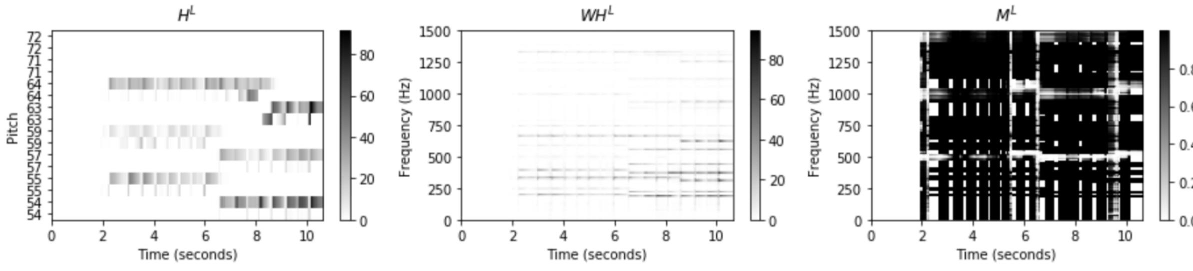
As we now have introduced a method to use musical information to guide the NMF algorithm, we show in this section how to use the resulting factorization to decompose music recordings. Therefore, we use the running example and separate the left hand signal from the one for the right hand. For this, we use annotation data where for each note event we do not only have information about pitch, beginning time and ending time, but also about which hand plays the note.

We start with the factorization resulting from musically informed NMF. The rows of the activation matrix tell us the estimated activity of the different templates over time. As each of these templates refers to one single pitch, we can use the annotation information to split the activation matrix $H$ into one matrix $H^{\mathrm{L}}$ containing the activations caused by the left hand and a respective matrix $H^{\mathrm{R}}$ for the right hand. This is done by applying a binary mask to the activation matrix. This mask is basically a separated version of the informed initialization of $H$. Using the separated activation matrices and the estimated templates from matrix $W$, we can approximate the reconstructed spectrograms for the left and the right hand signal by $WH^{\mathrm{L}}$ and $WH^{\mathrm{R}}$, respectively. We then compute spectral soft-masks for both components by computing the relative fraction of their contribution to the different time-frequency bins of the approximated total spectrogram:

$$M^{\mathrm{L/R}} = \left(WH^{\mathrm{L/R}}\right) \oslash (WH + \varepsilon)\,. \tag{2.25}$$

The constant $\varepsilon$ is once again a machine epsilon to avoid division by zero and $\oslash$ again denotes the pointwise division.

(a)



(b)

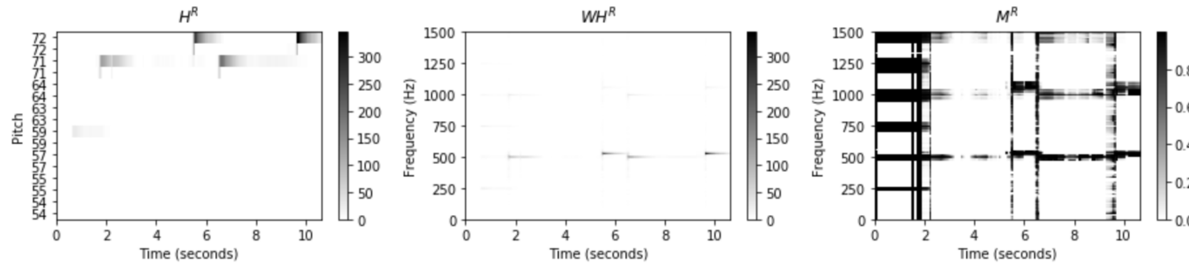


Figure 2.7: Split activation matrices, reconstructed spectrograms and spectral masks for separation of left and right hand: (a) Left hand, (b) Right hand.

The separated activation matrices as well as the resulting reconstructed spectrograms and spectral masks for both of the components are shown in Figure 2.7. We see that for the beginning of the excerpt, only the right hand is active. Furthermore, the left hand covers a bigger range of frequencies as more notes are played simultaneously by the accompaniment, which can also be seen in the score in Figure 2.2a.
Using these masks we can reconstruct the STFTs for the two components. For this, we simply apply the spectral masks by pointwise multiplication to the STFT $\mathcal{X}$ of the original audio signal. In order to convert these STFTs to audio signals for the separated sources, we use an approximation of the inverse of the STFT algorithm (for more details see Section 8.1.2.2 of [24]).

We use this pipeline for the separation and reconstruction of the left and right hand signals for all the decomposition approaches used in the remainder of this thesis.

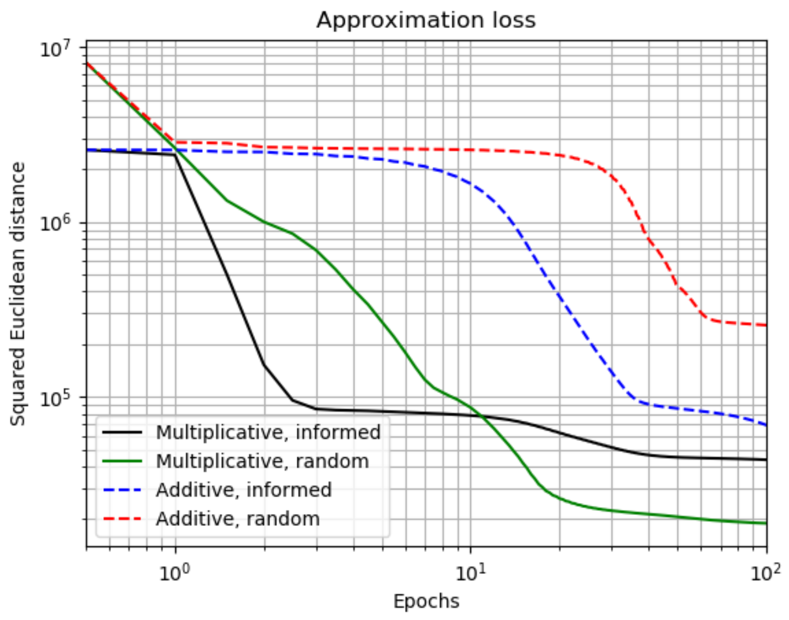


Figure 2.8: Loss curves for four NMF approaches.

## 2.3 Convergence of NMF

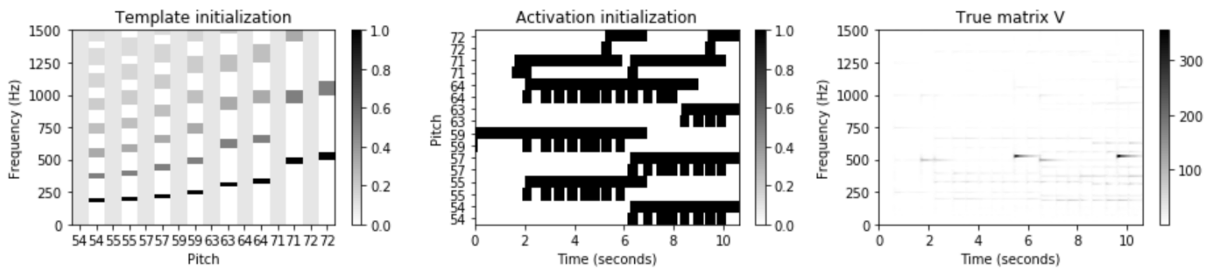
In the previous sections of this chapter, we showed how the NMF algorithm works and how to use it for music decomposition. In this section, we want to do a more in-depth study of the convergence behavior of different NMF algorithms. We want to compare on the one hand the effect of multiplicative versus additive updates and on the other hand the differences caused by using musically constrained instead of randomly initialized NMF. Therefore, we compare different metrics when applying all four possible combinations to the running example. For the additive NMF approaches, we use a constant learning rate of $\gamma = 10^{-4}$. Many of the metrics introduced in this section are also used in later parts of this thesis.

### 2.3.1 Loss Curves

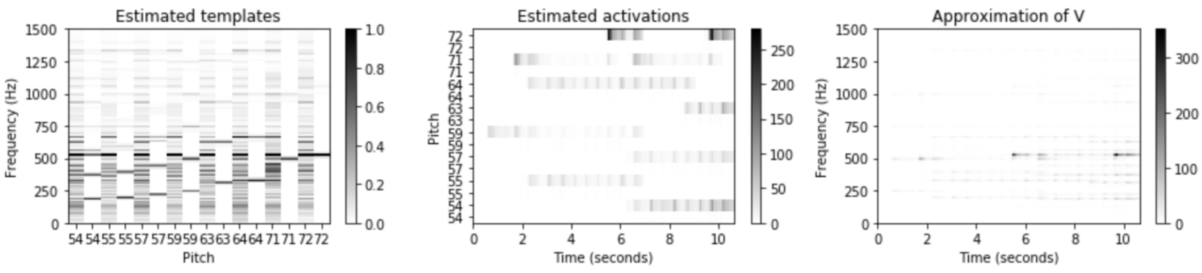
One of the most used metrics to investigate the convergence of algorithms, especially in deep learning scenarios, is the loss curve. In this thesis we only use one loss function, namely the squared Euclidean distance defined in Equation (2.4). It is the measure we want to minimize with our algorithms. Therefore, it may be an indicator for how fast and how good of a minimum our approach is able to find. The loss curves for the four approaches are plotted for the running example in Figure 2.8. Note that we compute a new loss value after every update of $W$ and $H$, respectively, and that the loss value at iteration 0.5 shows the loss resulting from using the initial values of $W$ and $H$ for the approximation of $V$. For the additive updates, we may encounter negative matrix entries after updating the matrices. In order to solve this problem, we set negative matrix entries to zero. Also note that we use the terms epoch and iteration interchangeably in this thesis. We can see that the multiplicative update rules show a faster convergence than the additive ones. Furthermore, the musically informed initialization seems to further speed up the optimization process as it already starts with a rough direction for the desired solution. This can also be seen in the initial loss values: the informed approaches start at a lower loss than the random approaches as the informed initialization already gives a rough approximation of the original matrix $V$. However, the effect on the convergence speed is not as strong as the one of the multiplicative updates. Another interesting result can be seen by comparing the two curves for the multiplicative update rules. Even though the random initialization takes longer to converge to a solution, it results in a smaller loss value. This can be explained by the fact that the optimization is not as constrained as for the musically informed initialization. For the convergence of the loss function, we can conclude that the high convergence speed is mostly achieved by the multiplicative update rules, while the musically informed initialization yields an also significant but comparably small speed-up.

When observing the curve for the combination of multiplicative update rules and musically informed initialization in Figure 2.8, we see that the loss seems to be mostly converged after one or at most ten iterations. In order to investigate the number of iterations needed for a good decomposition result we show the approximations for different numbers of iterations in Figure 2.9. We see that the results after only one iteration already look similar to the ones for a higher number of iterations but still need further refinement. The results for ten and 100 iterations however look almost equal. This is also reflected in the separation results. When listening to the audio signals, the results after one iteration still show a quite high amount of leakage. The results for ten and 100 iterations sound equally well with a low level of leakage.

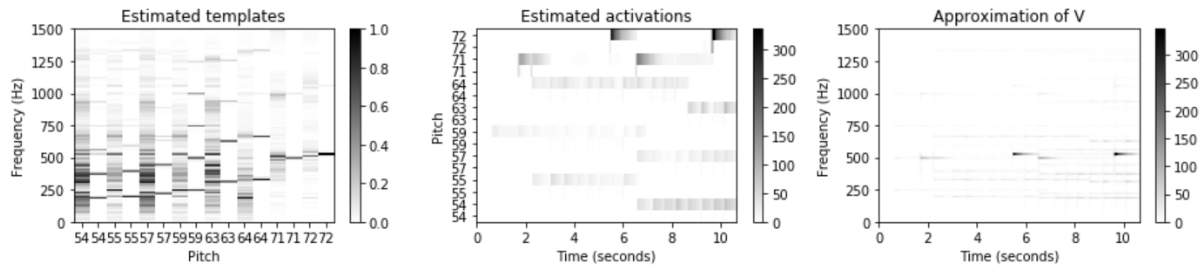
(a)



(b)



(c)



(d)

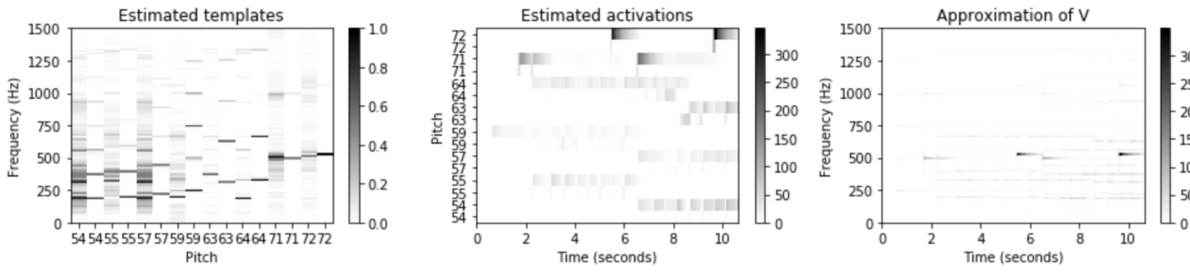


Figure 2.9: NMF-based decomposition using multiplicative update rules and musically informed initialization for different numbers of iterations: (a) Initialization and input matrix $V$, (b) 1 iteration, (c) 10 iterations, (d) 100 iterations.

### 2.3.2 Cauchy Errors

Another often used metric for studying convergence behavior are Cauchy errors. They measure the changes of weights between iterations by computing the Euclidean distance between the weight matrices for two subsequent iterations:

$$C_W(\ell) = \left|\left|W^{(\ell)} - W^{(\ell-1)}\right|\right|. \tag{2.26}$$

Therefore, the lower this metric, the smaller the changes to the optimizable weights, which means the algorithm seems to be converging to a solution. The same metric can analogously be applied to matrix $H$.

The resulting Cauchy errors are shown in Figure 2.10 for matrix $W$ and for matrix $H$ in Figure 2.11. We can see that the multiplicative updates show mostly monotonically decreasing Cauchy errors. The additive update rules show rather fluctuating, but still slightly decreasing Cauchy errors. Note that the differences in orders of magnitude between the curves for the Cauchy errors of one matrix are caused by the fact that we do not use any normalization during the optimization procedure. As mentioned in Section 2.1.2, the solutions to the NMF are not unique. The approximation is not affected by scaling of the factors as a scaling of a template by a certain factor results in the same approximation $\hat{V}$ when the corresponding activation pattern is scaled by the inverse of the factor. Therefore, the corresponding differences in orders of magnitude between the curves for matrix $W$ are reversed for the differences between the curves for matrix $H$.

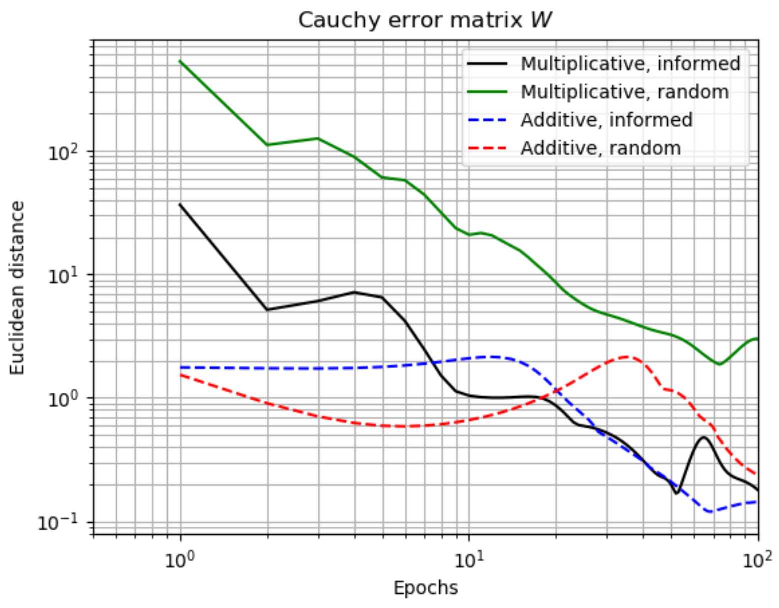


Figure 2.10: Cauchy errors for matrix $W$ for four NMF approaches.

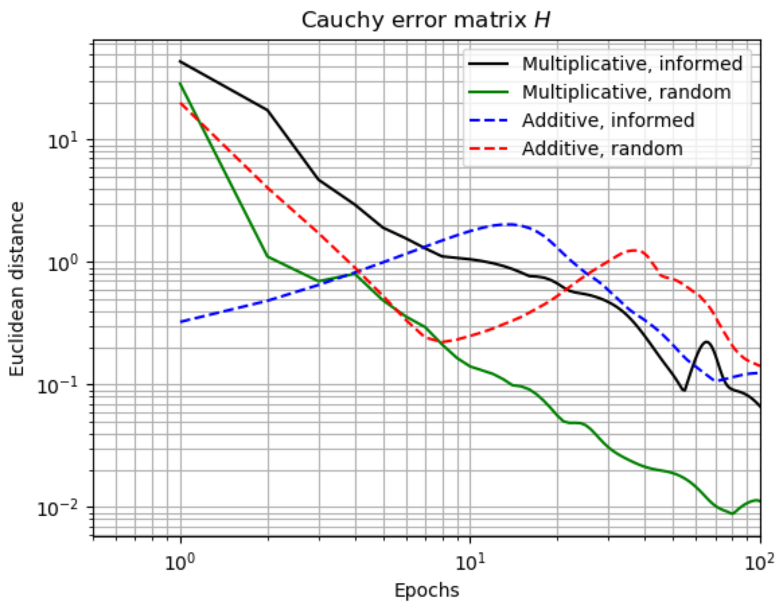


Figure 2.11: Cauchy errors for matrix $H$ for four NMF approaches.

Furthermore, we can also apply this measure to the approximation $WH$. To do so, we calculate the difference between complete iterations, i. e., after both $W$ and $H$ have been updated:

$$C_{WH}(\ell) = \left|\left|W^{(\ell)}H^{(\ell)} - W^{(\ell-1)}H^{(\ell-1)}\right|\right|. \tag{2.27}$$

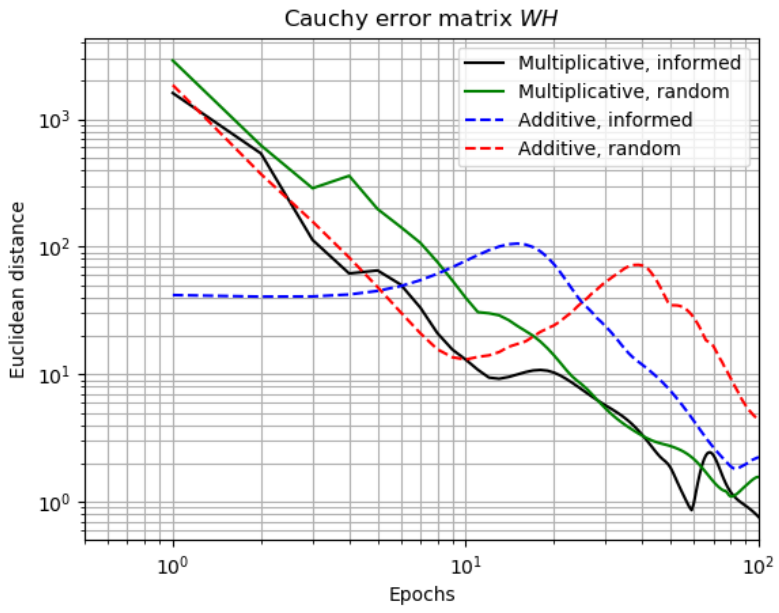


Figure 2.12: Cauchy errors for matrix $WH$ for four NMF approaches.

The resulting curves are plotted in Figure 2.12. The Cauchy errors show similar behavior as in Figures 2.10 and 2.11. Additionally, the curves do not show any differences in orders of magnitude as matrix $WH$ is not affected by any normalization of the factors.

### 2.3.3 Learning Rates

As we showed in Section 2.1.3, the multiplicative update rules can also be interpreted as additive update rules with adaptive learning rates. Therefore, we can investigate the behavior of these learning rates throughout the iterations of the algorithm. In order to compress the learning rate for one complete matrix into one single number per iteration, we arrange the learning rates as a matrix and calculate its $l_2$-norm. Using Equation (2.20) we can calculate the norm of the learning rate for matrix $W$ as

$$\gamma_W(\ell) = \sqrt{\sum_{k=1}^{K} \sum_{r=1}^{R} \left(\gamma_{kr}^{(\ell)}\right)^2}. \tag{2.28}$$

The computation of the learning rate for matrix $H$ can be done analogously. For the case of a constant learning rate $\gamma_{kr/rn}^{(\ell)} = \gamma$, this norm is simply a scaled version of the constant learning: $\gamma_W(\ell) = \gamma \cdot \sqrt{KR}$.

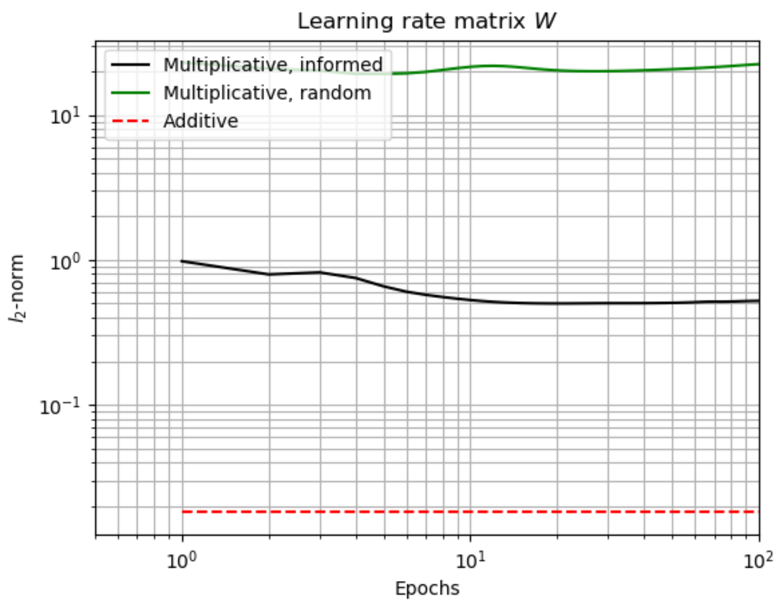


Figure 2.13: Learning rates for matrix $W$ for four NMF approaches.

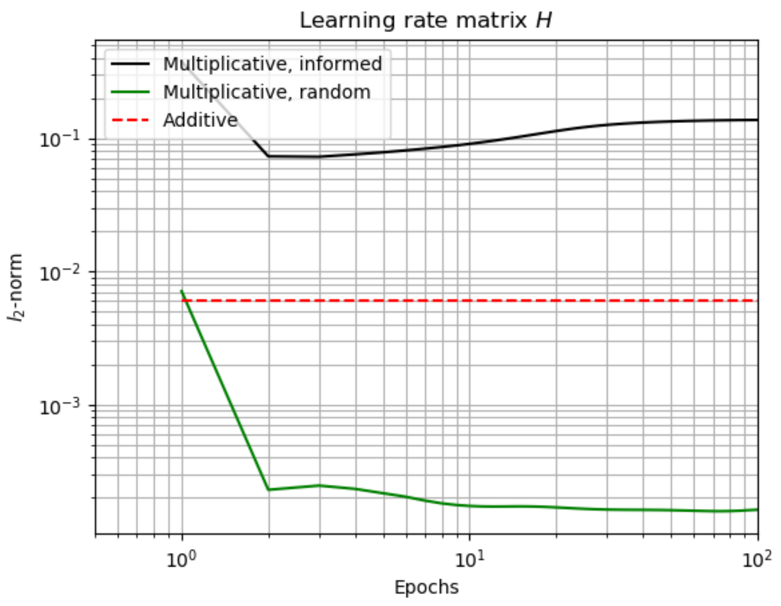


Figure 2.14: Learning rates for matrix $H$ for four NMF approaches.

The learning rates for the different scenarios are shown in Figures 2.13 and 2.14 for matrices $W$ and $H$, respectively. Note that the curves are again differing in orders of magnitude because we apply no normalization. We observe a decay of the learning rates during the first iterations of the multiplicative update rules. Furthermore, there seems to be a slight increase at later iterations. But mostly, there are no bigger changes of the learning rate throughout the optimization as there would be when applying learning rate decay, which is a common practice in deep learning.

Therefore, we can conclude that the fast convergence of the multiplicative update rules does not result from a continuous decay of the average learning rate. It thus seems that it is rather caused by having different learning rates for the individual matrix entries.

### 2.3.4 Update Step Behavior

After investigating how well the different approaches converge, we want to try to investigate what happens when the algorithm finds a minimum. Does it oscillate around it or always move in the same direction with decreasing step sizes? And how many updates above some significant threshold do we encounter in each iteration?
In order to find out whether the algorithm oscillates around a minimum, we will investigate the number of sign changes of the gradients. This means, for the case of matrix $W$, we count for each iteration the number of entries where

$$\operatorname{sign}\left(W_{kr}^{(\ell+1)} - W_{kr}^{(\ell)}\right) \cdot \operatorname{sign}\left(W_{kr}^{(\ell)} - W_{kr}^{(\ell-1)}\right) = -1. \tag{2.29}$$

Here, the sign-function yields a value of one for positive numbers and a value of negative one for negative numbers. For a value of zero, we define it to be zero. Therefore, we count the entries where the value is increased in one and decreased in the next step, or vice-versa.

We show the resulting curves for the template and activation matrix in Figures 2.15 and 2.16, respectively. We see that for matrix $W$, the number of sign changes is higher for multiplicative than for additive update rules. For matrix $H$ the numbers are roughly in the same range for all approaches. Note that there is a vertical line in the red curve in Figure 2.16, because there were no sign changes for matrix $H$ for the additive approach with random initialization before iteration three. However, the number of sign changes decreases faster for the multiplicative updates for both matrices. Finally, for all approaches and both matrices, we can see that only the gradients w.r.t. about 1% of the entries experience sign changes, from which we can conclude that the solutions converge mostly smoothly to the minimum rather than oscillating around it.

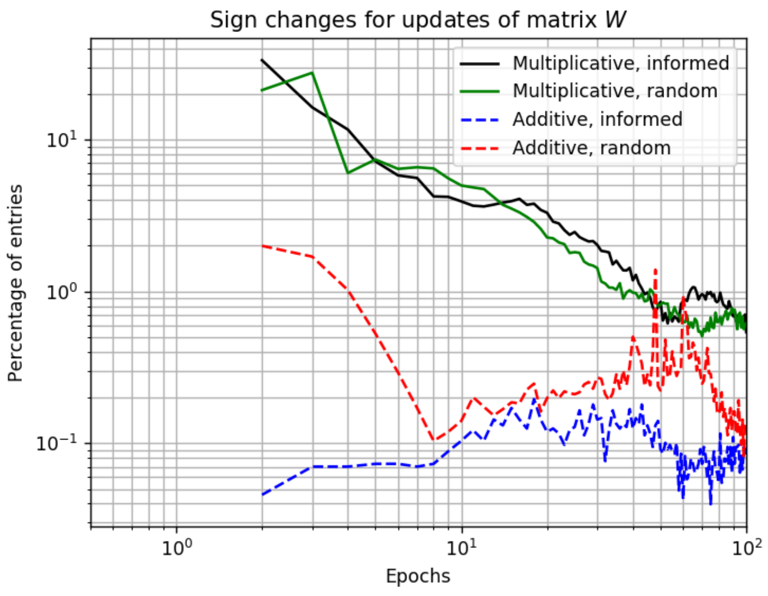


Figure 2.15: Sign changes for gradients w.r.t. matrix $W$ for four NMF approaches.

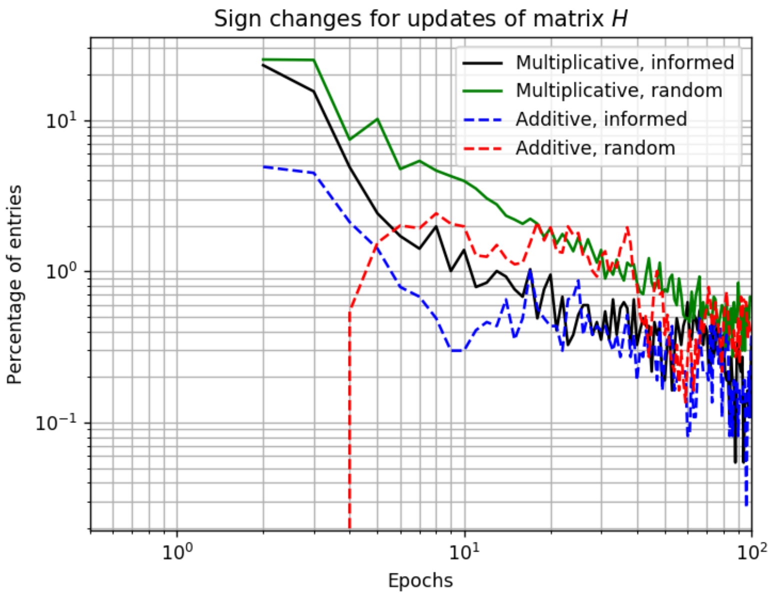


Figure 2.16: Sign changes for gradients w.r.t. matrix $H$ for four NMF approaches.

Another possibility to investigate the behavior of the update steps is to set a threshold $\alpha$ and count the number of entries, where

$$\left| W_{kr}^{(\ell+1)} - W_{kr}^{(\ell)} \right| > \alpha. \tag{2.30}$$

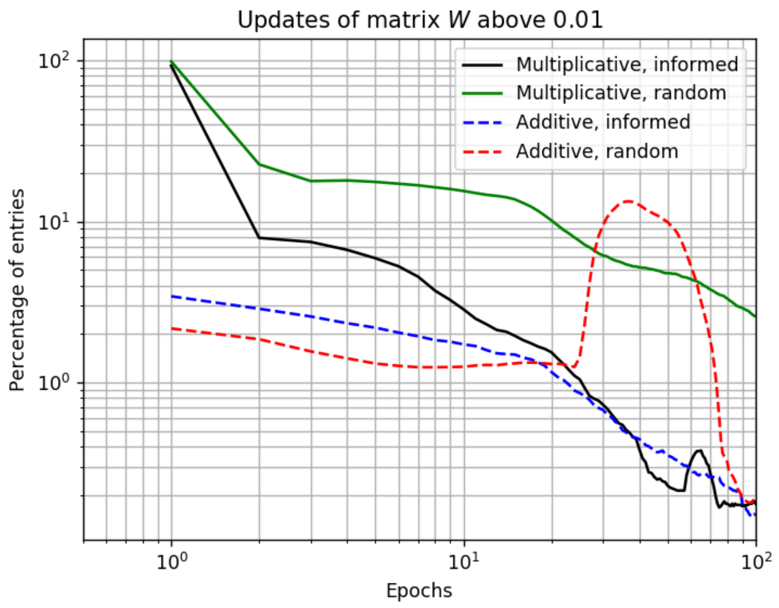


Figure 2.17: Updates of $W$ above 0.01 for four NMF approaches.

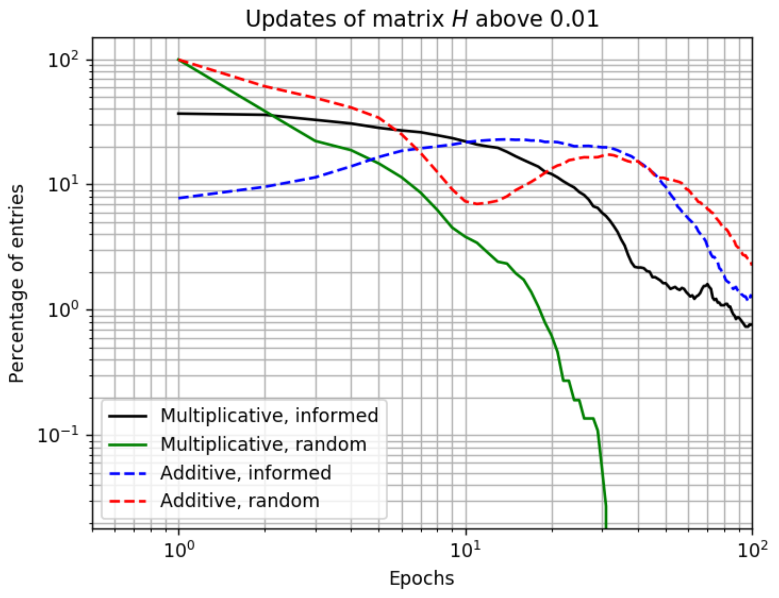


Figure 2.18: Updates of $H$ above 0.01 for four NMF approaches.

This measure is related to the Cauchy error in (2.26), as the Cauchy error directly measures the size of the updates, whereas the number of updates above a certain threshold is a more indirect measure.

We choose an exemplary threshold of $\alpha = 0.01$ and plot the according curves for $W$ and $H$ in Figures 2.17 and 2.18, respectively. We see that the multiplicative updates once again yield more monotonically decreasing curves than the additive update rules. Note that the order of

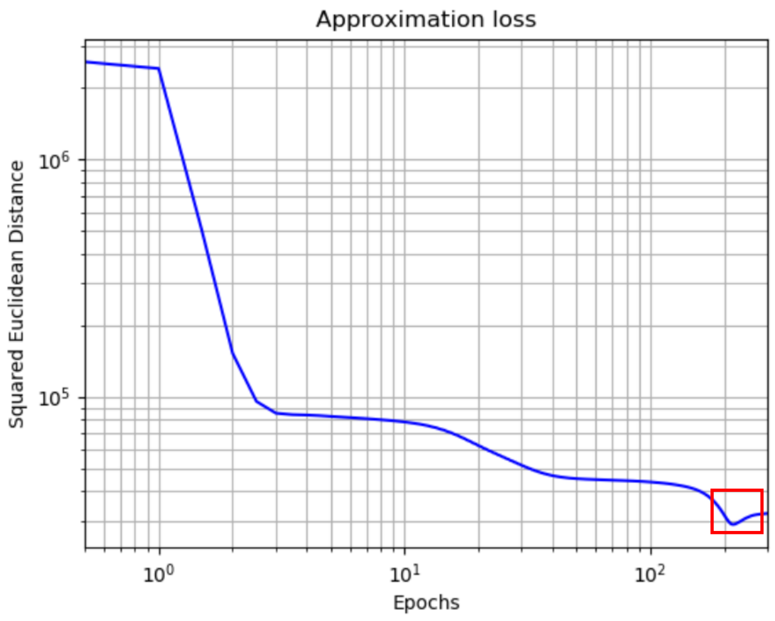


Figure 2.19: Loss curve for multiplicative updates and informed initialization, the red box indicates the part of the curve plotted in Figure 2.20.

magnitude of these curves, just as the Cauchy errors in Section 2.3.2, may vary due to the fact that no normalization was applied.

## 2.4 Rising Loss

When performing our NMF-related experiments, we encountered a problem concerning the loss function. As mentioned in Section 2.1.3, Lee and Seung prove in [18] that using the multiplicative update rules results in a non-increasing loss function (2.5). However, when using the multiplicative update rules in combination with musically informed initialization, we observe a rise of the loss function. This happens after 200 to 300 iterations. The loss curve for the running example is shown in Figure 2.19. The part of the curve where a rise of the loss value occurs is marked by a red box.

Figure 2.20 shows the rise of the loss function slightly after 200 iterations. Furthermore, the loss also rises within one iteration when updating only one of the matrices as shown in Figure 2.21. Both of these effects should theoretically not occur. However, we could observe it for various examples. This section explains what we expect to be the cause of the problem and how we reach this conclusion.

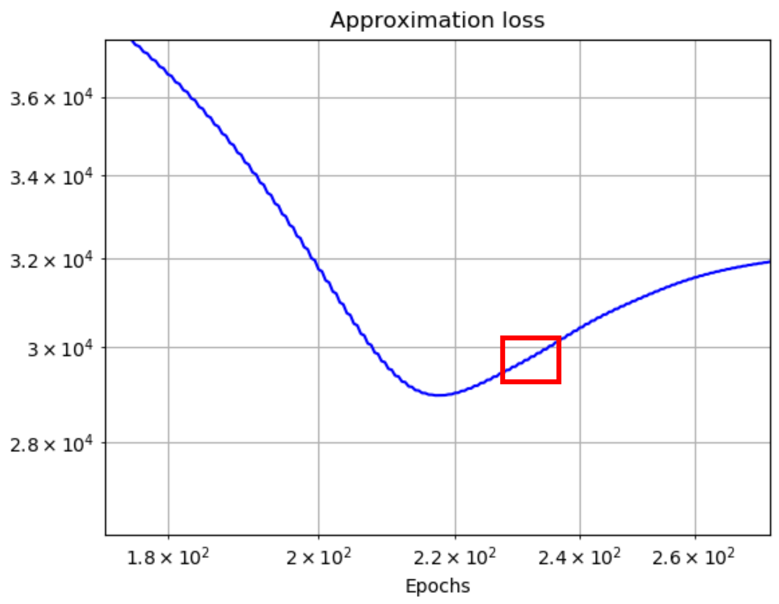


Figure 2.20: Rising loss for multiplicative updates and informed initialization (zoomed in), the red box indicates the part of the curve plotted in Figure 2.21.

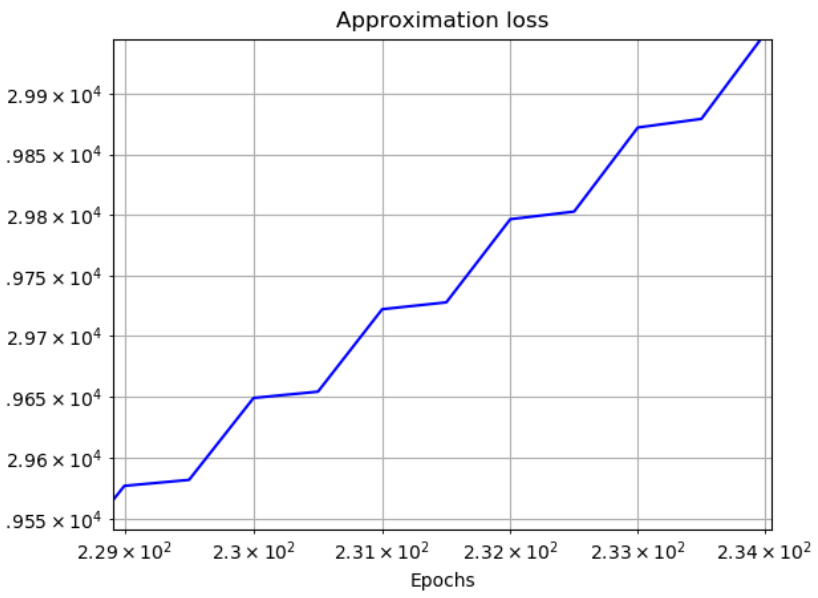


Figure 2.21: Rising loss for multiplicative updates and informed initialization (further zoomed in).

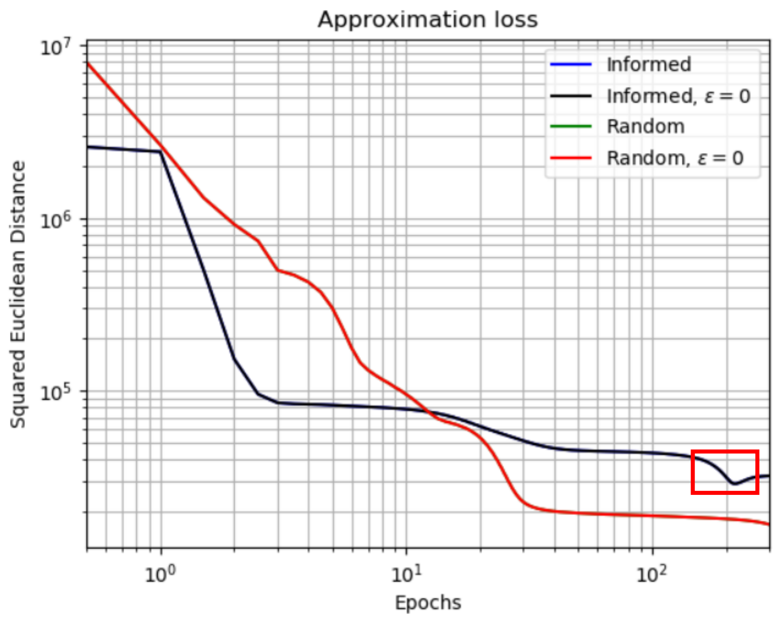


Figure 2.22: Loss curves for different initializations and values of $\varepsilon$, the red box indicates the part of the curves plotted in Figure 2.23. The curves for the same initialization lie exactly on top of each other.

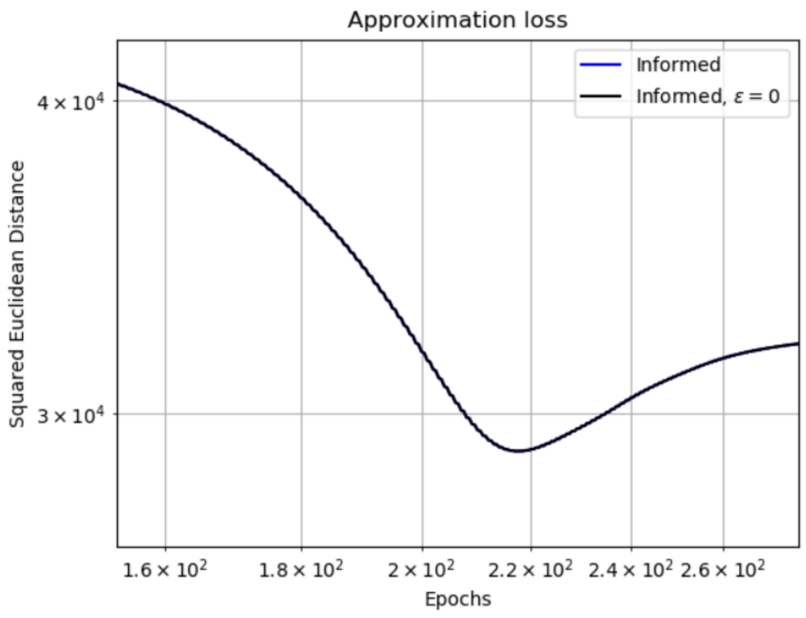


Figure 2.23: Rising loss for informed initialization and different values of $\varepsilon$. The curves lie exactly on top of each other.

| | **$W$ informed** | **$W$ random** |
|---|---|---|
| **$H$ informed** | Rise | Rise |
| **$H$ random** | Rise | No rise |

Table 2.2: Rising loss occurrences for different initializations of $W$ and $H$.

### 2.4.1 Numerical Instabilities

As the problem is that there occurs a difference between the theoretically proven and practically occurring behavior of the algorithm, our first intuition is that some numerical inaccuracies have to be the reason. However, investigating the numerators and denominators of the multiplicative update factors shows that the orders of magnitude of the values are in a range between $10^{-4}$ and $10^4$. Therefore, problems with under- or overflow can be ruled out.

Our next idea is to investigate the difference between our update equations and the ones defined in [18], which is the additional machine epsilon $\varepsilon$ in the denominator. Therefore, we compare the loss curves for our implementation and the alternative case, where we set $\varepsilon = 0$. By this, we risk a possible division by zero. However, the problem did not occur during our experiments. The resulting curves for informed and random initialization are shown in Figure 2.22. However, we can see that the machine epsilon does not make a difference as the curves for the same initializations lie exactly on top of each other. Therefore, the rise of the loss after 200 iterations, that we already saw in Figure 2.21, is still occurring for $\varepsilon = 0$ and shown in Figure 2.23.

### 2.4.2 Matrix Initializations

As the problem seems to not be caused by the implementation of the update rules, the remaining possible cause are the initializations of the matrices $W$ and $H$. Therefore, we experiment with various options for the initial values in order to find the cause.

First of all, we try to use random initializations for the different matrices. The results are shown in Table 2.2. We see that only when using random initialization for both $W$ and $H$ we can avoid the rising loss. Therefore, we can conclude that the musically informed initialization seems to cause the unexpected rise of the loss.
Our next step is to find the reason why the musically informed initialization causes the problem. There are two possible sources: the areas initialized with zero and the positive areas. The areas initialized by zero seem to be the cause as they are only produced by the informed initialization. Therefore, we investigate the behavior of the loss curve when replacing the zero entries in the initializations of $W$ and $H$ by positive values. To do so, we choose two options as a replacement for the zeros: random values (uniformly distributed in the range between zero and one) and a machine epsilon.

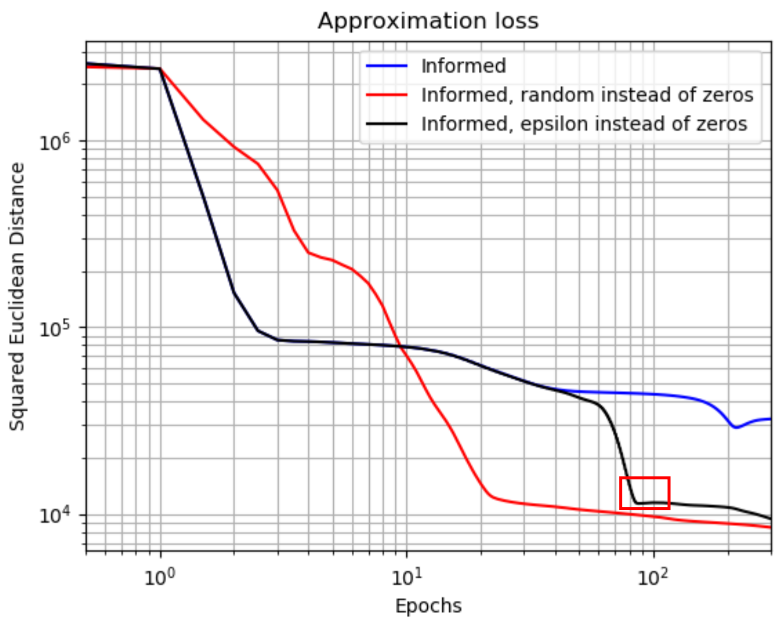


Figure 2.24: Loss curves for different informed initializations of $W$ and $H$, the red box indicates the part of the curves plotted in Figure 2.25.

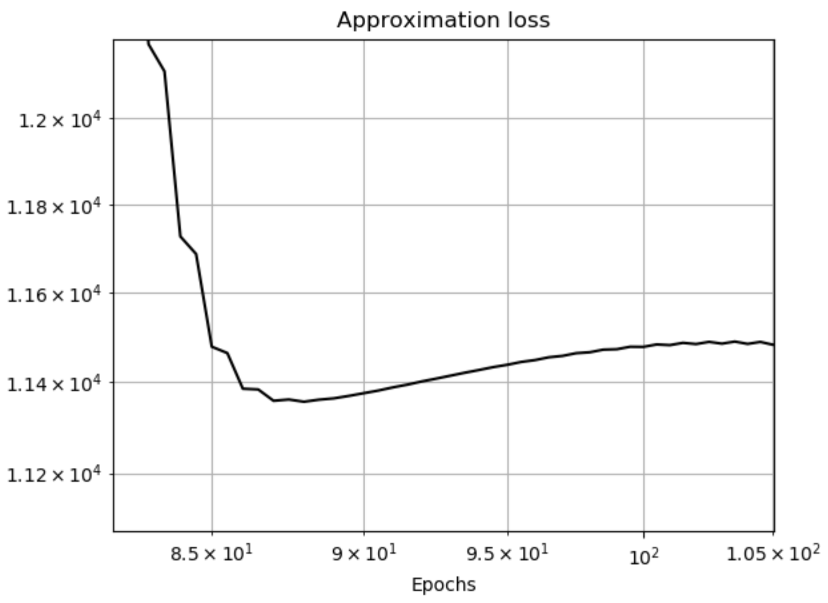


Figure 2.25: Rising loss for the replacement of zeros in the informed initialization by a machine epsilon.

| **Replacement** | Machine epsilon | Random values |
|---|---|---|
| **Result** | Rise | No rise |

Table 2.3: Rising loss occurrences for different replacements of zeros in the informed initialization.

Figure 2.24 shows the resulting loss curves for 300 iterations. The results are summarized in Table 2.3. We can see that replacing the values of zero by a machine epsilon does not solve the problem. In the beginning the loss curve behaves exactly like the curve for the original informed initialization. After almost 100 iterations the algorithm increases some values close to zero to higher positive numbers. Therefore, the loss curves start to differ after that point. However, the loss curve for the replacement of zeros by a machine epsilon still experiences a rise of the loss at around 100 iterations, as shown in Figure 2.25.
Replacing the zero entries by random values corresponds to no musically informed constraints and avoids the rising loss problem completely. Therefore, we can conclude that the unexpected rise of the loss function is caused by values in the initialization which are exactly zero or close to zero. However, we can yet only give practical but no theoretical proof for this.

# Chapter 3

# Nonnegative Autoencoders (NAEs)

Neural networks and other deep learning-based methods gained more and more importance in signal processing and related fields throughout the recent years. This is also true for the area of music information retrieval, where state-of-the-art solutions for many tasks, such as music tagging [4], piano transcription [16] or chord recognition [22], are based on neural networks [3]. This chapter covers the core topic of this thesis, which is modeling the behavior of the NMF algorithm explained in Chapter 2 as a neural network. Furthermore, we give extensions to the previous work on this topic by Suárez in [30].
This chapter is organized as follows: In Section 3.1 we cover basics about neural networks and explain the network structure used in this thesis. We subsequently discuss the application of these networks to the music decomposition scenario in Section 3.2. We show our own contributions to the topic in the remaining two sections. In Section 3.3 we derive multiplicative update rules for the used Nonnegative Autoencoder (NAE) network and discuss the convergence of these new update rules. Finally, we build upon these results in Section 3.4 and give ideas for the initialization of the encoder weights.

## 3.1 Theoretic Foundations

This section covers the basic theory behind the networks we use in this thesis. We explain the network components and how they are connected to NMF. Afterwards, we explain the topology of the used network and discuss the training of the network.

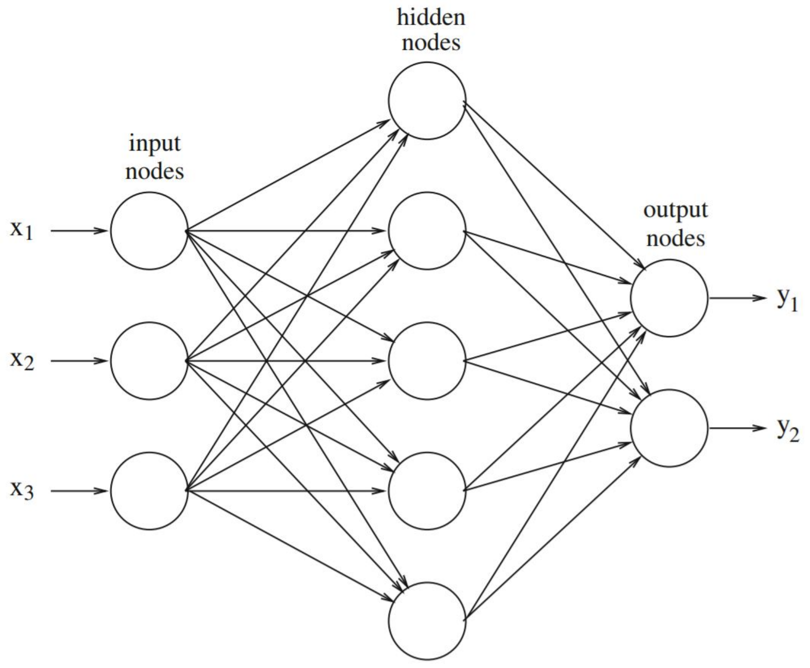


Figure 3.1: Structure of a FC network (from [2]).

### 3.1.1 Fully-Connected Neural Networks

One of the most basic layers applied in neural networks are fully-connected (FC) layers. They convert a $K_1$-dimensional input vector $x \in \mathbb{R}^{K_1}$ to a $K_2$-dimensional output vector $y \in \mathbb{R}^{K_2}$. To do so, they use a linear transformation defined by a weight matrix $W \in \mathbb{R}^{K_2 \times K_1}$ and a bias vector $b \in \mathbb{R}^{K_2}$. The mathematical definition is given by

$$y = Wx + b. \tag{3.1}$$

This is often used to compress the input by choosing $K_2 < K_1$. Therefore, we would in this case intuitively want this type of layer to learn how to extract the essence of the input vector. By stacking several of these layers together, we can build FC networks. The principle of FC networks is also illustrated in Figure 3.1. We see that for each FC layer, there is a direct connection between each pair of entries of input and output vector. FC networks typically consist of one input and one output layer as well as several hidden layers in between them.

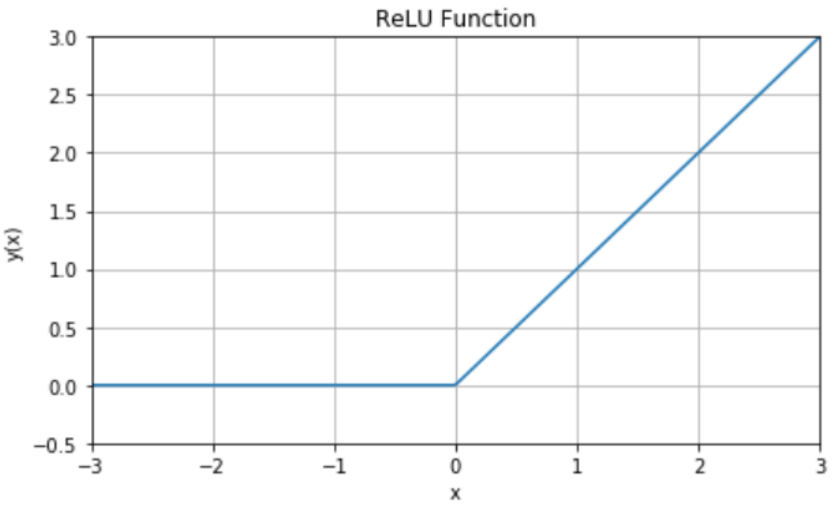


Figure 3.2: ReLU activation function.

### 3.1.2 Activation Functions

FC layers are typically used in combination with activation functions by applying them pointwise to the output of Equation (3.1). While the matrix multiplication is a linear operation, activation functions are normally chosen to be nonlinear as the nonlinearity of neural networks is one of the key factors for the performance of the network [17]. Therefore, the choice of activation is important. Some common choices are the sigmoid, the TanH or the ReLU activation function [27]. In this thesis, we choose the ReLU activation function, which is defined by:

$$\text{ReLU}(x) = \begin{cases} x & \text{if } x \geq 0 \\ 0 & \text{otherwise} \end{cases} \tag{3.2}$$

This function is also plotted in Figure 3.2. It can be seen that the nonlinearity for this layer is caused by the elimination of negative values by replacing them with zero. Nonnegative inputs however are unaffected by the ReLU function. In this thesis, we denote activation functions by $g(x)$.

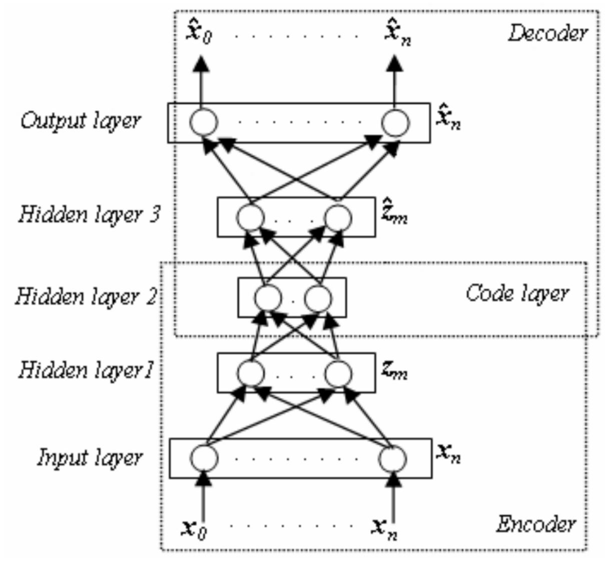


Figure 3.3: Exemplary autoencoder structure (from [31]).

### 3.1.3 Autoencoders

Autoencoders are a special kind of neural network. Unlike networks for classification problems, the goal of autoencoders is not to output an estimation of class probabilities but to reconstruct the input of the network. To do so, the network usually consists of two subnetworks, as illustrated in Figure 3.3. The first one is the encoder, which typically tries to compress the input to a lower-dimensional representation. The second one is the decoder, which is often a mirrored version of the encoder and tries to reconstruct the original input from the compressed representation extracted by the encoder.

### 3.1.4 Nonnegativity Constraints

The idea to use neural networks for nonnegative audio models was introduced by Samargdis et al. in [28]. They introduce nonnegativity constraints into the network by using the ReLU function (3.2) as the activation function for the network. In our case, this would mean that the activation functions of encoder and decoder would be chosen to be $g_{\mathcal{E}}(x) = g_{\mathcal{D}}(x) = \text{ReLU}(x)$. By doing so, we ensure that the intermediate representations and the output of the network are nonnegative, because the ReLU function eliminates negative values by replacing them by zero. Therefore, the weights and the input of the network could also contain negative entries as the ReLU activation functions would still avoid negative outputs of the FC layers. This is the reason why the network is called nonnegative autoencoder.
As our goal is to model the NMF algorithm with the autoencoder network, we constrain all inputs and weights of the network in this thesis to be nonnegative as well. The input is ensured

to be nonnegative as we use magnitude spectrogram frames for the network input. The weights of the network need to be constrained to be nonnegative after each update step. This can be done by applying the ReLU function to them after calculating the updated values:

$$\left(W_{\mathcal{D}/\mathcal{E}}^{(\ell+1)}\right)^{+} = \mathrm{ReLU}\left(W_{\mathcal{D}/\mathcal{E}}^{(\ell+1)}\right) \tag{3.3}$$

This can also be seen as a projected gradient method [20]. The embedding representation and output of the network would already be nonnegative due to the used ReLU activation functions as explained above. However, when constraining the network weights $W_{\mathcal{E}}$ and $W_{\mathcal{D}}$ to be nonnegative as well and only using nonnegative inputs, we do not need to apply the ReLU function anymore. As the results of the matrix products will already be nonnegative themselves, the application of the ReLU function would not have any effect. Therefore, we can omit the activation function and set $g_{\mathcal{E}}(x) = g_{\mathcal{D}}(x) = x$ instead for the remainder of this thesis.

### 3.1.5 Structured Dropout

In order to be able to further control and constrain the embedding representation of autoencoders, Ewert et al. introduced the structured dropout layer in [10]. Dropout layers are typically used to regularize networks in order to avoid overfitting [29]. This is done by randomly switching off neurons during the training process. Structured dropout however does not randomly switch off neurons but instead uses prior knowledge to selectively eliminate undesired activations. We do this by applying a binary mask to the embedding space vector.
In our scenario, we apply a structured dropout layer on the activation vector estimated by the encoder:

$$h' = m_h \odot h. \tag{3.4}$$

Vectors $h \in \mathbb{R}^R_{\geq 0}$ and $h' \in \mathbb{R}^R_{\geq 0}$ denote the output of the encoder and the input for the decoder, respectively. The binary masking vector $m_h \in \{0,1\}^R$ is created using annotation information, similar to the activation constraints in NMF. To be more specific, the mask $m_h$ is the column of the initialization matrix $H^{(0)}$ from NMF which corresponds to the current time frame.

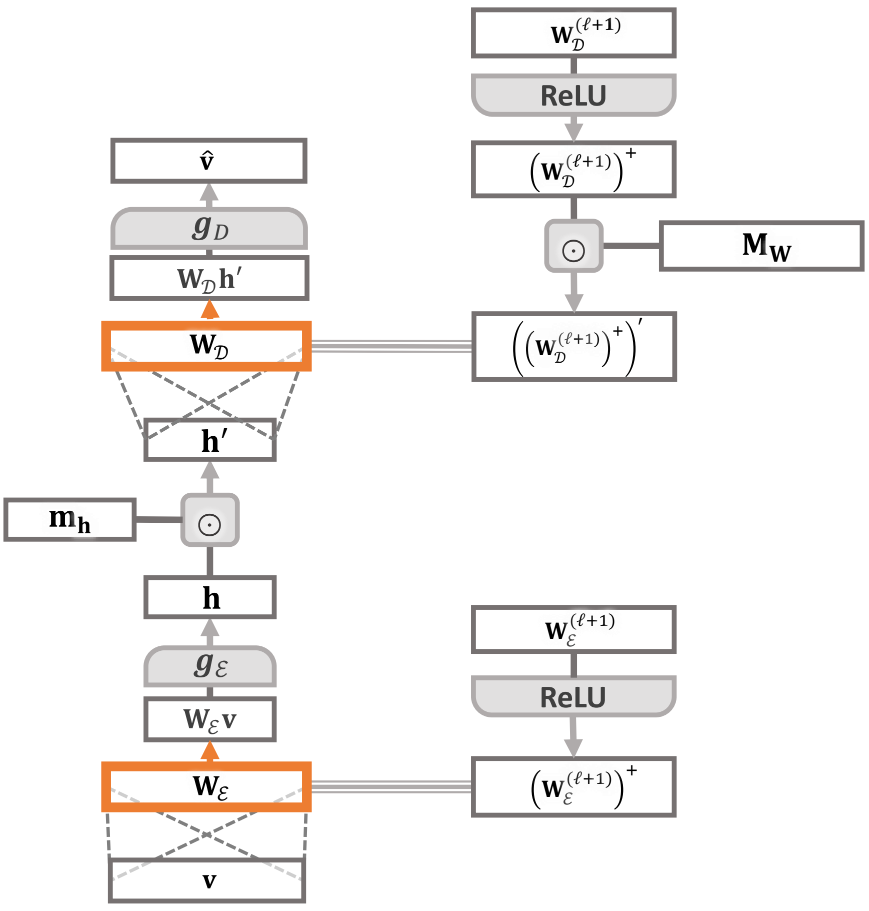

$\hat{\mathbf{v}}$
$g_{\mathcal{D}}$
$\mathbf{W}_{\mathcal{D}}\mathbf{h}'$
$\mathbf{W}_{\mathcal{D}}$
$\mathbf{h}'$
$\mathbf{m}_{\mathbf{h}}$
$\odot$
$\mathbf{h}$
$g_{\mathcal{E}}$
$\mathbf{W}_{\mathcal{E}}\mathbf{v}$
$\mathbf{W}_{\mathcal{E}}$
$\mathbf{v}$
$\mathbf{W}_{\mathcal{D}}^{(\ell+1)}$
ReLU
$\left(\mathbf{W}_{\mathcal{D}}^{(\ell+1)}\right)^{+}$
$\odot$
$\mathbf{M}_{\mathbf{W}}$
$\left(\left(\mathbf{W}_{\mathcal{D}}^{(\ell+1)}\right)^{+}\right)'$
$\mathbf{W}_{\mathcal{E}}^{(\ell+1)}$
ReLU
$\left(\mathbf{W}_{\mathcal{E}}^{(\ell+1)}\right)^{+}$

Figure 3.4: Structure of the NAE network. Vectors and matrices are denoted by boxes with lower and upper case letters, respectively. Weights are denoted by orange boxes and activation functions by boxes with two round corners. FC layers are denoted by dotted lines and the order of operations by arrows. Pointwise multiplications are denoted by the $\odot$ symbol.

### 3.1.6 NAE Network Structure

In this section, we explain the structure of the used NAE network. We show the different computations in matrix notation and also discuss the imposing of constraints on the weight matrices. An overview of the network topology can be seen in Figure 3.4. It is an autoencoder network, where the input $v \in \mathbb{R}^{K}_{\geq 0}$ and output $\hat{v} \in \mathbb{R}^{K}_{\geq 0}$ are both magnitude spectrogram frames consisting of $K$ frequency bins. The encoder and decoder both consist of a combination of a FC layer and an activation function. Between the encoder and the decoder, we apply a structured dropout layer to the intermediate representation $h \in \mathbb{R}^{R}_{\geq 0}$, where $R$ is the dimensionality of the compressed representation. Furthermore, we impose nonnegativity and musically informed constraints on the weights of the FC layers after updating them. We will discuss the different

components in more detail in the following.
For the remainder of this thesis, we will give the computations regarding the network in matrix notation. This means that we do not input one spectrogram frame $v$ at a time but instead stack various of them together which yields the complete spectrogram $V \in \mathbb{R}_{\geq 0}^{K \times N}$. In the encoder layer we then compress the $K$-dimensional frequency representation for each frame to a $R$-dimensional one by using a FC layer. We do not use bias vectors for the FC layers of this thesis in order to make our model more comparable to NMF. As mentioned in Section 3.1.4, we set $g_{\mathcal{E}}(x) = x$. Therefore, the output of the encoder is given by

$$H = W_{\mathcal{E}} V, \tag{3.5}$$

where $W_{\mathcal{E}} \in \mathbb{R}_{\geq 0}^{R \times K}$ is the weight matrix of the encoder layer and $H \in \mathbb{R}_{\geq 0}^{R \times N}$ the embedding representation computed by the encoder. Afterwards, we apply a structured dropout layer to the output of the encoder in order to eliminate undesired activations. The input for the decoder is therefore given by

$$H' = H \odot M_H, \tag{3.6}$$

where $M_H \in \{0, 1\}^{R \times N}$ is the binary mask for the dropout layer. As explained in Section 3.1.5, the column for one frame in this mask is simply given by the corresponding frame in the activation matrix initialization $H^{(0)}$ in NMF. Therefore, the complete mask for the structured dropout layer is given by $M_H = H^{(0)}$. By pointwise application of this mask, we compute the input $H'$ for the decoder.
In the decoder, we once again use no bias vector and set $g_{\mathcal{D}}(x) = x$. Therefore, the output of the decoder is given by

$$\hat{V} = W_{\mathcal{D}} H', \tag{3.7}$$

where $W_{\mathcal{D}} \in \mathbb{R}_{\geq 0}^{K \times R}$ is the weight matrix of the decoder layer and $\hat{V} \in \mathbb{R}_{\geq 0}^{K \times N}$ the approximation of the input spectrogram. When looking at the computation of the approximation $\hat{V}$, we see that it is very similar to the NMF-based approximation in (2.3). The weight matrix $W_{\mathcal{D}}$ serves as the template matrix and the embedding representation $H'$ as the activation matrix. Therefore, the only bigger difference between the NAE and NMF is that we do not directly learn the activations but instead train the encoder to extract them from the spectrogram frames in $V$.
This typically results in a smaller number of parameters for the NAE compared to NMF. The decoder of the NAE and the template matrix in NMF are both of size $K \times R$. The encoder matrix however is of size $R \times K$ opposed to the size $R \times N$ of the activation matrix in NMF. For

longer examples, which are typically of size $N > K$, this means that the NAE has less trainable parameters because the number of parameters does not directly depend on the number of time frames $N$. This is the case because we do not train individual parameters for each time frame. A smaller number of parameters for the NAE means on the one hand less parameters to train but on the other hand also less complexity of the model, typically resulting in a higher final loss value.
As discussed in Section 3.1.4, we need to constrain the weights of the network to be nonnegative after each update step. This is done by

$$\left(W_{\mathcal{D}/\mathcal{E}}^{(\ell+1)}\right)^{+} = \mathrm{ReLU}\left(W_{\mathcal{D}/\mathcal{E}}^{(\ell+1)}\right). \tag{3.8}$$

Furthermore, we also include a possibility to impose musically informed constraints on the weights similar to the constraints for musically informed NMF. Our goal is to keep entries that were initialized by zero to stay at this value. This is done similarly to the application of the structured dropout layer:

$$\left(\left(W_{\mathcal{D}}^{(\ell+1)}\right)^{+}\right)' = \left(W_{\mathcal{D}}^{(\ell+1)}\right)^{+} \odot M_W \tag{3.9}$$

As we see, we apply a binary mask $M_W \in \{0,1\}^{K\times R}$ after imposing the nonnegativity constraints on the network weights. We do this after each update step by pointwise multiplication. The mask $M_W$ only contains a value of one for those entries, that were initialized by positive values.

### 3.1.7 Gradient Descent

In order to train the weights of our network, we again use the gradient descent algorithm with the update rule

$$W^{(\ell+1)} = W^{(\ell)} - \gamma \cdot \frac{\partial \mathcal{L}}{\partial W}. \tag{3.10}$$

As a loss function, we again use the squared Euclidean distance introduced in Chapter 2:

$$\begin{aligned}
\mathcal{L}(W_{\mathcal{E}}, W_{\mathcal{D}}) &= \|V - \hat{V}\|^2 = \sum_{k=1}^{K}\sum_{n=1}^{N}(V_{kn} - \hat{V}_{kn})^2 && (3.11)\\
&= \sum_{k=1}^{K}\sum_{n=1}^{N}\left(V_{kn} - \sum_{r=1}^{R} W_{\mathcal{D},kr}H'_{rn}\right)^2 && (3.12)
\end{aligned}$$

From this, we see that when comparing the formulation of the loss function to the one for NMF in (2.9), matrix $W_{\mathcal{D}}$ takes the role of $W$ in NMF and $H'$ in the NAE structure takes the role of $H$ in NMF. Therefore, we can reuse the update rule from Equation (2.17) as the update rule for the decoder by replacing the names of the matrices accordingly:

$$W_{\mathcal{D},kr}^{(\ell+1)} = W_{\mathcal{D},kr}^{(\ell)} - \gamma_{kr}^{(\ell)} \cdot \left( \left( W_{\mathcal{D}}^{(\ell)} H' H'^{\top} \right)_{kr} - \left( V H'^{\top} \right)_{kr} \right) \tag{3.13}$$

In order to calculate the gradient of the loss w.r.t. the encoder weights, we need to rewrite the loss function in a more detailed way:

$$\begin{aligned}
\mathcal{L}(W_{\mathcal{E}}, W_{\mathcal{D}}) &= \sum_{k=1}^{K} \sum_{n=1}^{N} \left( V_{kn} - \sum_{r=1}^{R} W_{\mathcal{D},kr} H'_{rn} \right)^2 && (3.14) \\
&= \sum_{k=1}^{K} \sum_{n=1}^{N} \left( V_{kn} - \sum_{r=1}^{R} W_{\mathcal{D},kr} M_{H,rn} H_{rn} \right)^2 && (3.15) \\
&= \sum_{k=1}^{K} \sum_{n=1}^{N} \left( V_{kn} - \sum_{r=1}^{R} W_{\mathcal{D},kr} M_{H,rn} \sum_{m=1}^{K} W_{\mathcal{E},rm} V_{mn} \right)^2 && (3.16)
\end{aligned}$$

From this formulation with an explicit dependence on the entries of $W_{\mathcal{E}}$, we can compute the derivative w.r.t. one matrix entry $W_{\mathcal{E},\rho\mu}$.
We first start with a simple step for the derivation of the gradient:

$$\begin{aligned}
\frac{\partial \mathcal{L}}{\partial W_{\mathcal{E},\rho\mu}} &= \frac{\partial \sum_{k=1}^{K} \sum_{n=1}^{N} (V_{kn} - \hat{V}_{kn})^2}{\partial W_{\mathcal{E},\rho\mu}} && (3.17) \\
&= \sum_{k=1}^{K} \sum_{n=1}^{N} 2(\hat{V}_{kn} - V_{kn}) \frac{\partial \hat{V}_{kn}}{\partial W_{\mathcal{E},\rho\mu}} && (3.18)
\end{aligned}$$

Therefore, we need to calculate the derivative of one entry $\hat{V}_{kn}$ in the output matrix w.r.t. one entry in the encoder weight matrix:

$$\begin{aligned}
\frac{\partial \hat{V}_{kn}}{\partial W_{\mathcal{E},\rho\mu}} &= \frac{\partial \sum_{r=1}^{R} W_{\mathcal{D},kr} M_{H,rn} \sum_{m=1}^{K} W_{\mathcal{E},rm} V_{mn}}{\partial W_{\mathcal{E},\rho\mu}} && (3.19) \\
&= W_{\mathcal{D},k\rho} M_{H,\rho n} \frac{\partial \sum_{m=1}^{K} W_{\mathcal{E},\rho m} V_{mn}}{\partial W_{\mathcal{E},\rho\mu}} && (3.20) \\
&= W_{\mathcal{D},k\rho} M_{H,\rho n} V_{\mu n} && (3.21)
\end{aligned}$$

In (3.20), we use the fact that only the summand for $r = \rho$ depends on $W_{\mathcal{E},\rho\mu}$. For (3.21) we do the same for $m = \mu$.
Inserting the result from (3.21) into (3.18), we can do our final derivations:

$$\begin{aligned}
\frac{\partial \mathcal{L}}{\partial W_{\mathcal{E},\rho\mu}} &= \sum_{k=1}^{K}\sum_{n=1}^{N} 2(\hat{V}_{kn} - V_{kn}) W_{\mathcal{D},k\rho} M_{H,\rho n} V_{\mu n} && (3.22)\\
&= 2\sum_{k=1}^{K}\sum_{n=1}^{N} \hat{V}_{kn} W_{\mathcal{D},k\rho} M_{H,\rho n} V_{\mu n} - 2\sum_{k=1}^{K}\sum_{n=1}^{N} V_{kn} W_{\mathcal{D},k\rho} M_{H,\rho n} V_{\mu n} && (3.23)\\
&= 2\sum_{n=1}^{N} M_{H,\rho n} V_{\mu n} \sum_{k=1}^{K} W_{\mathcal{D},\rho k}^{\top} \hat{V}_{kn} - 2\sum_{n=1}^{N} M_{H,\rho n} V_{\mu n} \sum_{k=1}^{K} W_{\mathcal{D},\rho k}^{\top} V_{kn} && (3.24)\\
&= 2\sum_{n=1}^{N} M_{H,\rho n} V_{n\mu}^{\top} \left(W_{\mathcal{D}}^{\top}\hat{V}\right)_{\rho n} - 2\sum_{n=1}^{N} M_{H,\rho n} V_{n\mu}^{\top} \left(W_{\mathcal{D}}^{\top} V\right)_{\rho n} && (3.25)\\
&= 2\sum_{n=1}^{N} \left(\left(W_{\mathcal{D}}^{\top}\hat{V}\right) \odot M_H\right)_{\rho n} V_{n\mu}^{\top} - 2\sum_{n=1}^{N} \left(\left(W_{\mathcal{D}}^{\top} V\right) \odot M_H\right)_{\rho n} V_{n\mu}^{\top} && (3.26)\\
&= 2\left(\left(\left(W_{\mathcal{D}}^{\top}\hat{V}\right) \odot M_H\right) V^{\top}\right)_{\rho\mu} - 2\left(\left(\left(W_{\mathcal{D}}^{\top} V\right) \odot M_H\right) V^{\top}\right)_{\rho\mu} && (3.27)
\end{aligned}$$

In (3.24) we rearrange the sums and use the transpose $W_{\mathcal{D}}^{\top}$ of the decoder matrix. Afterwards, we use the transpose of matrix $V$ and write parts of the equation in matrix notation in order to derive (3.25). Finally, we rewrite further parts of the equation in matrix notation and arrive at (3.26) and subsequently at (3.27). Note that we omit the factor of 2 in the following, similar as we did in Section 2.1.2.
The gradient descent update rule for the encoder weights is therefore given by

$$\begin{aligned}
W_{\mathcal{E},rk}^{(\ell+1)} = W_{\mathcal{E},rk}^{(\ell)} - \gamma_{rk}^{(\ell)} \cdot \Bigg( &\left(\left(\left(W_{\mathcal{D}}^{\top} W_{\mathcal{D}} H'^{(\ell)}\right) \odot M_H\right) V^{\top}\right)_{rk} \\
&- \left(\left(\left(W_{\mathcal{D}}^{\top} V\right) \odot M_H\right) V^{\top}\right)_{rk} \Bigg).
\end{aligned} \tag{3.28}$$

Note that matrix $H'$ is dependent on the iteration as it depends on $W_{\mathcal{E}}$. Furthermore, after the application of the update rules (3.13) and (3.28) we have to project negative entries back to the nonnegative space by setting them to zero. As mentioned in Section 3.1.6, this is done after each update by application of the ReLU function:

$$\left(W_{\mathcal{D}/\mathcal{E}}^{(\ell+1)}\right)^{+} = \operatorname{ReLU}\left(W_{\mathcal{D}/\mathcal{E}}^{(\ell+1)}\right) \tag{3.29}$$

### 3.1.8 Optimizers

In deep learning, there are different popular choices for optimizers, which update the network weights. The simplest choice is the stochastic gradient descent (SGD) optimizer, which directly applies the gradient descent rules from Section 3.1.7. In the following, we cover this optimizer and two other popular choices, namely the RMSprop and the Adam optimizer [21]. Furthermore, we compare the convergence behavior of the loss curves for the different optimizers.

#### 3.1.8.1 Stochastic Gradient Descent (SGD)

Assume we have calculated the gradient of the loss w.r.t. some weights $W^{(\ell)} \in \mathbb{R}^{A\times B}$ we want to update. $A$ and $B$ denote the size of the weight matrix. We denote the respective gradient by $g^{(\ell)} \in \mathbb{R}^{A\times B}$. Then, the simple SGD update rule is given by

$$W^{(\ell+1)} = W^{(\ell)} - \gamma \cdot g^{(\ell)}, \tag{3.30}$$

where $\gamma \in \mathbb{R}_{\geq 0}$ is a constant learning rate we have to choose. A popular choice to start with is $\gamma = 0.01$. This optimizer is easy to implement but also has certain drawbacks due to its simplicity. As the learning rate is not adaptable in the basic SGD optimizer, the model can not adapt to changes of the required learning rate. For example, in the beginning of the training process, we need a rather high learning rate but towards the end we may want to decrease it in order to achieve a better refinement of the minimum. Furthermore, the fitting learning rate for different entries of the weight matrix may differ.

#### 3.1.8.2 RMSprop

One optimizer attempting to solve the problems we just mentioned is the RMSprop optimizer. It introduces an individual scaling factor for each matrix entry, that adapts over time. We start with a recursive average over the squared gradient:

$$r^{(\ell)} = \rho r^{(\ell-1)} + (1-\rho) g^{(\ell)} \odot g^{(\ell)} \tag{3.31}$$

The factor $\rho \in [0, 1]$ controls the influence of new gradients on the average. Using this, we can scale the update steps inversely to the recursive average of past gradients. We do this to solve the problem of the SGD optimizer concerning the different sizes of gradients. If the gradient w.r.t. one entry was very low in the recent past, we scale it by a higher value in order to still achieve big enough update steps and vice versa for higher gradients. This scaling results in the update rule

$$W^{(\ell+1)} = W^{(\ell)} - \frac{\gamma}{\sqrt{r^{(\ell)}} + \varepsilon} \odot g^{(\ell)}, \tag{3.32}$$

where we add a machine epsilon $\varepsilon$ in the denominator in order to avoid division by zero. The parameter $\gamma$ again defines the learning rate. Typical parameter choices are $\rho = 0.9$ and $\gamma = 0.001$.

#### 3.1.8.3 Adam

Another very popular choice is the Adam optimizer. Just like the RMSprop optimizer, it uses a recursive average to scale the update steps:

$$r^{(\ell)} = \rho r^{(\ell-1)} + (1-\rho) g^{(\ell)} \odot g^{(\ell)} \tag{3.33}$$

Additionally, it uses momentum in order to make it harder for the update steps to quickly change direction. The intuition behind this is that we try to use a similar direction as in previous epochs for the update of the weights. This is inspired by physics where a moving mass tries to keep on moving in the same direction. This is done by using a recursive average over past gradients:

$$v^{(\ell)} = \mu v^{(\ell-1)} + (1-\mu) g^{(\ell)}, \tag{3.34}$$

where $\mu \in [0, 1]$ is again a parameter to control the influence of new gradients. Furthermore, Adam uses bias correction on $r$ and $v$. By doing this, the estimates of the gradient and the squared gradient become unbiased, which means their estimation error is zero in the mean. This correcting is done according to

$$\hat{r}^{(\ell)} = \frac{r^{(\ell)}}{1-\rho^{\ell}} \tag{3.35}$$

and

$$\hat{v}^{(\ell)} = \frac{v^{(\ell)}}{1-\mu^{\ell}}. \tag{3.36}$$

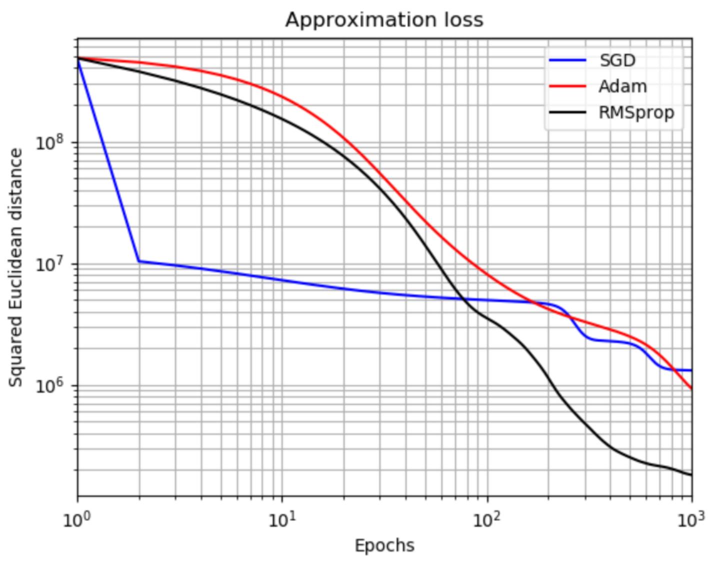


Figure 3.5: Loss curves for three different optimizers.

Using these corrected averages, we update our weights according to

$$W^{(\ell+1)} = W^{(\ell)} - \gamma \frac{\hat{v}^{(\ell)}}{\sqrt{\hat{r}^{(\ell)}} + \varepsilon}. \tag{3.37}$$

Once again, we have to set a learning rate $\gamma$ and use $\varepsilon$ to avoid division by zero. Popular parameter choices are $\rho = 0.999$, $\mu = 0.9$ and $\gamma = 0.001$.

### 3.1.8.4 Comparison

To conclude this section, we want to compare the convergence of the loss curve for the optimizers we just introduced. We do this by applying the NAE to the running example. We initialize all weights randomly and use no musically motivated constraints. For the parameter choices for the different optimizers, we use the settings suggested in the respective sections and run the training for 1000 epochs each.
The resulting curves are plotted in Figure 3.5. We see that the SGD optimizer shows a smaller loss value in the beginning due to a steep drop of the loss during the first epochs. However, the loss stagnates afterwards and only decreases slowly. After around 100 epochs, the other two optimizers catch up to the SGD optimizer and show a faster convergence. The Adam optimizer ends up at a loss value slightly below the SGD optimizer. The RMSprop optimizer however achieves a loss significantly lower than the other two optimizers.

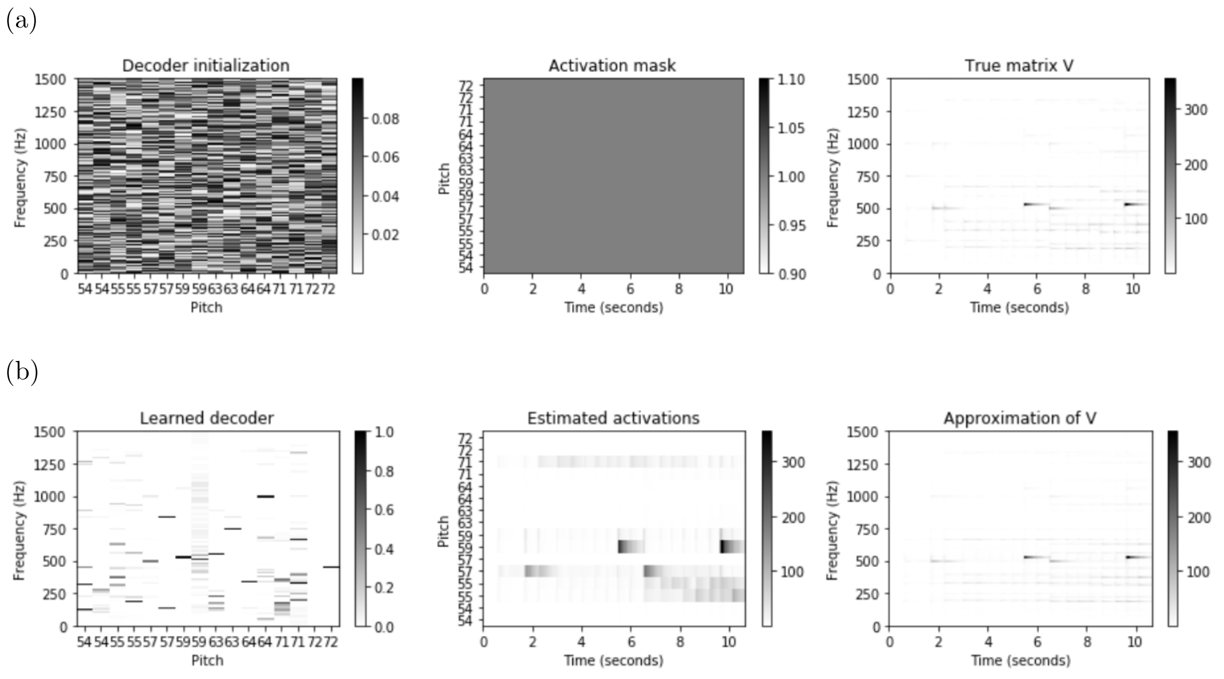


Figure 3.6: NAE-based decomposition using random initialization: (a) Decoder initialization, activation mask and input matrix $V$, (b) Resulting approximation.

## 3.2 Application to Music Decomposition

In this section, we talk about how we can use the methods introduced in Section 3.1 for our scenario of music decomposition. We introduce musically informed constraints on the decoder templates and the embedding activations similar to the ones in NMF. Afterwards, we show how we can use the NAE framework to decompose piano recordings into separated signals for the left and the right hand.

Analogously to NMF, we start by showing the resulting decomposition into embedding activations and decoder templates in Figure 3.6. We choose the dimensionality of the embedding space $R$ as two times the number of occurring pitches, just as we did for the rank of NMF when introducing onset models in Section 2.2.1.3. We train for 1000 epochs and use an SGD optimizer with a learning rate of 0.01 for the encoder and 0.1 for the decoder. We choose different learning rates for the encoder and the decoder as the practical experience from our experiments showed that the decoder needs a higher learning rate in order to achieve the same degree of convergence as the encoder after the same number of epochs. This may be caused by different orders of magnitude of the gradient. However, we have no theoretical proof for this phenomenon. We enforce the nonnegativity constraints by setting negative matrix entries to zero. When looking at the trained decoder in Figure 3.6b we see that the templates given by the columns of the matrix look rather

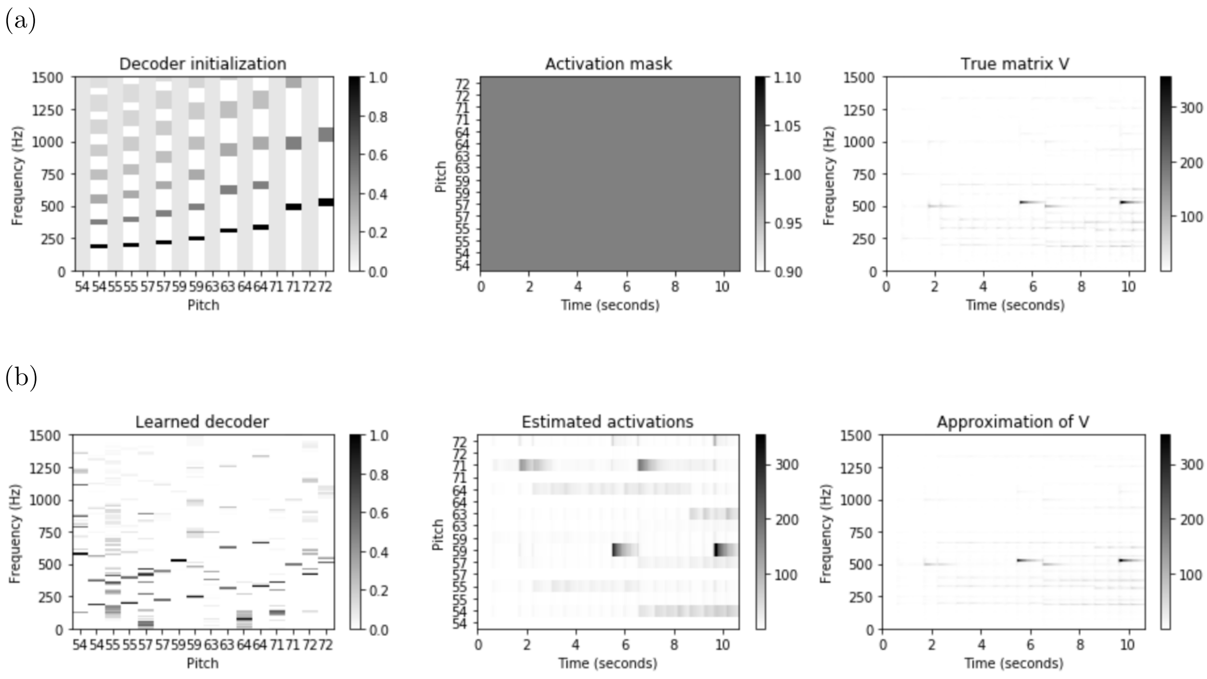


Figure 3.7: NAE-based decomposition using decoder constraints: (a) Decoder initialization, activation mask and input matrix $V$, (b) Resulting approximation.

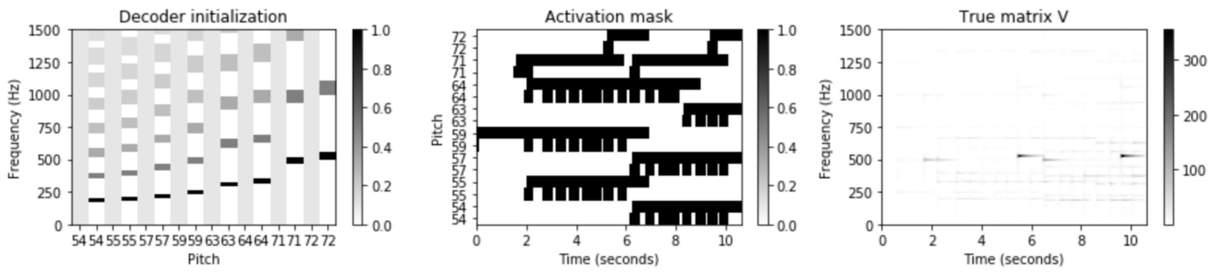
random and are not corresponding to any specific pitches. Therefore, we have a motivation to impose musically motivated constraints on our network to make the decoder templates and the embedding space more interpretable.

### 3.2.1 Decoder Constraints

The first constraints we introduce are template constraints for the decoder. Motivated by the constraints introduced in Sections 2.2.1.1 and 2.2.1.3, we constrain our decoder matrix $W_{\mathcal{D}}$ to have the same structure as the template matrix $W$ in the musically informed NMF scenario: for each occurring pitch, we want to have one template for the onset and one template for the harmonic component. Therefore, we initialize the weight matrix in the same way as we did for the template matrix. In order to keep the entries initialized by zero to stay at that value, we apply a binary mask $M_W$ as explained in Section 3.1.6.

Results for the decomposition using a musically informed decoder can be seen in Figure 3.7. Apart from the constraints for the decoder, all parameters remain unchanged compared to the random initialization in Figure 3.6. We see that we now mostly achieve the desired structures for our decoder templates. The templates modeling the harmonic components as well as the ones for the onsets are further refined. However, there are still two problems. First, the network can not

(a)



(b)

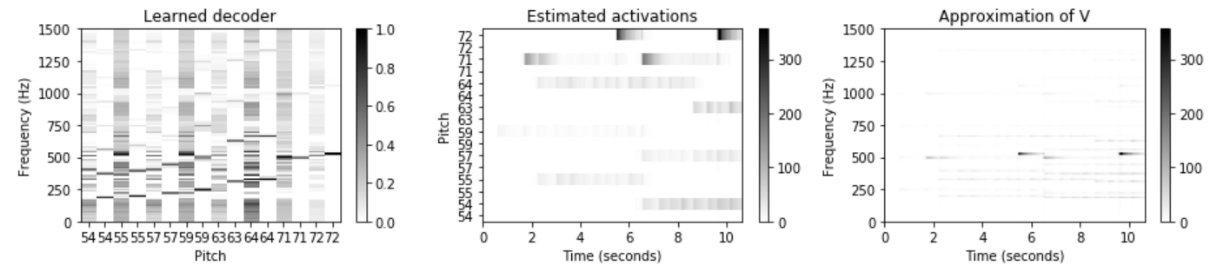


Figure 3.8: NAE-based decomposition using decoder and activation constraints: (a) Decoder initialization, activation mask and input matrix $V$, (b) Resulting approximation.

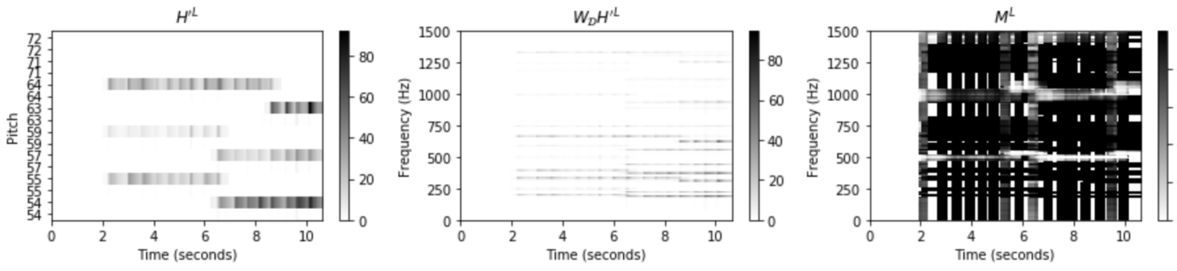
assign the different onset templates to one pitch each. For example, the onset template for MIDI pitch 59 in Figure 3.7b is instead used to model the harmonic behavior of pitch 72. Therefore, it also has undesired activations after six seconds and around ten seconds. Second, there are still undesired activations that do not match the annotation. For example, the harmonic template for MIDI pitch 71 shows some undesired activations before the two seconds time stamp.

### 3.2.2 Activation Constraints

In order to resolve the remaining problems after imposing musically informed constraints on the decoder weights, we also impose constraints on the embedding activations of the network. Our goal is to constrain matrix $H'$ such that the activations match the score-based annotation, similar to the constraints on the activation matrix in NMF. For this, we can use the structured dropout layer introduced in Section 3.1.5. We can take estimated activations $H$ produced by the encoder and explicitly enforce some constraints when computing $H'$. For this, we can directly use the initialization for the activation matrix in NMF as our binary mask $M_H$ as explained in Section 3.1.6.

The resulting decomposition for musically constrained decoder and activations can be seen in Figure 3.8. We see that the results now behave as desired and look similar to the NMF-based

(a)



(b)

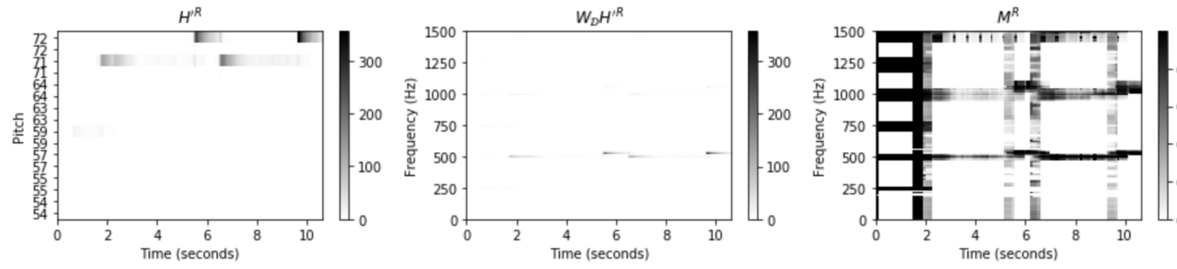


Figure 3.9: Split activation matrices, reconstructed spectrograms and spectral masks for separation of left and right hand: (a) Left hand, (b) Right hand.

results in Figure 2.6. We see that the onset templates now look rather noise like, which solves the problem with the onset template for pitch 59 in Figure 3.7b. Also the undesired activations of the harmonic template for pitch 71 before the two seconds time stamp are eliminated.

### 3.2.3 NAE-based Audio Decomposition

Using this musically informed network, we can use the resulting decomposition to reconstruct separated audio signals for the left and right hand of piano recordings. This is done similar to the procedure introduced in Section 2.2.2. The weight matrix $W_{\mathcal{D}}$ takes the role of the template matrix $W$ from NMF and the activation matrix $H'$ takes the role of the activation matrix $H$ from NMF. The pipeline for the generation of the separated audio signals stays the same. The split activation matrices, reconstructed spectrograms and soft-masks resulting from the application of the musically informed NAE-based framework to the running example are shown in Figure 3.9. We see that the results look similar to the ones using NMF shown in Figure 2.7.

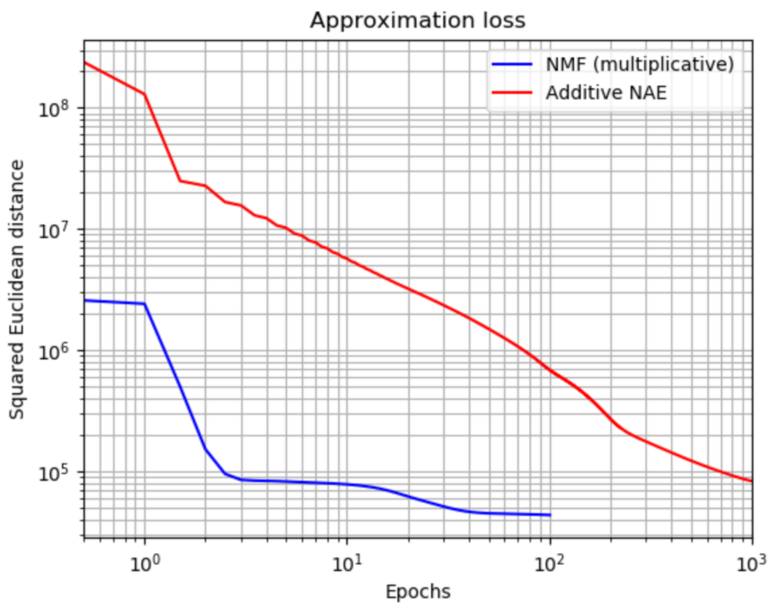


Figure 3.10: Loss curves for NMF with multiplicative and NAE with additive update rules.

## 3.3 Multiplicative Update Rules

One of the biggest drawbacks of the NAE-based audio decomposition compared to the NMF-based one is the number of iterations necessary for a good convergence of the algorithm. This can also be seen in Figure 3.10, where we compare NMF with multiplicative updates to the NAE with additive updates using an SGD optimizer. We use the musically constrained versions of the algorithms as explained in Sections 2.2 and 3.2. Furthermore, we use a learning rate of 0.01 for the encoder and a learning rate of 0.1 for the decoder. We can see that even after 1000 iterations, the autoencoder does not achieve a loss as low as the one for NMF after ten or at most 100 iterations. We therefore see that the NMF-based approach shows a higher convergence speed than the NAE-based one. A possible reason for the higher loss could also be the possibly lower number of parameters of the NAE-based approach as mentioned in Section 3.1.6. However, this is not the case for this short running example. It only occurs for longer examples as we show in Section 4.2. The better quality of the decomposition is also audible in the separation results, where the leakage between components for the additive NAE after 1000 epochs is still stronger than for NMF after 100 iterations. As the number of parameters is not the problem for this short example, we can conclude that the multiplicative update rules cause the faster convergence of the NMF-based approach.
As we saw in Section 2.3, the multiplicative update rules for NMF have the biggest contribution to the fast convergence of the algorithm. Therefore, inspired by the derivations in [18], we derive multiplicative update rules for the used NAE structure. Afterwards, we analyze the convergence behavior of the new update rules compared to other approaches.

We already showed in Section 3.1.7 that the decoder weight matrix $W_{\mathcal{D}}$ mathematically takes the same role as the template matrix $W$ in the NMF framework. Therefore, we concluded we could use the same gradient descent update rule as we use in NMF. The same can be done for the multiplicative update rules. We set the learning rate for matrix $W_{\mathcal{D}}$ to

$$\gamma_{kr}^{(\ell)} = \frac{W_{\mathcal{D},kr}^{(\ell)}}{\left(W_{\mathcal{D}}^{(\ell)} H' H'^{\top}\right)_{kr}}. \tag{3.38}$$

Inserting this into the gradient descent update equation

$$W_{\mathcal{D},kr}^{(\ell+1)} = W_{\mathcal{D},kr}^{(\ell)} - \gamma_{kr}^{(\ell)} \cdot \left( \left(W_{\mathcal{D}}^{(\ell)} H' H'^{\top}\right)_{kr} - \left(V H'^{\top}\right)_{kr} \right) \tag{3.39}$$

results in a multiplicative update rule for the decoder given by

$$W_{\mathcal{D},kr}^{(\ell+1)} = W_{\mathcal{D},kr}^{(\ell)} \cdot \frac{\left(V H'^{\top}\right)_{kr}}{\left(W_{\mathcal{D}}^{(\ell)} H' H'^{\top}\right)_{kr}}. \tag{3.40}$$

Analogously following the corresponding steps for the encoder yields a learning rate of

$$\gamma_{rk}^{(\ell)} = \frac{W_{\mathcal{E},rk}^{(\ell)}}{\left( \left( \left(W_{\mathcal{D}}^{\top} W_{\mathcal{D}} H'^{(\ell)}\right) \odot M_H \right) V^{\top} \right)_{rk}}, \tag{3.41}$$

which in combination with the gradient descent update equation

$$\begin{aligned} W_{\mathcal{E},rk}^{(\ell+1)} = W_{\mathcal{E},rk}^{(\ell)} - \gamma_{rk}^{(\ell)} \cdot \Bigg( & \left( \left( \left(W_{\mathcal{D}}^{\top} W_{\mathcal{D}} H'^{(\ell)}\right) \odot M_H \right) V^{\top} \right)_{rk} \\ & - \left( \left( \left(W_{\mathcal{D}}^{\top} V\right) \odot M_H \right) V^{\top} \right)_{rk} \Bigg). \end{aligned} \tag{3.42}$$

yields the multiplicative update rule for the encoder:

$$W_{\mathcal{E},rk}^{(\ell+1)} = W_{\mathcal{E},rk}^{(\ell)} \cdot \frac{\left( \left( \left(W_{\mathcal{D}}^{\top} V\right) \odot M_H \right) V^{\top} \right)_{rk}}{\left( \left( \left(W_{\mathcal{D}}^{\top} W_{\mathcal{D}} H'^{(\ell)}\right) \odot M_H \right) V^{\top} \right)_{rk}} \tag{3.43}$$

```
Algorithm: MULTIPLICATIVE NAE ($V \approx W_\mathcal{D} H'$)

Input:     Nonnegative matrix $V$ of size $K \times N$
           Rank parameter $R \in \mathbb{N}$
           Activation mask $M_H \in \{0,1\}^{R \times N}$
           Number of iterations $L \in \mathbb{N}$
Output:    Nonnegative decoder matrix $W_\mathcal{D}$ of size $K \times R$
           Nonnegative activation matrix $H'$ of size $R \times N$

Procedure: Define nonnegative matrices $W_\mathcal{E}^{(0)}$ and $W_\mathcal{D}^{(0)}$ by some random or informed initialization. Furthermore set $\ell = 0$. Apply the following update rules (written in matrix notation):

    (1)  $W_\mathcal{D}^{(\ell+1)} = W_\mathcal{D}^{(\ell)} \odot \big((VH'^{(\ell)\top}) \oslash (W_\mathcal{D}^{(\ell)} H'^{(\ell)} H'^{(\ell)\top} + \varepsilon)\big)$
    (2)  $W_\mathcal{E}^{(\ell+1)} = W_\mathcal{E}^{(\ell)} \odot \big((((W_\mathcal{D}^{(\ell+1)\top} V) \odot M_H) V^\top) \oslash (((W_\mathcal{D}^{(\ell+1)\top} W_\mathcal{D}^{(\ell+1)} H'^{(\ell)}) \odot M_H) V^\top + \varepsilon)\big)$
    (3)  $H'^{(\ell+1)} = \big(W_\mathcal{E}^{(\ell+1)} V\big) \odot M_H$
    (4)  Increase $\ell$ by one.

Repeat the steps (1) to (4) until $\ell = L$. Finally, set $H' = H'^{(L)}$ and $W_\mathcal{D} = W_\mathcal{D}^{(L)}$.
```

Table 3.1: Iterative algorithm for learning an NAE-based decomposition. The multiplicative update rules are given in matrix notation, where the operator $\odot$ denotes pointwise multiplication and the operator $\oslash$ pointwise division.

The resulting algorithm using the multiplicative update rules is shown in Table 3.1. We use a machine epsilon in the denominators to avoid division by zero.
These new update rules have the same advantages as the multiplicative update rules for NMF. First, when initializing both weight matrices with nonnegative values, the multiplicative update factor will always be nonnegative. Therefore, we implicitly guarantee that the nonnegativity constraints are not violated. This means we do not have to additionally apply the ReLU function to the network weights after updating them. Second, we can impose constraints similar to the ones in NMF by intelligent initialization of the weights as an initialization by zero keeps the entry fixed at zero for all iterations. This again saves the additional computation for the application of the mask $M_W$. In general, the additional computations related to the projected gradient descent algorithm can be skipped completely. For the decoder the initialization can be done in the same fashion as for the template matrix in NMF. Possibilities for the initialization of the encoder are covered in Section 3.4. Last, the use of multiplicative update rules speeds up the convergence of the network compared to additive updates. We will demonstrate this in the following.

In order to show the better convergence speed of the multiplicative update rules, we show a comparison of the loss curve for our new algorithm with the curves from Figure 3.10 in Figure 3.11. We see that the multiplicative update rules speed up the convergence of the NAE such that

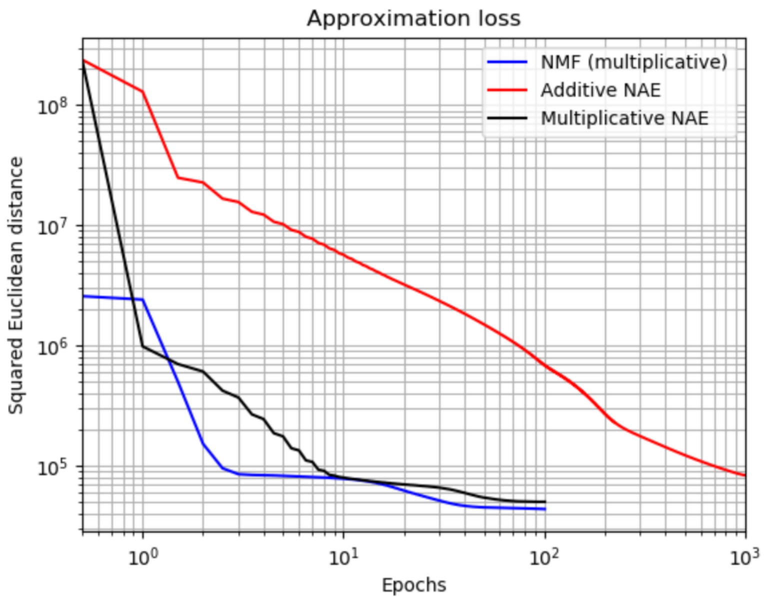


Figure 3.11: Loss curves for NMF with multiplicative and NAE with additive and multiplicative update rules.

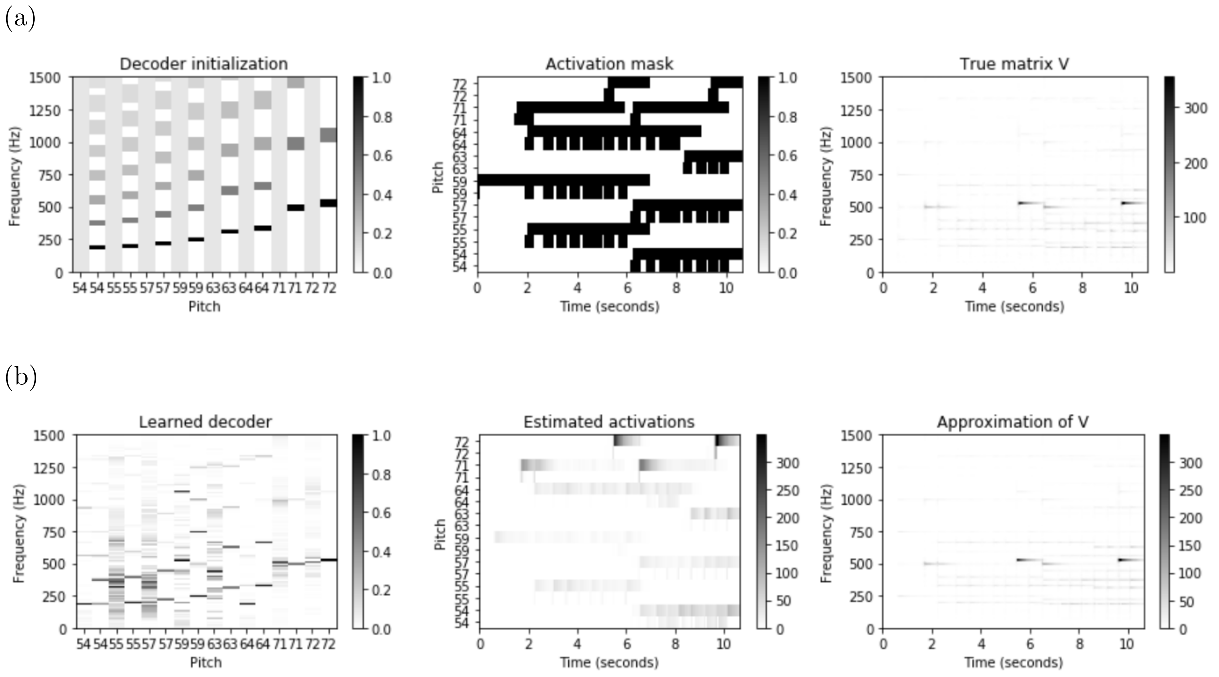


Figure 3.12: NAE-based decomposition using multiplicative update rules: (a) Decoder initialization, activation mask and input matrix $V$, (b) Resulting approximation.

it shows similar behavior to NMF with multiplicative updates. After at most 100 epochs the multiplicative NAE is already at a lower loss value than the additive version after 1000 epochs.

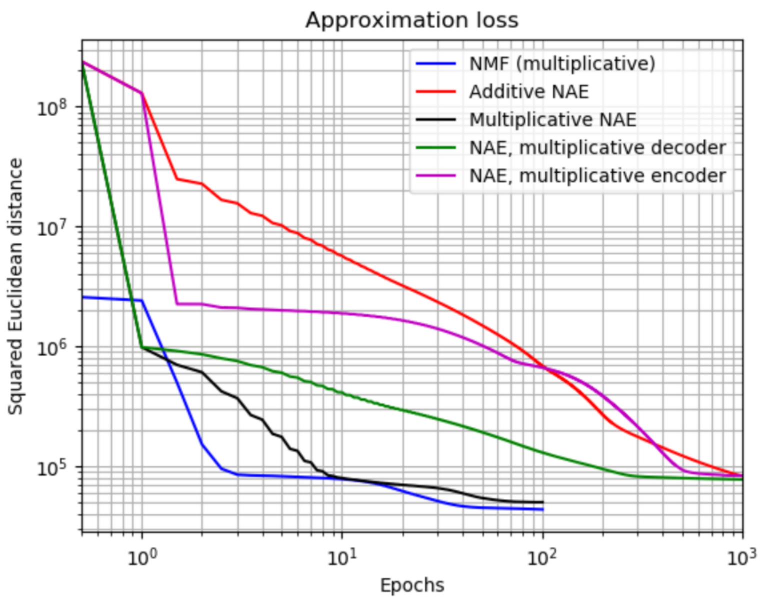


Figure 3.13: Loss curves for different decomposition approaches.

Thus, the new update rules speed up the convergence of the network by a factor of roughly ten. Interestingly, the NAE with multiplicative updates achieves roughly the same loss as NMF. As mentioned before, this is only the case for shorter examples like our running example. For longer examples, as the one in Section 4.2, the loss for the NAE is typically higher.
This better convergence of multiplicative updates can also be observed in the separated audio signals. The quality of the results for the NAE after 100 epochs are similar to the ones for NMF after 100 iterations. Furthermore, the amount of leakage is noticeably reduced when using multiplicative rules and 100 epochs compared to additive rules and 1000 epochs. We also see in the decomposition plotted in Figure 3.12 that the results are quite similar to the ones for NMF shown in Figure 2.5b.

In order to further study the convergence behavior of the newly derived update rules, we show and discuss various curves in the following. We plot these curves for five different approaches: the three approaches from Figure 3.11 as well as one approach where we only use multiplicative updates for the decoder and one where we only use them for the encoder. The other weight matrix is updated using additive SGD update rules with the learning rates mentioned in the beginning of this section.
We compare multiple curves for the different approaches: the loss curves as well as the Cauchy errors for matrices $W_{\mathcal{E}}$, $H'$ ($H$ for NMF), $W_{\mathcal{D}}$ ($W$ for NMF) and $\hat{V}$. Note that we can not plot any NMF-based counterpart for the Cauchy error of $W_{\mathcal{E}}$ as there is no corresponding matrix in

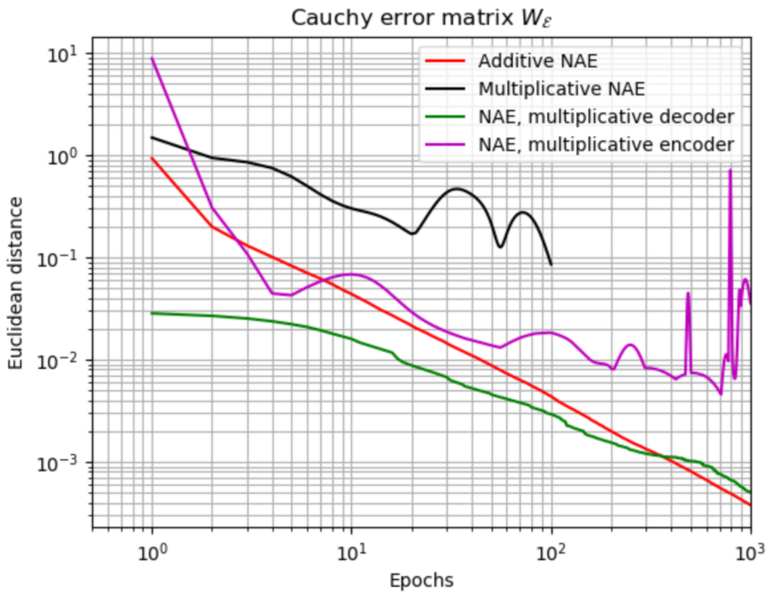


Figure 3.14: Cauchy errors for the encoder matrices of different decomposition approaches.

the NMF framework. We compute the Cauchy errors for the weight matrices after applying the constraints. Therefore, the Cauchy error for the decoder is, for example, defined as

$$C_{W_\mathcal{D}}(\ell) = \left\| \left( \left( W_\mathcal{D}^{(\ell)} \right)^+ \right)' - \left( \left( W_\mathcal{D}^{(\ell-1)} \right)^+ \right)' \right\|. \tag{3.44}$$

In Figure 3.13, we show the loss curves for the different approaches. We can see that out of all NAE approaches, the one with solely multiplicative update rules shows the fastest convergence speed. But also using multiplicative updates for only one of the weight matrices speeds up the convergence compared to the purely additive NAE. The multiplicative updates for the decoder seem to have a stronger effect on the loss curve compared to the ones for the encoder. Interestingly, the loss curves for the approaches where at least one of the matrices is updated using additive update rules stagnate at a higher value than the approaches using only multiplicative update rules even though the number of parameters is equal for all the NAE-based approaches.

Furthermore, we also plot the different Cauchy errors in Figures 3.14, 3.15, 3.16 and 3.17. We see that the multiplicative updates for the encoder seem to cause fluctuations during the decrease of the Cauchy error for matrix $W_\mathcal{E}$. This can be seen at the curve for the multiplicative NAE between epochs 20 and 100 in Figure 3.14. However, these fluctuations are less prominent in the corresponding Cauchy error curve for the activation matrix in Figure 3.15 and therefore seem to be partly eliminated by the structured dropout layer. Apart from that, all the Cauchy errors show mostly monotonously decreasing behavior. Note that the differences in orders of magnitude are caused because we use no normalization during the training. The differences

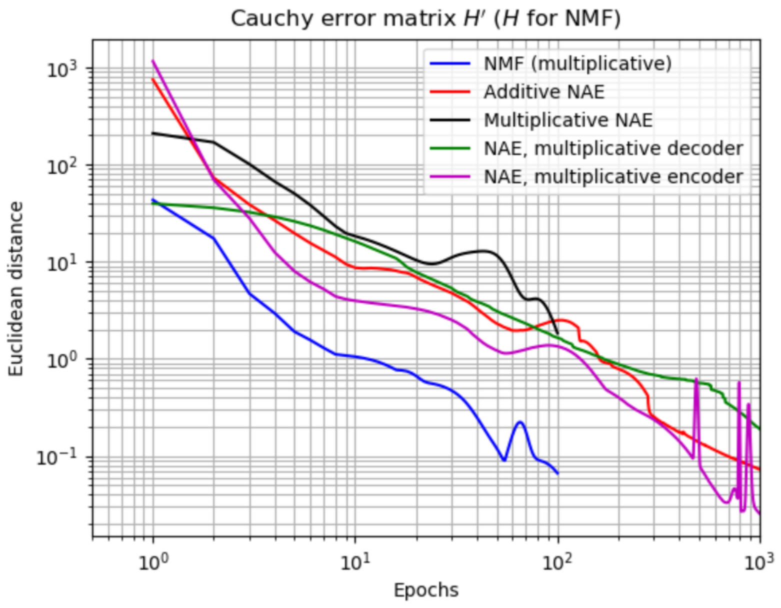


Figure 3.15: Cauchy errors for the activation matrices of different decomposition approaches.

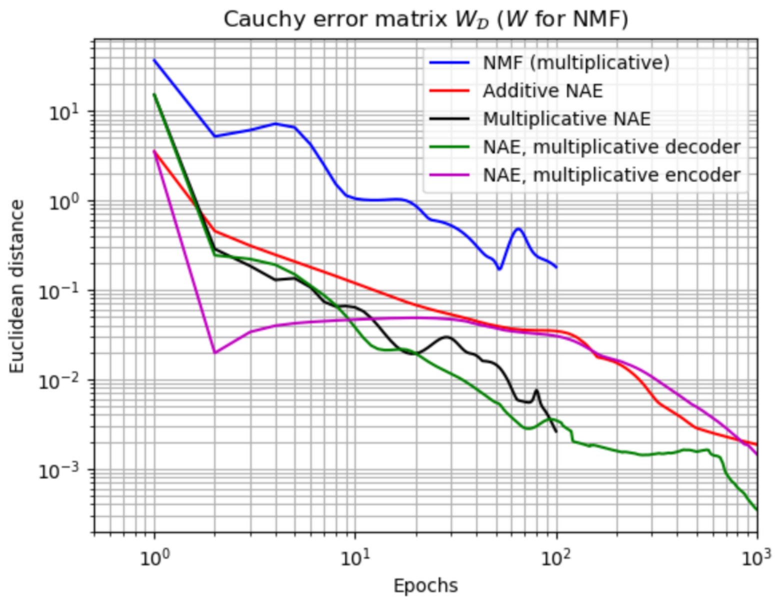


Figure 3.16: Cauchy errors for the template matrices of different decomposition approaches.

occurring between the curves for $W_\mathcal{E}$, and therefore also between the ones for $H'$, are exactly reversed between the curves for $W_\mathcal{D}$ in Figure 3.16. These differences do not occur between the curves for matrix $\hat{V}$ in Figure 3.17, as it is not affected by normalization of the factors.

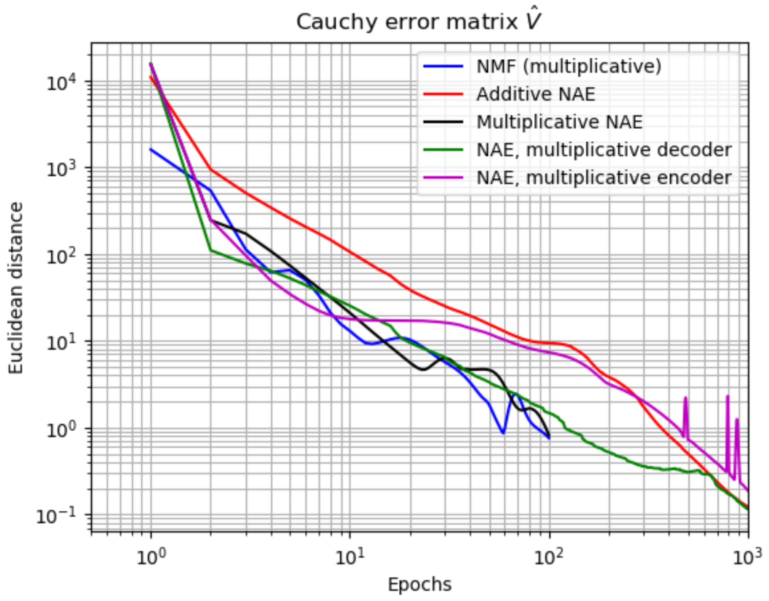


Figure 3.17: Cauchy errors for the approximation matrices of different decomposition approaches.

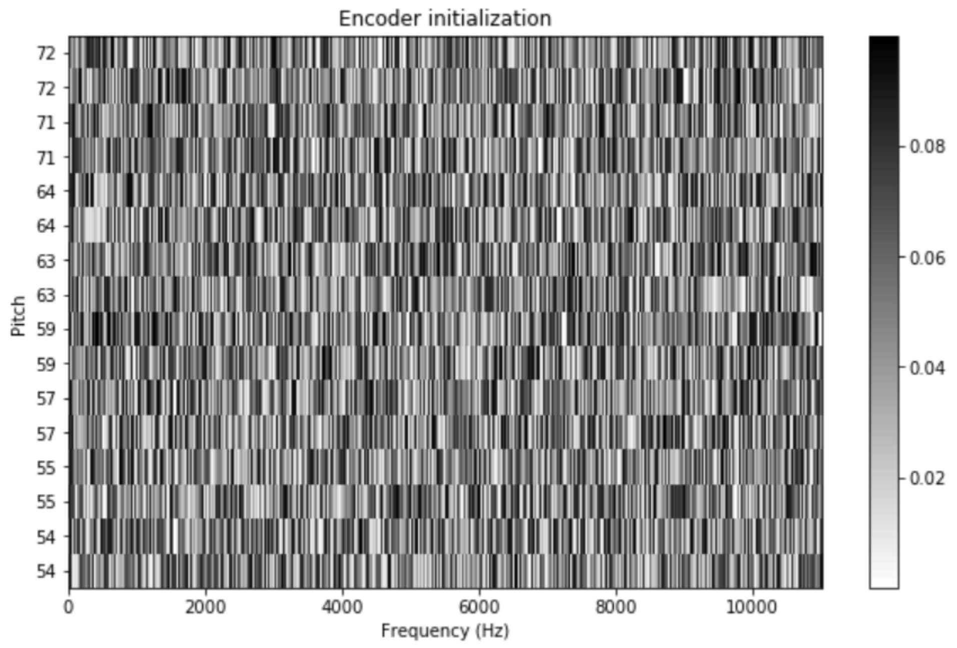
## 3.4 Encoder Initialization

Building upon the multiplicative update rules for the NAE structure, we cover various possibilities for the initialization of the encoder matrix. These different initializations result in different implicitly stated constraints on the encoder weights. Furthermore, we also compare the convergence behavior of the loss function for different encoder initializations. For all of the results in this section, we use the NAE with multiplicative update rules and musically informed constraints for the activations and the decoder weights. We train the networks for 100 epochs.

### 3.4.1 Random Initialization

The easiest and most uninformed initialization is the one by random values. By using this approach, we do not impose any constraints on the encoder weights and introduce no musical information into the encoder network. The initialization of the encoder weights and the resulting weights after training are shown in Figure 3.18. We use musically informed constraints on the decoder and the activations. We see that the encoder mostly focuses on the lower frequency range in order to extract the estimated activations for the different pitches. This makes sense as the encoder can be seen as a transcriber, which attempts to estimate the activity of different notes. Therefore, it focuses on the most dominant frequencies, which are the lower harmonics of the pitch.

(a)



(b)

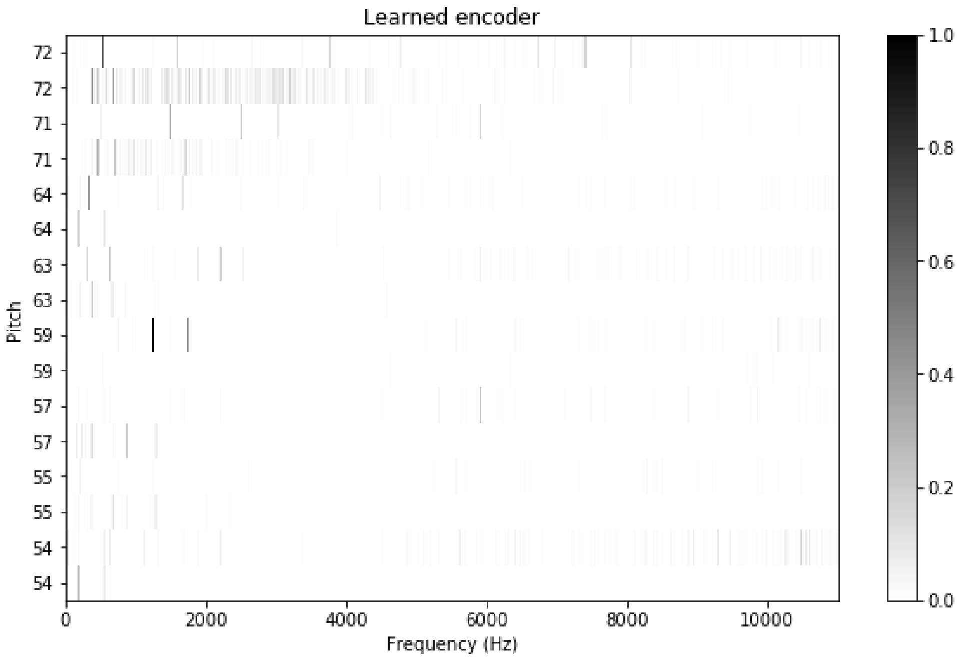
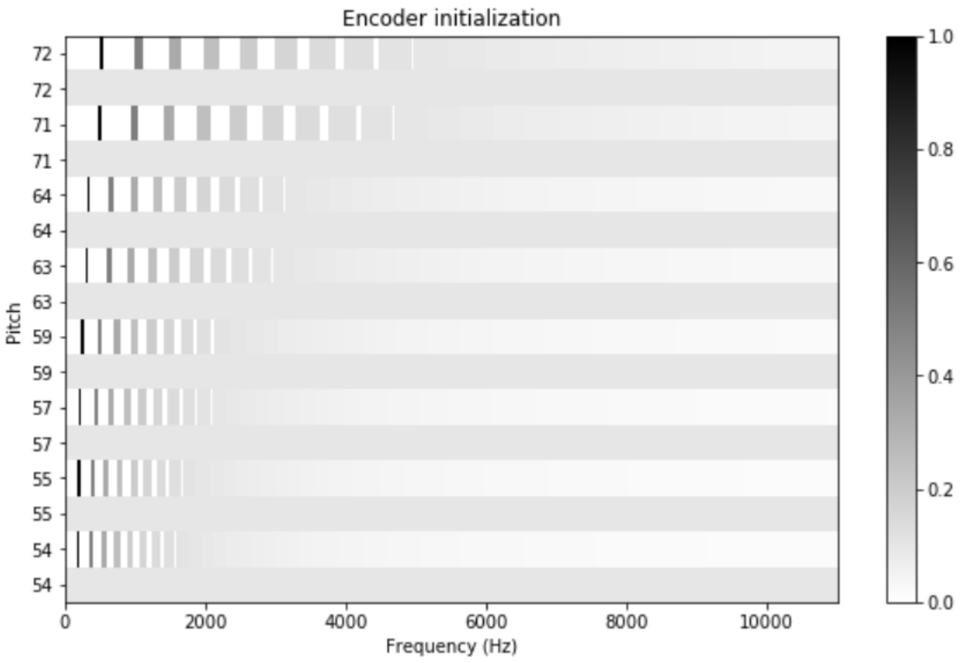


Figure 3.18: Encoder results for random encoder initialization: (a) Encoder initialization, (b) Learned encoder weights.

### 3.4.2 Musically Informed Initialization

As we already mentioned, the encoder seems to focus on certain frequencies for transcribing from spectrogram to activations. Thus, we could try to guide this into a direction, which we can interpret well. One possibility to do so is to use a musically informed initialization for the encoder as well. In the same fashion as for the decoder, we constrain the weights for the harmonic components to only look at certain frequencies. To do so, we can simply take the transpose of the decoder initialization as the initial values for the decoder weights.
The corresponding results are illustrated in Figure 3.19. We can once again see the desired harmonic structure in the initialization. The learned weights however look very similar to the ones for random initialization in Figure 3.18b. This could mean that the musically informed constraints on the decoder and the activations already guide the optimization towards a solution where the encoder shows musically motivated behavior even without using musically informed encoder initialization.

(a)



(b)

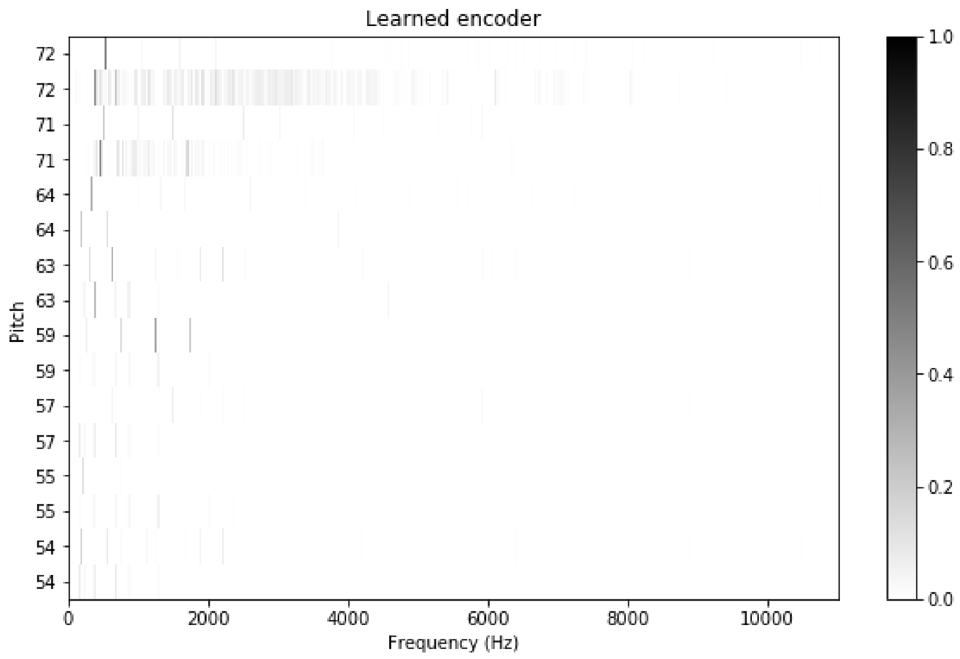
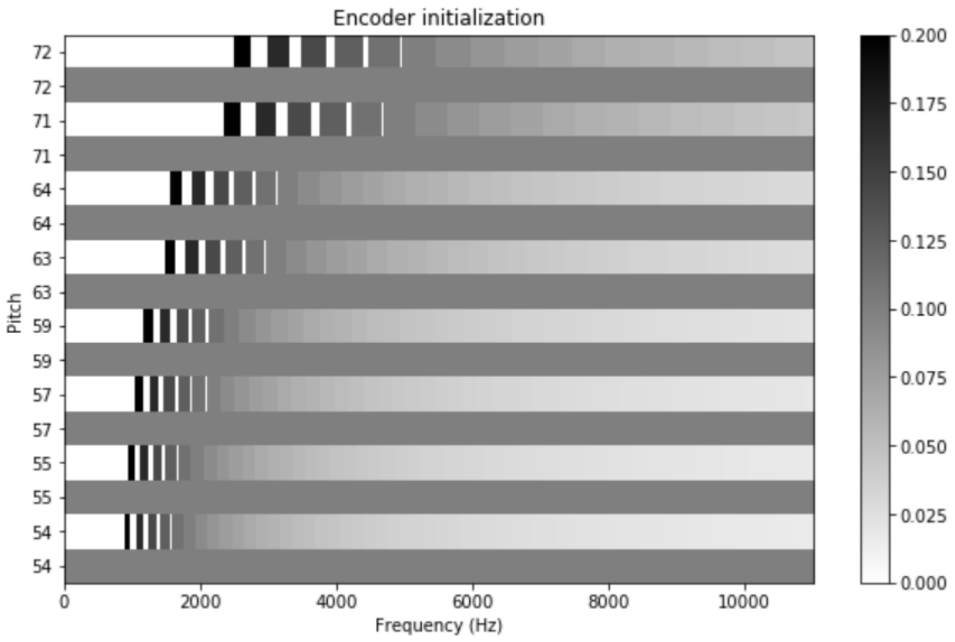


Figure 3.19: Encoder results for musically informed encoder initialization: (a) Encoder initialization, (b) Learned encoder weights.

### 3.4.3 Higher Harmonics

We just saw that also for the musically informed initialization, the encoder still focuses on the lower harmonics. Therefore, an alternative could be to further constrain the encoder such that it can only use higher harmonics. This can, due to the multiplicative updates, be done by setting areas in the rows for harmonic templates to zero. We choose a minimum harmonic and set the frequency areas for all harmonics below it to zero. This could be of advantage as the resolution w.r.t. the logarithmic frequency axis increases for higher frequencies.
The resulting initialization and learned weights for the encoder are shown for the exemplary case of choosing the fifth harmonic as the minimum harmonic in Figure 3.20. We see that there are significantly more components visible in the higher frequency ranges than for the cases before. This is what we expected, as the encoder can no longer focus on the more dominant lower harmonics.

(a)



(b)

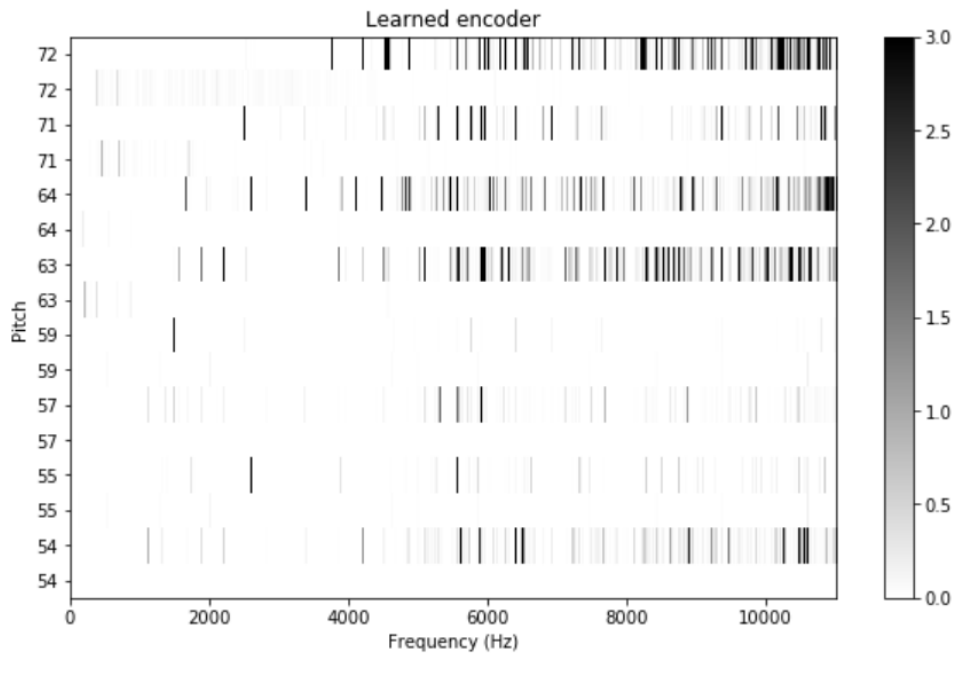


Figure 3.20: Encoder results for musically informed encoder initialization with a minimum harmonic of five: (a) Encoder initialization, (b) Learned encoder weights.

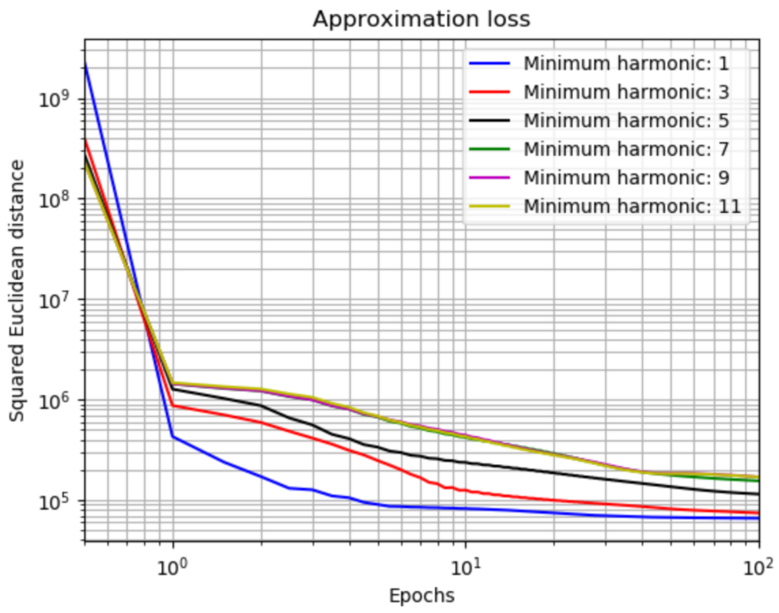


Figure 3.21: Loss curves for musically informed encoder initialization with different numbers of minimum harmonics.

We compare the loss curves for different numbers of minimum harmonics in Figure 3.21. It becomes clear that choosing higher numbers for the minimum harmonic results in higher loss values. This can be explained by the fact that setting more areas to zero means more constraints for the encoder.

### 3.4.4 Initialization Comparison

We sum up the results for encoder initialization by comparing the loss curves for three approaches in Figure 3.22. We compare random initialization from Section 3.4.1, musically informed initialization from Section 3.4.2 and the initialization using higher harmonics from Section 3.4.3, where we use the fifth harmonic as the minimum harmonic. We can see that the random and the musically informed initialization yield very similar loss curves. While the loss for the informed initialization decreases faster in the beginning, the random initialization achieves a lower loss value at the end of the training. The stronger constraints resulting from the initialization using only higher harmonics naturally yield a higher loss value.

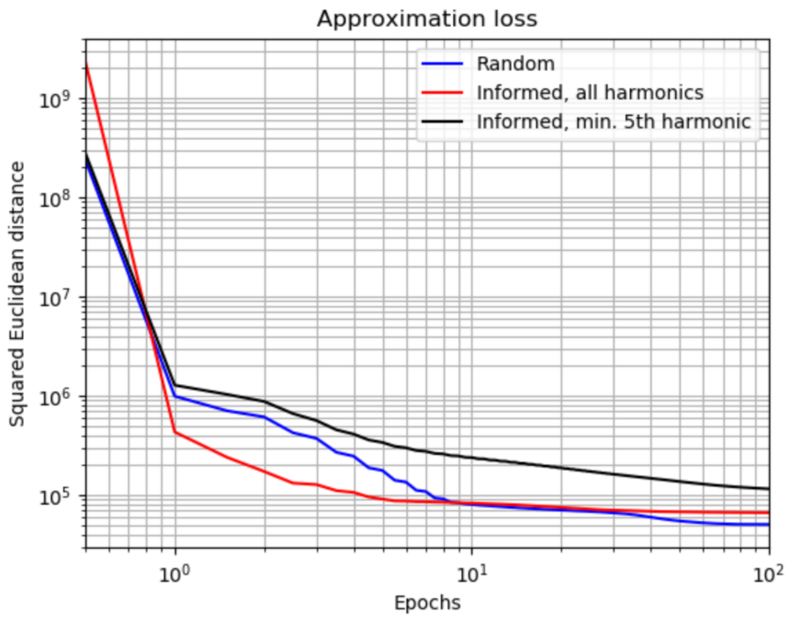


Figure 3.22: Loss curves for different encoder initializations.

# Chapter 4

# Experiments

In Chapters 2 and 3, we introduced different techniques for decomposition of music recordings. So far, we have only evaluated these techniques on a short running example. Therefore, we cover the application to a bigger dataset in this chapter. First, we introduce the used dataset in Section 4.1. Afterwards, we discuss the results for the application of various approaches on this dataset in Section 4.2. Finally, we cover Jupyter notebooks with audio examples in Section 4.3.

## 4.1 Dataset

The application scenario for this thesis is the decomposition of piano recordings into the signals for the left and the right hand as explained in Section 2.2. Therefore, we need a dataset with recordings of solo piano performances. Furthermore, we also need annotation data for the audio files containing information about the assignment of note events to one of the two hands.

Big parts of this thesis are inspired by the work about musically informed NMF for source separation by Ewert et al. in [9]. Their work also covers the task of the separation of the left and the right hand of piano recordings. Therefore, we use the dataset used for the demonstration of their results as an exemplary dataset for the experiments with our methods. The dataset contains recordings of live performances of classical solo piano pieces by different performers as well as the corresponding annotations. There are eight recordings, each of them being the performance of a different song with an average length of around four minutes and a total length of the complete dataset of roughly 30 minutes. We give an overview over the different examples together with their tags for further reference in Table 4.1. Note that we have nine examples in our dataset as we treat the short running example as a separate one. Furthermore, note that some of the performers' names are anonymized because the corresponding examples are part of the Saarland Music Data (SMD) dataset [25], which only contains anonymous performers named

| **Tag** | **Composer** | **Piece** | **Performer** |
|---|---|---|---|
| CH04B | Chopin | Beginning of Op. 28, No. 4 (Running example) | SMD-003 |
| CH04 | Chopin | Op. 28, No. 4 | SMD-003 |
| BA01 | Bach | BWV 875 | SMD-002 |
| BE31 | Beethoven | Op. 31 No. 2 | SMD-002 |
| BE11 | Beethoven | Op. 111, No. 1 | E. Petri |
| CH01 | Chopin | Op. 28, No. 1 | SMD-003 |
| CH15 | Chopin | Op. 28, No. 15 | SMD-006 |
| CH64 | Chopin | Op. 64, No. 1 | D. Lipatti |
| CH66 | Chopin | Op. 66 | SMD-006 |

Table 4.1: Examples in the used piano dataset.

by a three digit number. We therefore give these performers names of the form SMD-*nnn*, where *nnn* denotes the number assigned to the performer in the SMD dataset.

## 4.2 NMF and NAE Experiments

In the previous chapters, we only applied our methods to a small running example. However, one of the advantages of the NAE compared to NMF lies in the processing of longer examples as mentioned in Section 3.1. The number of parameters for the NAE structure is given by $2 \cdot K \cdot R$, where $K$ is again the number of frequency bins and $R$ the number of templates. Therefore, the number of parameters does not directly depend on the number of time frames $N$. For NMF on the other hand, the number of parameters is given by $K \cdot R + R \cdot N$, which means that the number of parameters directly increases when using longer examples. As the number of pitches, and therefore the number of templates $R$, typically increases when the piece is longer, the number of parameters for the NAE structure can still be considered indirectly dependent on the length of the example. But still, the number of parameters for NMF increases way more drastically for increasing values of $N$. Furthermore, there is an upper bound of $2 \cdot 88$ for $R$ as there are only 88 keys on a piano.

So far, we have only considered a separated processing of the dataset examples, where the NMF and NAE algorithms were applied to each example individually. In this section, we show results where we process all examples of the dataset jointly. To do so, we concatenate them along the time axis and apply the algorithms on this one long example. To begin with, we show the loss curves for various musically informed approaches in Figure 4.1. We show NMF with multiplicative updates and NAEs with different update rules. We use additive updates using two different approaches. On the one hand we use our own approach using the SGD optimizer with individual learning rates for the weight matrices. On the other hand we also use the approach by Suárez [30]. This approach uses the RMSprop optimizer with the parameters proposed in

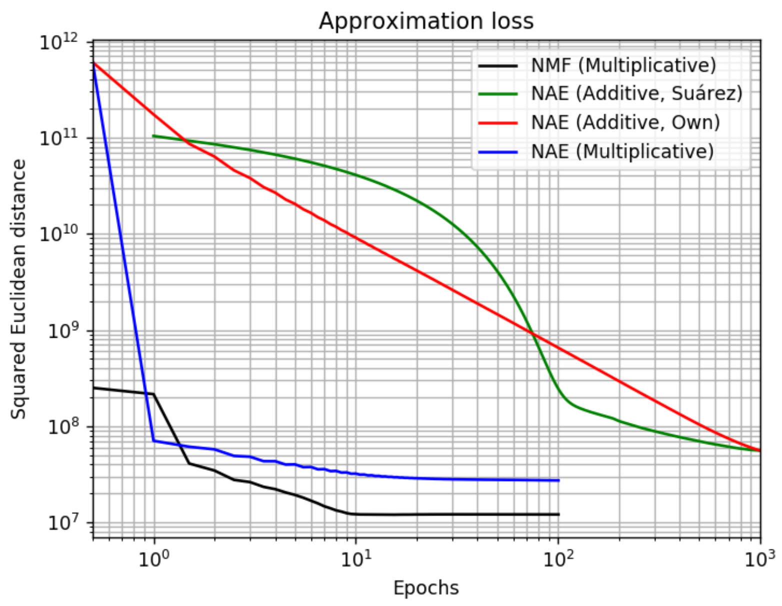


Figure 4.1: Loss curves for different decomposition approaches using the whole dataset.

Section 3.1.8.2. Furthermore, we also show the results for the NAE with multiplicative update rules. We can see that the multiplicative update rules for the NAE structure once again speed up the convergence rate and enable the algorithm to converge as fast as the NMF algorithm with multiplicative updates. However, the final loss value for the NAE-based approaches is higher than the one for NMF. This can be explained by the fact that NMF has a higher number of parameters for longer examples. In this example, NMF has $3 \cdot 10^6$ parameters compared to $3 \cdot 10^5$ parameters for the NAE structure. Therefore, the NMF model is more complex and can achieve a lower loss value for the approximation. We can also see that the curve for the SGD optimizer does not seem to have fully converged yet. Furthermore, the approach by Suárez shows an interesting behavior caused by the RMSprop optimizer: in the beginning, the loss curve is quite flat but once the loss starts to decrease more steeply it overtakes the curve for the SGD optimizer. At the end, both additive NAE optimizers achieve the same loss but do not seem to be fully converged. Note that the curve for the approach by Suárez starts later and at a lower loss value than the other NAE approaches because this approach is the only one implemented in Keras [5], which results in some slight differences regarding the implementation of the loss value computation.

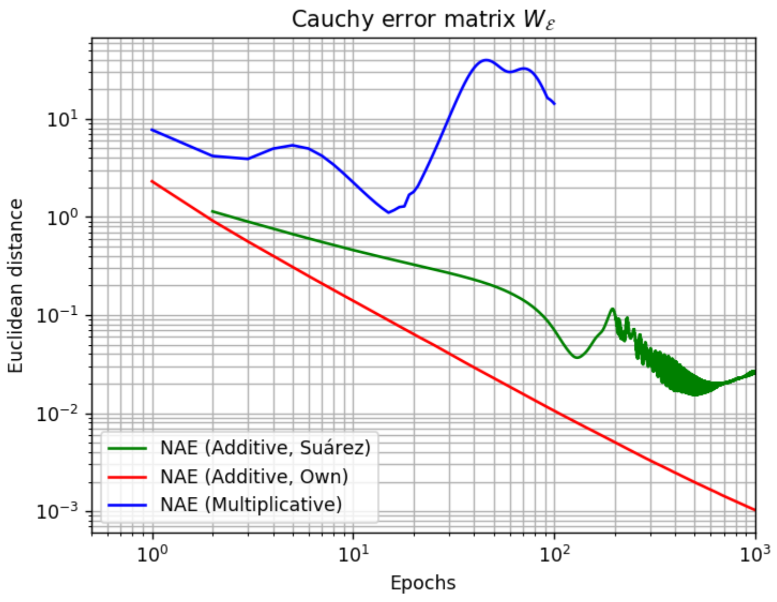


Figure 4.2: Cauchy errors for the encoder matrices of different decomposition approaches using the whole dataset.

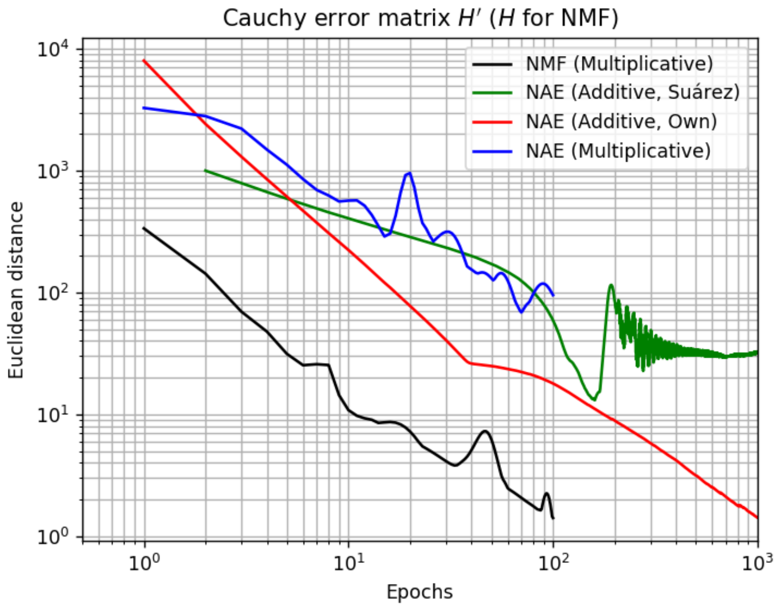


Figure 4.3: Cauchy errors for the activation matrices of different decomposition approaches using the whole dataset.

Furthermore, we also show the according Cauchy errors in Figures 4.2, 4.3, 4.4 and 4.5. We see that the behavior of the curves for the approaches compared in Section 3.3 is mostly similar to the behavior of the corresponding curves shown in Figures 3.14 to 3.17. The curve for the NAE with multiplicative updates once again shows some oscillations and rises of the Cauchy error for

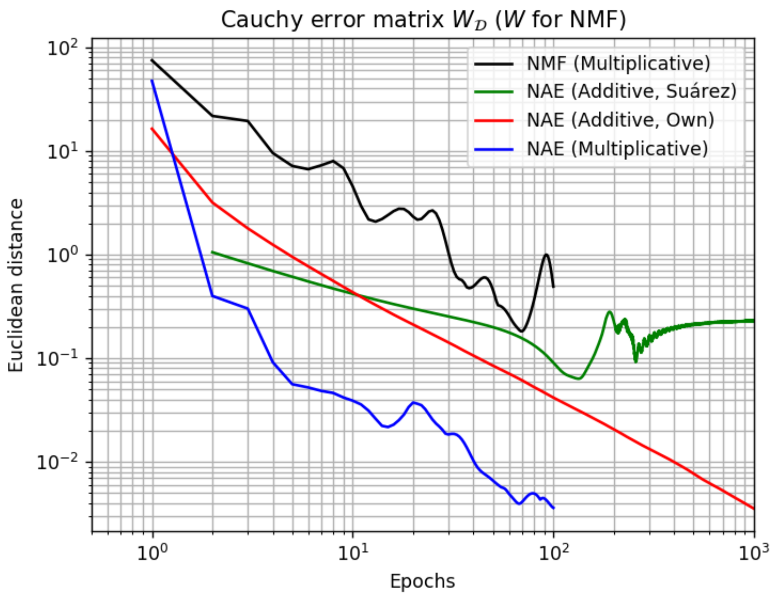


Figure 4.4: Cauchy errors for the template matrices of different decomposition approaches using the whole dataset.

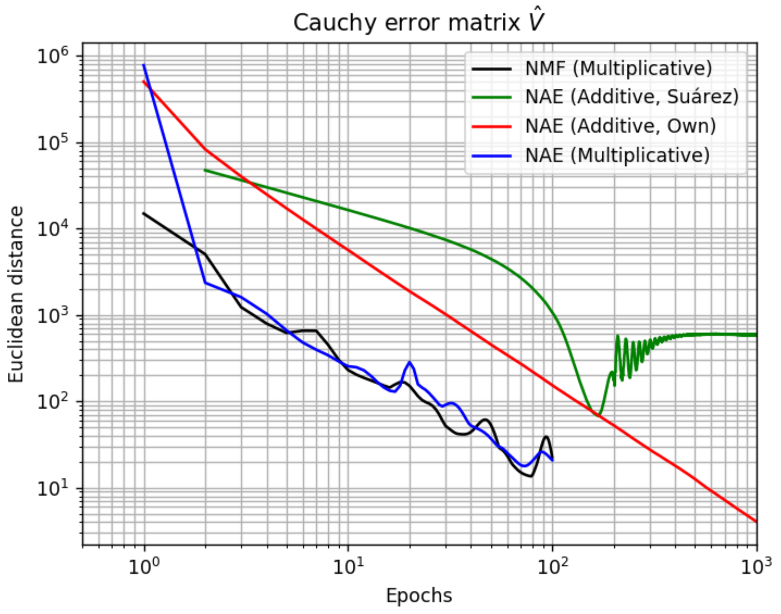
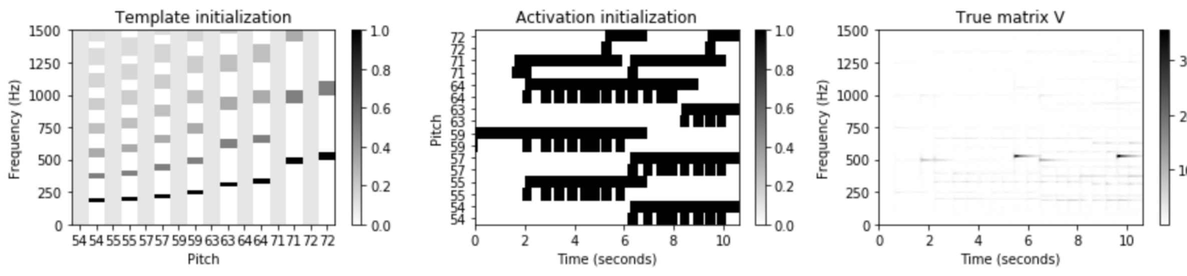


Figure 4.5: Cauchy errors for the approximation matrices of different decomposition approaches using the whole dataset.

matrix $W_{\mathcal{E}}$ in Figure 4.2, which are not visible for matrix $H'$ in Figure 4.3 after the structured dropout. Apart from this, all Cauchy error curves seem to be decreasing except for the approach by Suárez, which uses the RMSprop optimizer. Each of the Cauchy error curves for this approach only shows a very slow decay of the Cauchy error compared to the other approaches. Furthermore,

(a)



(b)

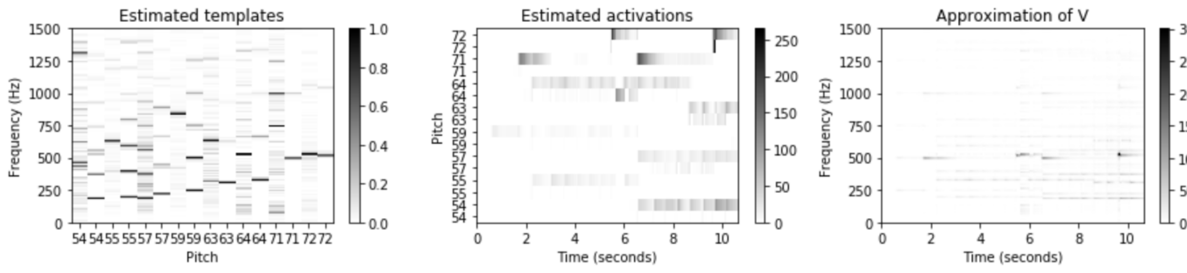
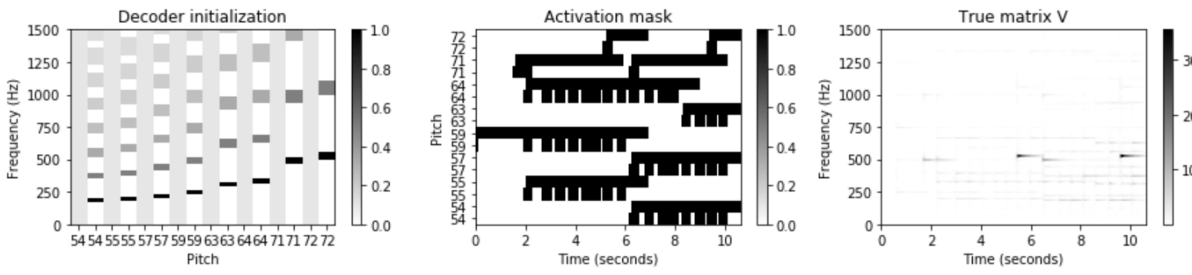


Figure 4.6: NMF-based audio decomposition results for the running example using joint processing of all examples: (a) Initializations and input matrix $V$, (b) Resulting decomposition.

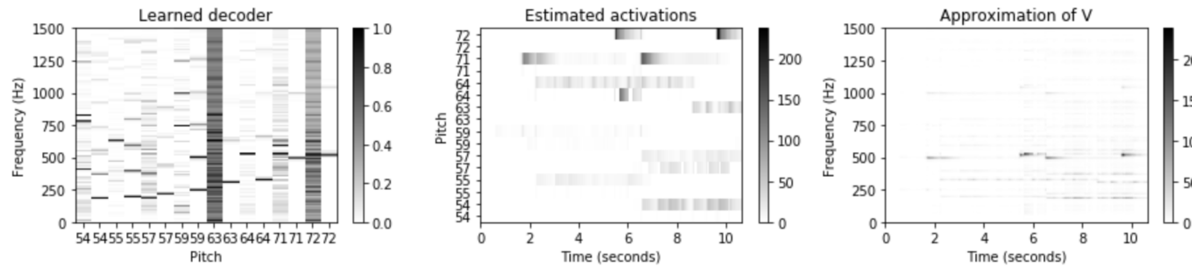
each of the Cauchy errors for the approach by Suárez seems to stagnate at a relatively high level after around 200 epochs. This can especially be seen in Figure 4.5. Interestingly, this stagnation occurs shortly after sharp drop of the loss in Figure 4.1. This behavior seems to be caused by the RMSprop optimizer as this is the only difference for the approach by Suárez and also occurs for a separated processing of the examples.

After discussing the different curves describing the convergence behavior, we also show the resulting decompositions for the running example for the various approaches in Figures 4.6 and 4.7. To do so, we take the result of the jointly processed dataset and reverse the concatenation by corresponding splitting along the time axis. Furthermore, we only plot the templates and activations for the pitches which are active in the example. We can see some problems in the resulting approximations of $V$, where one of the most dominant pitches in the approximations of $V$ at around 500 Hz and 5.5 seconds seems to be oscillating in Figures 4.6b, 4.7b and 4.7d. This is probably caused by the bigger number of examples in the dataset as it does not occur for NMF in Figure 2.6b or for the NAE with multiplicative updates in 3.12b. As we process various performances at once, the pitch models have to model different piano models and quite possibly also different tunings by just one frequency model. As one could expect, this leads to some problems and would probably not occur for datasets containing only examples from the same piano model and tuning. One example for a problem caused by different tunings can be seen

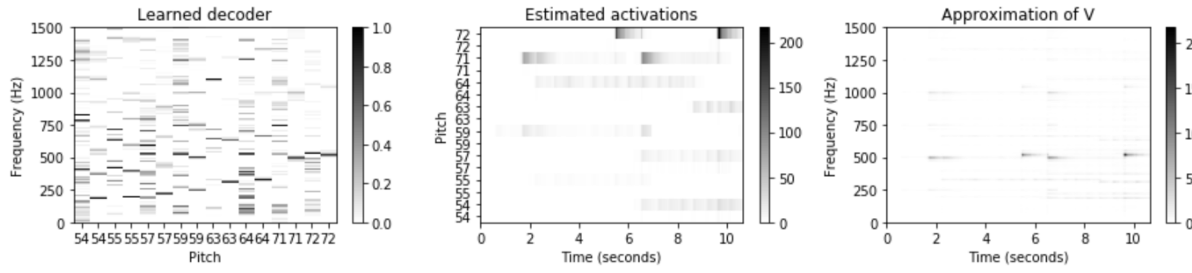
(a)



(b)



(c)



(d)

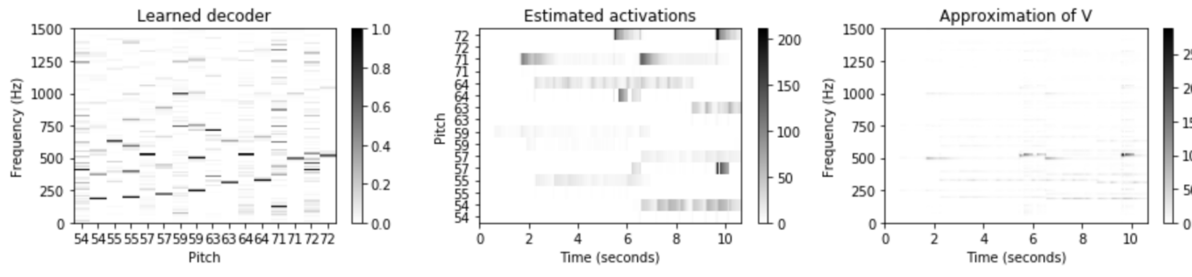
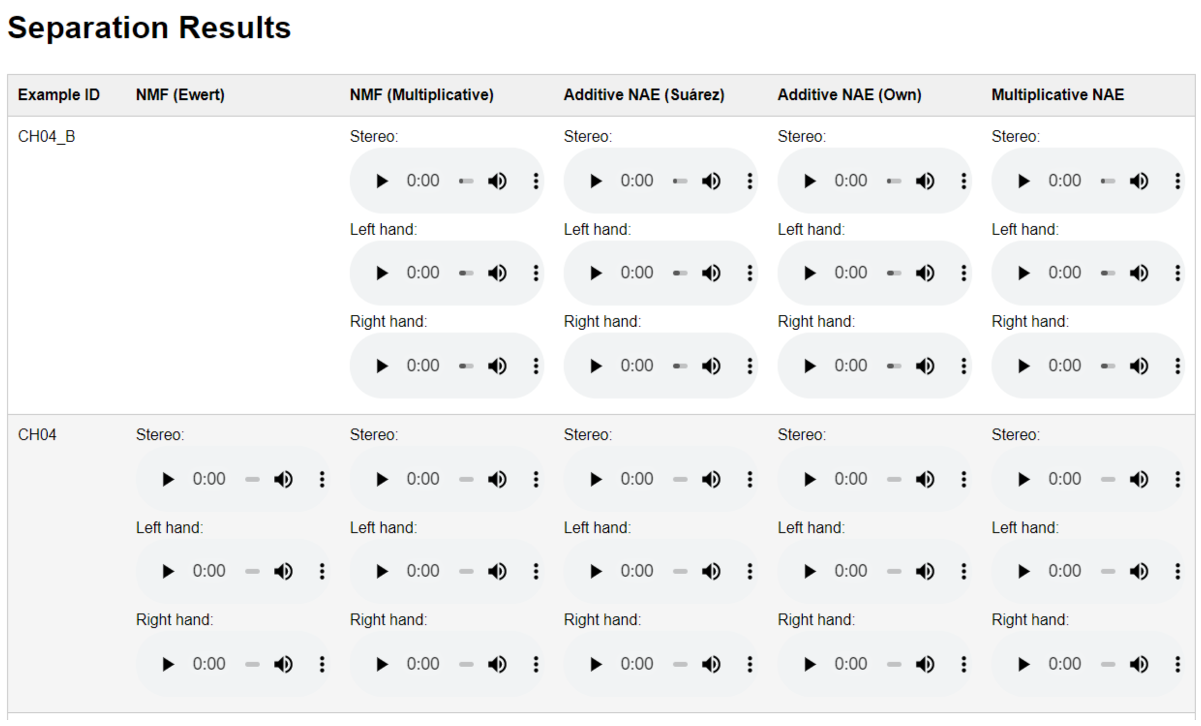


Figure 4.7: NAE-based audio decomposition results for the running example using joint processing of all examples: (a) Decoder initialization, activation mask and input matrix $V$, (b) Resulting decomposition using the approach by Suárez, (c) Resulting decomposition using additive updates with individual learning rates for encoder and decoder, (d) Resulting decomposition using multiplicative updates.

for the decoder templates in Figure 4.7c. When looking at the third harmonic in the harmonic template for MIDI pitch 54, we see that there are two frequency bins with higher positive values. However, we would only expect one dominant frequency bin for the harmonic.
When comparing the results for the NAE with multiplicative updates in Figure 4.7d with the ones for separate processing of the examples in Figure 3.12b, we see that the most significant difference can be seen in the decoder templates for the onsets. The results when using the complete dataset jointly are more sparse compared to the ones when processing the running example alone. The same can be seen when comparing Figures 4.6b and 2.6b for NMF. Therefore, the usage of a bigger dataset seems to have a regularizing effect on the onset templates that makes the frequency representations more sparse.
When comparing the results for the different approaches using the whole dataset, we see that the results for NMF in Figure 4.6b and for the NAE with multiplicative updates in Figure 4.7d look very similar. The approach by Suárez in Figure 4.7b seems to experience some problems as the onset templates for MIDI pitches 63 and 72 are still roughly constant along the frequency axis and therefore look more similar to the initialization in Figure 4.7a than we would expect and desire. Overall, we can conclude that the results for the NAE with multiplicative update rules look the best among all NAE approaches in Figure 4.7.



**Separation Results**

| Example ID | NMF (Ewert) | NMF (Multiplicative) | Additive NAE (Suárez) | Additive NAE (Own) | Multiplicative NAE |
|---|---|---|---|---|---|
| CH04_B | | Stereo:<br>▶ 0:00<br>Left hand:<br>▶ 0:00<br>Right hand:<br>▶ 0:00 | Stereo:<br>▶ 0:00<br>Left hand:<br>▶ 0:00<br>Right hand:<br>▶ 0:00 | Stereo:<br>▶ 0:00<br>Left hand:<br>▶ 0:00<br>Right hand:<br>▶ 0:00 | Stereo:<br>▶ 0:00<br>Left hand:<br>▶ 0:00<br>Right hand:<br>▶ 0:00 |
| CH04 | Stereo:<br>▶ 0:00<br>Left hand:<br>▶ 0:00<br>Right hand:<br>▶ 0:00 | Stereo:<br>▶ 0:00<br>Left hand:<br>▶ 0:00<br>Right hand:<br>▶ 0:00 | Stereo:<br>▶ 0:00<br>Left hand:<br>▶ 0:00<br>Right hand:<br>▶ 0:00 | Stereo:<br>▶ 0:00<br>Left hand:<br>▶ 0:00<br>Right hand:<br>▶ 0:00 | Stereo:<br>▶ 0:00<br>Left hand:<br>▶ 0:00<br>Right hand:<br>▶ 0:00 |

Figure 4.8: Layout of the provided Jupyter notebooks.

## 4.3 Jupyter Notebooks

As one of the best ways to evaluate the quality of the decomposition results is listening to the separated audio signals for the left and the right hand, we also provide Jupyter notebooks accompanying this thesis. We provide three notebooks containing the separation results for the whole dataset and various approaches. The structure of these notebooks can be seen in Figure 4.8. One cell contains the result for one combination of example and approach. In this cell we provide the mono audio signals for the left and the right hand individually as well as a stereo signal with the signals for both hands stored in the corresponding channels.

The approaches contained in each notebook are summarized in Table 4.2. We provide two notebooks with separate processing of the individual examples without concatenation of all examples along the time axis. The first one contains NMF-based results, namely the ones by Ewert [9] and the ones recreated by us. Furthermore, it contains results for the NAE-based approaches using additive updates: on the one hand the approach by Suárez [30] and on the other hand our own approach. Additionally, it also contains our NAE-based results using the multiplicative update rules. The second one contains some alternative approaches. These are the NAE-based approaches using a mix of additive and multiplicative update rules. The last notebook provides results for the approaches from the first notebook (apart from the results by Ewert), but with a joint processing of all examples at once. We use a training time of 100 epochs

| **Notebook name** | **Processing** | **Approaches** |
|---|---|---|
| `audio_demo.ipynb` | Separate | Mult. NMF, Ewert<br>Mult. NMF, Own<br>Add. NAE, Suárez<br>Add. NAE, Own<br>Mult. NAE |
| `audio_demo_alternatives.ipynb` | Separate | NAE, mult. decoder<br>NAE, mult. encoder |
| `audio_demo_allAtOnce.ipynb` | Joint | Mult. NMF, Own<br>Add. NAE, Suárez<br>Add. NAE, Own<br>Mult. NAE |

Table 4.2: Contents of the different Jupyter notebooks.

for the NMF-based results and for the NAE-based results with multiplicative updates. For the remaining NAE-based approaches we use 1000 epochs.
When listening to the results for separate processing of the examples, we can make various observations. First, our NMF-based results for the separate processing of the examples achieve results very similar to the ones by Ewert. Second, the NAE-based approach with multiplicative update rules achieves results comparable to the NMF-based ones. Third, the NAE-based results with additive updates for one or both of the weight matrices achieve worse results with noticeably more leakage. Thus, the NAE with multiplicative updates achieves the best results among all NAE-based approaches even though it is only trained for 100 instead of 1000 epochs.
When listening to the results for joint processing of all examples, we can observe a severe degradation in quality for the separated audio signals. This is consistent for all approaches. For the running example, the effect of leakage from the right hand signal to the left hand one is more prominent. For many of the other examples, there are undesired distortions and pauses in the audio signals. Therefore, we can conclude that the joint processing of our complete dataset is not suited for the investigated approaches. Reasons for this could be the differences in piano models, tunings or possibly also recording devices and quality.

# Chapter 5

# Conclusions

In this thesis, we covered the task of music decomposition. To do so, we used NMF- and NAE-based approaches for audio decomposition. More specific, we dealt with the task of the separation of the left and the right hand for piano recordings. We introduced NMF with multiplicative update rules and showed how we can use musically informed approaches to use it for music decomposition. Furthermore, we studied the convergence behavior of different NMF approaches. From this, we concluded that the multiplicative updates and the musically informed constraints speed up the convergence of the algorithm. The latter had the less significant impact. Additionally, we also studied the practically occurring problem of the rising loss for multiplicative NMF. We showed that the problem is caused by values of zero or close to zero in the initialization. Afterwards, we introduced nonnegative autoencoders, which are neural networks with special constraints. We furthermore introduced musical constraints for the network, such that the results are comparable to the ones from NMF. As our biggest contribution, we derived multiplicative update rules for the network weights, which result in a speed-up of the convergence speed by a factor of ten. These multiplicative updates also enabled us to impose constraints on the network weights through informed initialization instead of more complex projected gradient approaches. Based on this, we compared various initialization approaches for the encoder weights resulting in different constraints. Finally, we performed mass experiments with the introduced approaches and provided additional Jupyter notebooks containing audio signals with the separation results. The NAE-based approach using multiplicative updates yielded promising results compared to other NAE-based approaches. We saw some problems regarding the decomposition and the separation quality for all approaches when processing all examples of the dataset at once, possibly caused by the diversity of the different piano models and tunings.

There are various directions for future work on this topic. As we only used the squared Euclidean distance as a loss function, one could instead use other popular choices as the Kullback-Leibler or the Itakura-Saito divergence and study the effect of different choices for the loss function.

Another point could be the complexity of the network. Opposed to the rather shallow and mostly linear networks we used in this thesis, one could use deeper networks with more layers or different activation functions as they could boost the performance of the network. Additionally, one could then try to derive multiplicative updates for these network structures, similar to our derivations. Furthermore, one could investigate convolutional or recurrent networks as they are also able to model dependencies between subsequent time frames. The input representation could be an important point as well. One could use other frequency representations such as the constant-Q transform instead of the STFT, experiment with the choice of parameters for the STFT or apply different compression techniques such as log-compression to the input. Another topic to study could be the relationship between encoder and decoder weights of the network, from which one could derive additional constraints for the weights. Finally, one could apply the approaches introduced in this thesis to other datasets. These could, e. g., either be piano datasets only containing data from one piano model or datasets with other instruments or mixtures of them.

# Appendix A

# Source Code

In this chapter, the headers of selected 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.
We cover our implementation of the NMF algorithm as well as two different implementations of the NAE. Implementations were done in Python using LibROSA [23], NumPy [14] and Keras [5]. We also use the libraries LibFMP (part of the FMP notebooks [26]) and LibAE. The latter is a product of the thesis by Suárez [30].
Additionally, we also provide notebooks containing examples for the usage of the functions mentioned in the following.

```
def NMF_methods(V, R, L=1000, W=None, H=None, updates_mult=True, norm=False, use_epsilon=True,
                learning_rate=0.0001, step_thresh=(0.01, 0.001)):

    """NMF algorithm with Euclidean distance. (Based on LibFMP code)

        Parameters
        -----------
            V : array-like
                Nonnegative matrix of size K x N

            R : int
                Rank parameter

            L : int
                Number of iterations

            W : array-like
                Nonnegative matrix of size K x R used for initialization

            H : array-like
                Nonnegative matrix of size R x N used for initialization

            updates_mult : bool
```

```
                When True, uses multiplicative updates for H and W

            norm : bool
                When True, applies max-normalization of columns of final W

            use_epsilon : bool
                When True, adds machine epsilon in denominator

            learning_rate : float > 0
                Learning rate for additive updates

            step_thresh : array-like
                Thresholds to check for update steps

        Returns
        -------
            W : array-like
                Nonnegative matrix of size K x R

            H : array-like
                Nonnegative matrix of size R x N

            V_approx : array-like
                Nonnegative matrix W*H of size K x N

            V_approx_err : float
                Error between V and V_approx

            H_W_error : array-like
                History of errors of subsequent H and W matrices

            loss_values : array-like
                History of squared Euclidean distance loss values

            H_W_learning_rates : array-like
                History of learning rates for H and W

            H_W_grad_sign_changes : array-like
                History of update sign changes of subsequent H and W matrices

            H_W_grad_above : array-like
                History of updates above given thresholds of subsequent H and W matrices

            V_error : array-like
                History of errors of subsequent V_approx errors
    """


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,
                        W_D_init=None, W_E_init=None, optimizer='rmsprop', learning_rate=0.001,
                        compute_cauchy=True, save_checkpoints=False):
    """Shallow autoencoder with structured dropout layer (based on code by Suárez), following the proposed
       architecture in [1].
```

```
References
----------
    [1] 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 [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.

    epochs: int
        Number of epochs for training.

    W_D_init: array_like
        Initialization matrix for decoder weights

    W_E_init: array-like
        Initialization matrix for encoder weights

    optimizer: str
        Optimizer to be used for the network

    learning_rate: float >= 0
        Learning rate for the opimizer

    compute_cauchy: bool
```

```
                When True, calculates Cauchy errors for the different matrices

            save_checkpoints: bool
                When True, saves and keeps model checkpoints after each epoch

        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.

            W_E_error: array-like
                History of errors of subsequent W_E matrices

            H_W_error: array-like
                History of errors of subsequent H and W_D matrices

            V_hat_error: array-like
                History of errors of subsequent V_hat matrices
    """


def nae_multiplicative_update(V, R, num_epochs, W_E_init=None, W_D_init=None, H_const=None, mult_enc=True,
                              mult_dec=True, learning_rates=[0.01, 0.1]):
    """Shallow autoencoder with structured dropout layer, own implementation including multiplicative
       updates.

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

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

            num_epochs: int
                Number of epochs for training.

            W_E_init: array_like
                Initialization matrix for encoder weights, zero entries are constrained to stay zero

            W_D_init: array-like
                Initialization matrix for decoder weights, zero entries are constrained to stay zero
        H_const: array-like
            Constraints for the activations, given as the binary mask for the structured dropout layer

        mult_enc: bool
            When True, enables multiplicative updates for the encoder

        mult_dec: bool
            When True, enables multiplicative updates for the decoder

        learning_rates: array-like
            List or array of length 2, containing individual learning rates for encoder and decoder
            for additive updates

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

        W_E: array-like
            Learned encoder weight matrix

        W_D: array-like
            Learned decoder weight matrix

        H_masked: array-like
            Learned code matrix

        losses: array-like
            History of squared Euclidean distance loss values

        cauchy_errors_W: array-like
            History of errors of subsequent W_E and W_D matrices

        cauchy_error_H: array-like
            History of errors of subsequent H matrices

        cauchy_error_V: array-like
            History of errors of subsequent V_hat matrices
"""
```

# Bibliography

[1] O. Berné, C. Joblin, Y. Deville, P. Pilleri, J. Pety, D. Teyssier, M. Gerin, and A. Fuente, *Blind decomposition of Herschel-HIFI spectral maps of the NGC 7023 nebula*, in Proceedings of the Annual meeting of the French Society of Astronomy and Astrophysics (SF2A), Dec. 2012, pp. 507–512.

[2] F. Camastra and A. Vinciarelli, *Machine Learning for Audio, Image and Video Analysis*, Springer London, London, England, 2015.

[3] K. Choi, G. Fazekas, K. Cho, and M. Sandler, *A tutorial on deep learning for music information retrieval.* arXiv preprint arXiv:1709.04396, 2017.

[4] K. Choi, G. Fazekas, and M. B. Sandler, *Automatic tagging using deep convolutional neural networks*, in Proceedings of the International Society for Music Information Retrieval Conference (ISMIR), New York City, USA, 2016, pp. 805–811.

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

[6] A. Cichocki, R. Zdunek, and S. Amari, *New algorithms for non-negative matrix factorization in applications to blind source separation*, in IEEE International Conference on Acoustics Speech and Signal Processing Proceedings, vol. 5, 2006, pp. 621–624.

[7] 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.

[8] 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.

[9] 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.

[10] 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.

[11] C. Févotte and J. Idier, *Algorithms for nonnegative matrix factorization with the beta-divergence*, Neural Computation, 23 (2011), pp. 2421–2456.

[12] Y. Gong, D. Cui, G. Yu, and Xiong Jianbin, *An improved image watermarking algorithm based on NMF and DWT*, in International Conference on Information and Network Security (ICINS), 2014, pp. 6–11.

[13] S. Gupta, K. M. Bhurchandi, and A. G. Keskar, *An efficient noise-robust automatic speech recognition system using artificial neural networks*, in International Conference on Communication and Signal Processing (ICCSP), 2016, pp. 1873–1877.

[14] C. R. Harris et al., *Array programming with NumPy*, Nature, 585 (2020), pp. 357–362.

[15] J. Hongjiao, *Application of advanced BP neural network in image recognition*, in 18th International Symposium on Distributed Computing and Applications for Business Engineering and Science (DCABES), 2019, pp. 17–20.

[16] R. Kelz, M. Dorfer, F. Korzeniowski, S. Böck, A. Arzt, and G. Widmer, *On the potential of simple framewise approaches to piano transcription*, in Proceedings of the International Society for Music Information Retrieval Conference (ISMIR), New York City, USA, 2016, pp. 475–481.

[17] M. Kubat, *An Introduction to Machine Learning*, Springer, Cham, Switzerland, 2017.

[18] D. D. Lee and H. S. Seung, *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. Lefèvre, F. Bach, and C. Févotte, *Online algorithms for nonnegative matrix factorization with the Itakura-Saito divergence*, in IEEE Workshop on Applications of Signal Processing to Audio and Acoustics (WASPAA), 2011, pp. 313–316.

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

[21] A. Maier, *Lecture notes in deep learning.* Friedrich-Alexander-Universität Erlangen-Nürnberg, February 2020.

[22] B. McFee and J. P. Bello, *Structured training for large-vocabulary chord recognition*, in Proceedings of the International Society for Music Information Retrieval Conference (ISMIR), Suzhou, China, 2017, pp. 188–194.

[23] B. McFee, C. Raffel, D. Liang, D. P. Ellis, M. McVicar, E. Battenberg, and O. Nieto, *librosa: Audio and music signal analysis in python*, in Proceedings of the 14th python in science conference, 2015, pp. 18–25.

[24] M. Müller, *Fundamentals of Music Processing – Audio, Analysis, Algorithms, Applications*, Springer Verlag, 2015.

[25] M. Müller, V. Konz, W. Bogler, and V. Arifi-Müller, *Saarland music data (SMD)*, in Late-Breaking and Demo Session of the 12th International Conference on Music Information Retrieval (ISMIR), Miami, USA, 2011.

[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] J. Pomerat, A. Segev, and R. Datta, *On neural network activation functions and optimizers in relation to polynomial regression*, in IEEE International Conference on Big Data (Big Data), 2019, pp. 6183–6185.

[28] 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.

[29] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, *Dropout: A simple way to prevent neural networks from overfitting*, Journal of Machine Learning Research, 15 (2014), pp. 1929–1958.

[30] E. A. Suárez Guarnizo, *DNN-based matrix factorization with applications to drum sound decomposition*, Master's thesis, Friedrich-Alexander-Universität Erlangen-Nürnberg, 2020.

[31] C. C. Tan and C. Eswaran, *Reconstruction of handwritten digit images using autoencoder neural networks*, in Canadian Conference on Electrical and Computer Engineering, 2008, pp. 465–470.

[32] 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.

[33] B. Yousefi, C. I. Castanedo, and X. P. V. Maldague, *Measuring heterogeneous thermal patterns in infrared-based diagnostic systems using sparse low-rank matrix approximation: Comparative study*, IEEE Transactions on Instrumentation and Measurement, 70 (2021), pp. 1–9.