Université de Montréal

Non-negative Matrix Decomposition Approaches to Frequency Domain Analysis of Music Audio Signals

par

Sean Wood

Département d'informatique et de recherche opérationnelle

Faculté des arts et des sciences

Mémoire présenté à la Faculté des arts et des sciences en vue de l'obtention du grade de Maître ès sciences (M.Sc.) en informatique

Décembre, 2009

Université de Montréal

Faculté des arts et des sciences

Ce mémoire intitulé:

Non-negative Matrix Decomposition Approaches to Frequency Domain Analysis of Music Audio Signals

présenté par :

Sean Wood

a été évalué par un jury composé des personnes suivantes :

Yoshua Bengio

président-rapporteur

Douglas Eck

directeur de recherche

Jean Meunier

membre du jury

Résumé

On étudie l'application des algorithmes de décomposition matricielles tel que la Factorisation Matricielle Non-négative (FMN), aux représentations fréquentielles de signaux audio musicaux. Ces algorithmes, dirigés par une fonction d'erreur de reconstruction, apprennent un ensemble de fonctions de base et un ensemble de coefficients correspondants qui approximent le signal d'entrée. On compare l'utilisation de trois fonctions d'erreur de reconstruction quand la FMN est appliquée à des gammes monophoniques et harmonisées: moindre carré, divergence Kullback-Leibler, et une mesure de divergence dépendente de la phase, introduite récemment. Des nouvelles méthodes pour interpréter les décompositions résultantes sont présentées et sont comparées aux méthodes utilisées précédemment qui nécessitent des connaissances du domaine acoustique. Finalement, on analyse la capacité de généralisation des fonctions de bases apprises par rapport à trois paramètres musicaux: l'amplitude, la durée et le type d'instrument. Pour ce faire, on introduit deux algorithmes d'étiquetage des fonctions de bases qui performent mieux que l'approche précédente dans la majorité de nos tests, la tâche d'instrument avec audio monophonique étant la seule exception importante.

Mots clés: Apprentissage machine non-supervisé, Apprentissage machine semi-supervisé, Factorisation matricielle non-négative, Encodage parcimonieux, Extraction de l'information musicale, Détection de la hauteur de notes.

Abstract

We study the application of unsupervised matrix decomposition algorithms such as Non-negative Matrix Factorization (NMF) to frequency domain representations of music audio signals. These algorithms, driven by a given reconstruction error function, learn a set of basis functions and a set of corresponding coefficients that approximate the input signal. We compare the use of three reconstruction error functions when NMF is applied to monophonic and harmonized musical scales: least squares, Kullback-Leibler divergence, and a recently introduced "phase-aware" divergence measure. Novel supervised methods for interpreting the resulting decompositions are presented and compared to previously used methods that rely on domain knowledge. Finally, the ability of the learned basis functions to generalize across musical parameter values including note amplitude, note duration and instrument type, are analyzed. To do so, we introduce two basis function labeling algorithms that outperform the previous labeling approach in the majority of our tests, instrument type with monophonic audio being the only notable exception.

Keywords: Unsupervised Machine Learning, Semi-supervised Machine Learning, Non-negative Matrix Factorization, Sparse Coding, Music Information Retrieval, Pitch Detection.

Contents

Résumé **i**

Abstract **ii**

Table of Contents **v**

List of Figures **ix**

List of Symbols **x**

List of Abbreviations **xi**

La Dédicace **xii**

Remerciments **xiii**

1 Introduction **1**

1.1 Introduction 1

1.2 Problem Statement 1

1.3 Motivations 2

1.3.1 Music Information Retrieval 2

1.3.2 Signal Processing 2

1.3.3 Machine Learning 3

1.4 Approach 3

2 Processing Audio Signals **5**

2.1 Processing Audio Signals 5

2.1.1 Discretization 6

2.1.2 Frequency Domain Analysis 6

2.2 Music Audio Signals . . . 8
2.2.1 Pitch . . . 8
2.2.2 Timbre . . . 9
2.2.3 Polyphonic Music . . . 9
2.3 Music Preprocessing . . . 10
2.3.1 Downsampling . . . 10
2.3.2 Window Size . . . 11
2.3.3 Window Step Size . . . 11
2.3.4 Spectrogram Type . . . 12
2.3.5 Log Scaled Frequency . . . 13
2.4 Dataset Generation . . . 13

3 Learning Algorithms **15**
3.1 Unsupervised Algorithms . . . 15
3.1.1 Non-negative Matrix Factorization . . . 15
3.1.2 Non-negative Matrix Factor 2D Deconvolution . . . 19
3.1.3 Coefficient Sparsity . . . 22
3.2 Semi-supervised Algorithms . . . 24
3.3 Applications to Music Audio . . . 24
3.3.1 Spectrogram Factorization . . . 25
3.3.2 Spectrogram Deconvolution . . . 25
3.3.3 Phase-aware Factorization . . . 26

4 Experiments **28**
4.1 Introduction . . . 28
4.2 Cost Functions . . . 29
4.3 Dictionary Initialization . . . 31
4.4 Dictionary Labeling . . . 34
4.5 Dictionary Size . . . 37
4.6 Inference . . . 43
4.7 Musical Parameters . . . 46
4.7.1 Monophonic Data Results . . . 46
4.7.2 Polyphonic Data Results . . . 48

5 Conclusions and Future Work **52**

5.1 Conclusions 52

5.2 Future Work 53

Bibliography **54**

List of Figures

2.1 Conceptual depiction of the decomposition of sound into constituent frequencies by the human ear, similar in nature to a prism that separates light into its constituent frequencies. 5

2.2 Conceptual depiction of discretization in time and amplitude a) source signal: continuous in time and amplitude b) sampled signal: discrete in time, continuous in amplitude c) quantized sampled signal: discrete in time and amplitude. 7

2.3 Downsampling to fractions of the original sample rate (44.1kHz), from top to bottom: 1.0 (44.1 kHz), $\frac{1}{2}$ (22.05 kHz), $\frac{1}{4}$ (11.025 kHz), $\frac{1}{8}$ (5.5125 kHz), $\frac{1}{16}$ (2.75625 kHz). 10

2.4 The effect of various window sizes, from top to bottom, in samples: 2048 (185.8ms), 1024 (92.9ms), 512 (46.4ms), 256 (23.2ms), 128 (11.6ms), 64 (5.8ms). Only the lowest $\frac{1}{8}$ of the frequency bins are shown. 11

2.5 The effect of window step size on temporal resolution, for a window size of 2048, and decreasing step sizes from top to bottom: 2048 (no overlap), 1024, 512, 256. 12

2.6 Different Spectrogram Types: a) magnitude spectrogram $|X_k|$ b) energy spectral density $|X_k|^2$ c) log magnitude spectrogram $\log|X_k|$ 12

2.7 Log-scaled Spectrogram 13

2.8 Dataset Generation Flowchart. Rounded boxes represent processing code, while rectangular boxes represent types of data. 14

3.1 The two basis functions used to generate artificial data for the NMF Example a) Gaussian b) Square of an exponentially damped sinusoid 16

3.2 Basis functions learned by NMF from artificial data: a) learned Gaussian b) learned square of exponentially damped sinusoid 16

3.3 Five examples of the results of the NMF algorithm applied to artificial data described above a) input examples b) input coefficients for the Gaussian (left bar) and squared exponentially decayed sinusoid (right bar) c) corresponding reconstructed inputs d) learned coefficients. Both the input and learned coefficients are normalized to the maximum coefficient value across the entire dataset, and the bounds of their graphs are 0 to 1. 17

3.4 Example application of NMFD↑. The reconstruction of 25 randomly chosen examples are shown. From left to right: the reconstruction matrix $\mathbf{\Lambda}$ (F x T), computed by convolving $\mathbf{W}$ with $\mathbf{H}$, the learned basis matrix $\mathbf{W}$ (F x D), and the learned shift coefficients $\mathbf{H}$ (F x T x D). Since there is only one basis vector, D = 1. The learned basis vector and reconstruction matrices are qualitatively indistinguishable from the true basis vector and input matrix $\mathbf{V}$, respectively. 21

3.5 Example application of NMFD→. From left to right: the reconstruction matrix $\mathbf{\Lambda}$ (F x T), computed by convolving $\mathbf{W}$ with $\mathbf{H}$, the learned basis matrix $\mathbf{W}$ (F x τ), and the learned shift coefficents $\mathbf{H}$ (T). The learned basis matrix and reconstruction matrices are qualitatively indistinguishable from the true basis matrix and input matrix $\mathbf{V}$, respectively. 21

3.6 Example application of NMF2D. From left to right: the reconstruction matrix $\mathbf{\Lambda}$ (F x T), computed by convolving $\mathbf{W}$ with $\mathbf{H}$, the learned basis matrix $\mathbf{W}$ (F x τ), the learned shift coefficents $\mathbf{H}$ (F x T). 22

3.7 Example for which NMFD↑ does not yield sparse decompositions. Note that while the reconstructions in $\mathbf{\Lambda}$ (F x T) are accurate, the basis functions have been distributed across $\mathbf{W}$ (F) and $\mathbf{H}$ (F x T). 23

3.8 Same example as 3.7, however with the addition an l_1 sparsity constraint, which allows us to find a "correct" minimum. The basis functions are no longer distributed across $\mathbf{W}$ and $\mathbf{H}$, and $\mathbf{H}$ is correctly sparse. 23

3.9 Semi-supervised NMF: NMF unsupervised training, followed by logistic regression supervised training. Note that the parameters are not shared between the two phases. 25

4.1 Legend for Figures 4.2 and 4.10. 29

4.2 Training error curves using different error functions, with a dictionary size of 25. 3 x 3 grid showing minimization of each error (rows) by each algorithm (columns). Vertical axis represents the cost of the corresponding column, labeled horizontal axis is number of epochs. The histograms at the left of each graph show the trajectories' final values (lower-left histogram is omitted since the majority of trajectories diverged). See Figure 4.1 for a legend of the instruments used. 30

4.3 Dictionary Initializations: while different initializations yield similar reconstruction matrices $\mathbf{\Lambda}$ (F x T), the ordering of the dictionary matrices $\mathbf{W}$ (F x D) and coefficient matrices $\mathbf{H}$ (D x T) are completely different. 32

4.4 a) Randomly initialized dictionary b) Harmonically initialized dictionary 33

4.5 Harmonic Dictionary Initialization. The reconstruction matrices $\mathbf{\Lambda}$ (F x T) remain accurate as before, however now the dictionary matrices $\mathbf{W}$ (F x D) and coefficient matrices $\mathbf{H}$ (D x T) are qualitatively very similar. The general structure of the ascending scale is evident in each coefficient matrix. 33

4.6 Example of hard labeling of dictionary elements using max correlation with initialized dictionary. a) Dictionary matrix b) Ideal dictionary matrix c) resulting similarity matrix 35

4.7 Example of hard labeling of dictionary elements using max correlation with ground truth, with activation threshold percentage of 0.01, for a threshold value 0.077. 36

4.8 Example of soft labeling using logistic regression. a) weights normalized such that smallest weight is white, largest weight is black. b) same as in a), but with negative weights set to zero. 36

4.9 Activations of different labeling algorithms a) dictionary matrix b) coefficient matrix c) hard dictionary labeling activations d) hard coefficient labeling activations e) logistic activations . 37

4.10 Evolution of cost over the course of optimization for various dictionary sizes. Columns for each cost function, vertical axes represents appropriate cost. Indexed horizontal axis represents epoch index. Dictionary size indicated on the left. See Figure 4.1 for a legend of the instruments used. 39

4.11 Final error histograms for the three optimizations. Note that the scale for $D = 100$ is enlarged to show differences between instruments. 40

4.12 Example showing dictionary interpretability on monophonic data, for different dictionary sizes and each cost function. Input consists of an ascending single-octave A major scale, played by the Grand Piano instrument. 41

4.13 Example showing dictionary interpretability on polyphonic data, for different dictionary sizes and each cost function. Input consists of an ascending single-octave harmonized A major scale (see figure 4.20), played by the Grand Piano instrument. 42
4.14 Inference results for each cost function, using the two inference techniques (pseudo inverse and NMF). Left column contains the ideal coefficients. 44
4.15 Comparison of squared errors between the ideal coefficients and inferred coefficients for each cost function. The left columns depict the ideal coefficient matrices **H**, while the center and right columns depict the reconstructed coefficient matrices using the pseudo-inverse and NMF methods respectively. 45
4.16 Musical notation of monophonic data used in section 4.7.1. 46
4.17 Mean F-score ± standard error values for monophonic velocity task, for the different NMF algorithms, and different labeling algorithms. 48
4.18 Mean F-score ± standard error values for monophonic duration task, for the different NMF algorithms, and different labeling algorithms. 49
4.19 F-score values for instrument task, for the different NMF algorithms, and different labeling algorithms. 49
4.20 Musical notation of polyphonic data used in section 4.7.2. 50
4.21 Mean F-score ± standard error values for polyphonic velocity task, for the different NMF algorithms, and different labeling algorithms. 50
4.22 Mean F-score ± standard error values for polyphonic duration task, for the different NMF algorithms, and different labeling algorithms. 51
4.23 F-score values for polyphonic instrument task, for the different NMF algorithms, and different labeling algorithms. 51

List of Symbols

- A440 : reference frequency for musical tones: 440Hz
- f_s : sample rate
- T_s : sampling period
- n : sample index
- F0 : fundamental frequency
- t : time
- p : partial index of a complex tone
- α_p: amplitude of the p^{th} partial of a complex tone
- θ_p : phase of p^{th} partial of a complex tone
- η : learning rate
- $\mathbf{V}$: input matrix
- $\mathbf{\Lambda}$: reconstructed input matrix
- $\mathbf{W}$: dictionary matrix
- $\mathbf{H}$: coefficient matrix

List of Abbreviations

- DFT : Discrete Fourier Transform
- MIDI : Musical Instrument Digital Interface
- MIR : Music Information Retrieval
- NMF : Non-negative Matrix Factorization
- PCM : Pulse-Code Modulation
- STFT: Short-Time Fourier Transform

La Dédicace

I dedicate this thesis to Sam T. Roweis, whose continued guidance, encouragement, and support have proven essential in my life.

Remerciements

J'aimerais remercier les professeurs du laboratoire LISA (Yoshua Bengio, Douglas Eck et Pascal Vincent), pour m'avoir fourni un environnement dans lequel je me suis senti libre d'explorer. Ceci fut un élément très important dans mes développements académiques ainsi que personnels. J'aimerais aussi exprimer mon appréciation profonde en vers ma famille et mes amis, tout simplement pour avoir partagé ces moments de ma vie avec moi.

Chapter 1

Introduction

1.1 Introduction

Music audio signals may be seen as a set of events generated by musical instruments, or more generally, audio sources. Musical events have many factors of variation such as amplitude, duration, pitch and timbre. In the case of *monophonic* music, events do not overlap in time, whereas in *polyphonic* music they do. We will study a family of unsupervised machine learning algorithms including Non-negative Matrix Factorization (NMF) that have been shown to extract amplitude, pitch and timbre information from monophonic and polyphonic music audio signals. While music has features at many different timescales, these particular features are found at a timescale on the order of 20 ms, i.e. longer than transient note onsets and shorter than note durations and higher level musical structure information.

In this chapter, we first define the problem we will study, followed by the motivations behind the problem. We then present motivations for the use of signal processing and machine learning to solve the problem, and finish with the approach we use to study this approach to the problem. The remainder of the thesis is organized as follows. In chapter 2, we present the data we will use as well as an analysis of preprocessing techniques. In chapter 3, we present the machine learning algorithms we will study. The experiments themselves are presented in chapter 4, and finally chapter 5 presents a discussion of future work and conclusions.

1.2 Problem Statement

We study the application of unsupervised matrix decomposition algorithms such as NMF to frequency domain representations of music audio signals. We are interested in determining the kinds of information that may be extracted by such algorithms using different algorithm variants and parameters settings. Finally, we are

interested in how this information may be used to determine the fundamental frequencies of notes in unseen music audio.

1.3 Motivations

1.3.1 Music Information Retrieval

Music Information Retrieval (MIR) is an active area of research that involves the extraction of information from raw data such as music audio signals and musical scores. We focus on a sub-field of MIR involving the extraction of musical information from audio signals. The extraction of musical features such as amplitude, pitch and timbre are important subproblems for many MIR tasks such as automatic transcription and instrument identification. As personal and commercial music databases continue to grow in size, the ability to easily navigate these databases is becoming increasingly difficult. Music recommendation engines are growing in popularity, however there is significant potential for improvement by incorporating musical similarity measures, as musical features of the audio such as melody and harmony provide emotional content to listeners.

The extraction of amplitude, pitch and timbre information is an important subproblem for more complex problems such as analysis of harmony and longer timescale structure analysis. Chord detection depends both on the notes present in a given frame of audio as well as the musical context, for example while a C6 chord and an Am7 chord contain the same pitch classes, the musical context will suggest one label over the other. Finally, although we will study the extraction of information from music audio signals, there are important applications to speech audio signals including source separation.

1.3.2 Signal Processing

Signal processing is a field of engineering and applied math that involves the processing and analysis of signals such as functions of time, e.g. sound, and of space, e.g. images. We will make use of signal processing algorithms to transform the raw audio data into a representation that will be used as input to the learning algorithms. This preprocessing step is an important step in the learning process. First, the preprocessing algorithms allow us to provide the learning algorithms with a representation of the audio in which the features relevant to our tasks are made apparent. Without this transformation, the learning algorithms would be required to learn such transformations themselves, increasing their required complexity, as well as the difficulty of the learning task. Second, the preprocessing algorithms may allow us to significantly reduce the dimensionality of the input data, without losing too much information relevant to the learning tasks.

1.3.3 Machine Learning

Machine learning is an interdisciplinary area of research related to artificial intelligence that involves probability theory, statistics and optimization. Central to machine learning is the development of learning algorithms that provide an output function based on input data. Typically, one defines a (possibly infinite) family of functions through which the learning algorithm must search to find the optimal algorithm, based on a given evaluation metric. The evaluation metric encourages the learning algorithms to find output functions that generalize well, i.e. that perform well on data unseen by the algorithm. This differs from a memorization kind of learning, in which the ability to generalize is not required. Finally, machine learning differs from classical artificial intelligence in that the intelligence is learned by the algorithms, as opposed to being solely provided by the designer.

There are several motivations for the use of machine learning for the analysis of music audio data. First is the need for robustness: the space of music audio data is infinite, yet we wish to find an algorithm capable of performing reliably anywhere in this space. Machine learning is well suited to this problem as we may train an algorithm on a small subset of this space, with the goal that it generalizes well to the rest. Second, music audio is very high-dimensional, e.g. audio is encoded on a standard audio CD at 16 bits per sample, with 44 100 samples per second. An average stereo song three minutes in length contains approximately 30 MB of data. The machine learning algorithms we will use are adept at finding patterns in high dimensional data such as this. Another advantage of the machine learning approach is that we may choose a family of models that are well suited to the problem at hand. Indeed, we will make certain assumptions about the input data that allow us to use a relatively simple family of models to solve the problems we will be interested in.

1.4 Approach

To accomplish the goals laid out in the problem statement, we will run two general types of experiments. The first type are primarily qualitative in nature, and will focus on the factors of variations of the algorithms, the goal being to determine how the different factors affect the information extracted from the audio. For example, we will investigate the use of different cost functions and hyperparameter values. The second type of experiments is quantitative in nature, and focus on variations in the data. For these experiments, the algorithm is to determine the pitch of the notes present in a given frame of audio. We will compare the ability of the different algorithms to generalize across three factors of variation: amplitude, duration and instrument. For example, we study the ability of the various algorithms trained on a given set of instruments to perform pitch detection on another set of instruments.

For all the experiments, we will use simple musical datasets: monophonic and harmonized scales played

by a single instrument. A monophonic scale is simply an ordered set of notes, played ascending in our case. A harmonized scale is polyphonic, and consists of chords instead of single notes, and will also be played in ascending order. The organized nature of these scales, i.e. uniform note duration and velocity, and progressively increasing pitch, will prove to be very useful in analyzing the resulting decompositions. For the quantitative experiments, it will be necessary to have a low-dimensional description of the musical events in the input signals. This information will be used both for training (in the case of the supervised algorithms), as well as testing. A standard digital format for organizing such information is MIDI (Musical Instrument Digital Interface), which contains amplitude, duration and pitch information of the constituent notes.

We will automate the generation of audio datasets using the Kontakt sampler software instrument [5] and MIDI representations of the datasets. Since the Kontakt sampler uses recordings of real acoustic instruments, the resulting audio signals are very realistic and high quality. Once the audio has been generated, it is preprocessed using signal processing techniques described in chapter 2. The preprocessed audio is then used as input to the unsupervised machine learning algorithms. In the case of the supervised algorithms, the ground truth information is also provided as input.

Chapter 2

Processing Audio Signals

2.1 Processing Audio Signals

Audio signals are representations of sound that exist in many different forms such as voltage variations in a microphone cable or sequences of bits in an digital audio file. According to the Fourier Theorem, such signals may be expressed as a sum of sinusoids of different amplitudes, frequencies and phases. The human ear is sensitive to frequencies in the range of approximately 20Hz to 20kHz, i.e. the *acoustic* range of sound, and an important function thereof is to separate sound into its constituent frequencies.

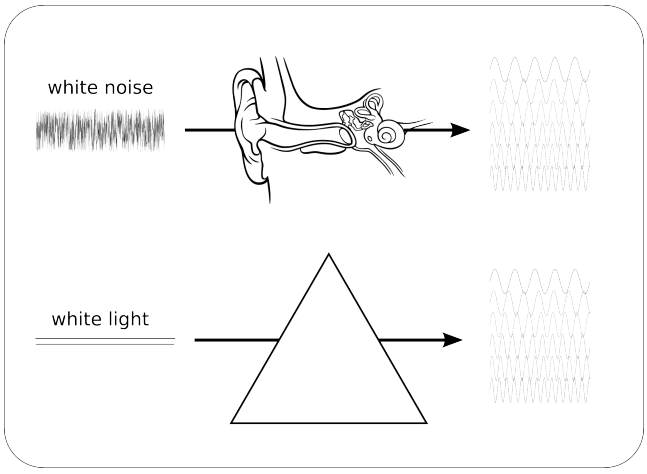

Figure 2.1: Conceptual depiction of the decomposition of sound into constituent frequencies by the human ear, similar in nature to a prism that separates light into its constituent frequencies.

In order to process audio signals using a digital computer, they must first be discretized in both time and amplitude. If we wish to analyze the signal's frequency content, we then transform the digital signal from the time domain into the frequency domain. We will start by discussing the process of discretization,

followed by the short-time Fourier transform (STFT), and analysis in the frequency domain. We will then discuss important features of music audio signals, and finish with a qualitative analysis of the effect of the parameters of the various preprocessing stages.

2.1.1 Discretization

Discretization in time is typically done at uniform time intervals, called the sampling period or T_s. The sample rate, or sampling frequency, is then defined as $f_s = T_s^{-1}$. A sampled version of a continuous function $f(t)$ can be written as $x_n = f(nT_s)$, where n is the sample index. Music audio signals are typically sampled with a sample rate of 44.1 kHz, however higher sample rates (48, 96 or 192 kHz) may be used in professional recording settings, and lower sample rates (8 kHz) may be used in digital telephones. The choice of sampling frequency is important for the preservation of the signal's information after the sampling process. The Nyquist-Shannon sampling theorem [33] states that a signal may be perfectly reconstructed from its sampled version if and only if the sampling frequency is at least two times the highest frequency present in the signal. Any frequencies above the *Nyquist frequency*, equal to half of the sample rate, will result in aliasing during the sampling process. Aliasing is an effect in which those frequencies higher than the Nyquist frequency show up as lower frequencies in the sampled signal, and is typically undesirable.

Discretization in amplitude is typically done using the pulse-code modulation (PCM) representation. Each sample is quantized to one of 2^{B} possible values, where B is typically 16 or 24 in the case of music audio signals. The quantized signal takes on values $\in \left[-2^{\mathrm{B}-2}, 2^{\mathrm{B}-2}\right] \epsilon\, \mathbb{N}$, which may be mapped onto $[-1, 1] \epsilon\, \mathbb{R}$ for processing. The bit depth effects the dynamic range and the signal to noise ratio of the encoded signal. The audio signals we will be analyzing have a sample rate of 44.1 kHz and a bit depth of 16.

2.1.2 Frequency Domain Analysis

2.1.2.1 Fourier Transform

We may convert a finite duration audio signal from the time domain into the frequency domain using the Discrete Fourier Transform (DFT). The DFT expresses the time domain signal as a sum of sinusoids of different frequencies, each with a corresponding amplitude and phase. The DFT of a signal x_n is defined as $\mathrm{X}_k = \sum_{n=0}^{N-1} x_n e^{\frac{2\pi i}{N} kn}$ for $k = 0, 1, ..., N-1$. Since $e^{i\theta} = \cos(\theta) + i\sin(\theta)$, the real and imaginary parts of each term in the sum correspond to a dot product of the signal with a sine and cosine of frequency $\frac{k}{N}$ times the Nyquist frequency, respectively. Thus, X_k is a complex-valued scalar, where $|X_k|^2$ corresponds to the amount of energy at frequency $\frac{k}{N}$ times the Nyquist frequency, and $\tan^{-1}\left(\frac{Im(X_k)}{Re(X_k)}\right)$ corresponds to its phase.

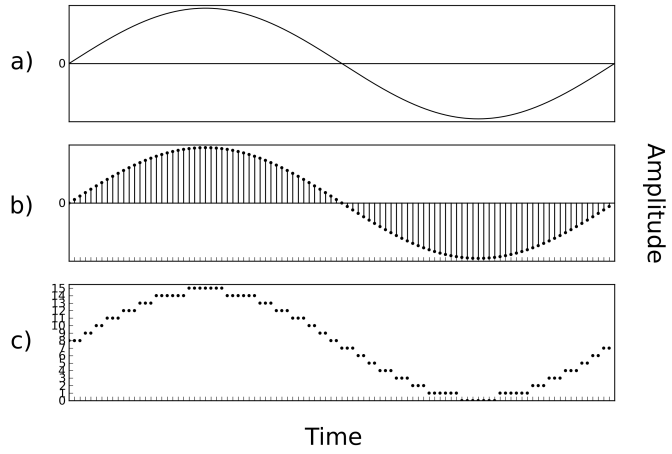

Figure 2.2: Conceptual depiction of discretization in time and amplitude a) source signal: continuous in time and amplitude b) sampled signal: discrete in time, continuous in amplitude c) quantized sampled signal: discrete in time and amplitude.

2.1.2.2 Windowing

Application of the DFT to a finite duration signal implies an assumption that the signal represents one period of an infinite duration signal. As this is not the case in most situations, we consider the finite duration signal to be a windowed version of an infinite duration signal, where the windowing function is a unit height boxcar function, $boxcar\,(x) = \{1\,for\,0 < n < N,\,0\,otherwise$. Other windowing functions may be used, and the choice affects the resulting transformation due to a phenomenon called spectral leakage. As multiplication in the time domain is equivalent to convolution in the frequency domain, the DFT results in the convolution of the signal with the windowing function in the frequency domain. In the general case, this has the effect of smearing the energy of a given frequency across multiple frequencies. The shape of the windowing function is therefore important for the interpretability of the transformation of the time domain signal.

Taking the DFT of a non-stationary signal tells us which frequencies are present over the course of the signal, but does not tell us when the frequencies occur. If we are interested in how the frequency content varies with time, we may implement a sliding window approach, in which we isolate small portions of the audio signal that are then independently transformed into the frequency domain. This process is referred to as the short-time Fourier transform (STFT), and in the case of audio is referred to as a spectrogram. The choice of window size involves a trade-off between temporal and spectral resolution, as temporal resolution decreases with increasing window size, while spectral resolution increases and vice versa. With a fixed sample rate, the area of the time-frequency tiles is fixed, and the choice of window size determines the temporal and spectral resolutions.

2.1.2.3 Alternative Time-Frequency Transforms

There are several alternatives to the STFT for generating a time-frequency representation of a time-domain signal. While we will use the STFT for our experiments, we briefly mention a few others here for the sake of completeness. Wavelet transforms express the input signal in terms of scaled versions of a mother wavelet function, and allows for a frequency-dependent tiling of the time-frequency plane removing many of the problems associated with fixed window size Fourier transforms [17]. The reassignment method is a technique for sharpening spectrograms by remapping the data, yielding a much sparser and less blurry time-frequency transform [24].

2.2 Music Audio Signals

Music audio signals have several salient features at different time scales. These signals are typically composed of events, referred to as notes. A common model of the production of notes is the Attack-Decay-Sustain-Release (ADSR) model. A note's onset, or transient attack, occurs at a very short time scale: the pluck of a guitar string or the hammering of piano strings. Attacks are an important feature used in distinguishing between instruments. After the note onset, the note's amplitude quickly decays until it enters a quasi-periodic phase in which the amplitude and frequency content remains relatively stationary. This phase accounts for the majority of the duration of the note, and will be our primary focus. The final phase is the release which ends with the note having zero amplitude, referred to as the note's offset. There exist several other musical features including the note durations and higher-level musical structure such as melodies and chord progressions.

As mentioned above, we will focus on the sustain phase of notes. During this phase, there are three salient features: amplitude, pitch and timbre. Amplitude is a relatively simple feature that describes how loud the note is. Pitch and timbre are more subtle, and are described in the following sections. Finally, we will discuss subtleties that arise in polyphonic music, when more than one note are played at the same time.

2.2.1 Pitch

Pitch plays an important role in vocal communication and lies at the heart of music. Most musical instruments produce complex tones containing energy at positive integer multiples of a fundamental frequency, F0. The different frequency components may be referred to as partials, and their sum can be written mathematically as $\sum_p \alpha_p \sin(p\text{F0}t + \theta_p)$ where p is the partial index, and α_p and θ_p are the amplitude and phase of the p^{th} partial. Pitch can be defined as the perceived fundamental frequency of such a tone. In Western music, the

set of allowable fundamental frequencies is defined as equal-tempered pitch, in which each octave (doubling of frequency), is divided into twelve equally spaced pitches. The fundamental frequency of a given tone is then $440 \cdot 2^{\frac{a}{12}}$ where a is the integer distance between the note and the A440 reference tone. Many algorithms have been designed to determine the fundamental frequency of monophonic audio such as a voice or an instrument. For a good review, see [25]. YIN [18] is a recently introduced state of the art algorithm that works using the autocorrelation and difference functions to determine the most likely periods of repetition over the course of the signal.

2.2.2 Timbre

Timbre can be defined as the aspect of a note that is not pitch, amplitude or duration. Intuitively, timbre describes the quality of the sound: that which makes two instruments playing the same pitch at the same loudness sound different. In speech, different vowels spoken with the same pitch and intensity, may be said to have different timbres. An important aspect of a note's timbre is its spectral envelope, which can be deduced from the relative amplitudes of its partials. One algorithm that may be used to extract this information is the Linear Predictive Coding (LPC) algorithm.

2.2.3 Polyphonic Music

Amplitude, duration, pitch and timbre are features we can attribute to a single note. When multiple notes are played simultaneously, however, it becomes very difficult to determine the features of the individual notes. First, we are no longer able to determine to which note a particular frequency component belongs. Second, it is common for notes played simultaneously to have small integer ratios between their fundamental frequencies, resulting in multiple notes contributing energies at identical frequencies, due to overlapping partials. Since we cannot know a priori how much energy will come from each note, this makes the timbre detection problem significantly more difficult than the monophonic case. Finally, depending on the phase relationships between two overlapping partials, there will be a varying degree of destructive interference between them. In the worst case, if two overlapping partials of equal energy have a phase difference of precisely π radians, the components will completely cancel each other out. In practice, frequencies contributed by different sources are rarely identical, and the phase differences are not stationary over the notes' durations, resulting in a varying amount of interference perceived as beating in the resulting waveform. This problem will be discussed further in section 3.3.3, where we discuss work that has been done to incorporate this knowledge into matrix factorization algorithms.

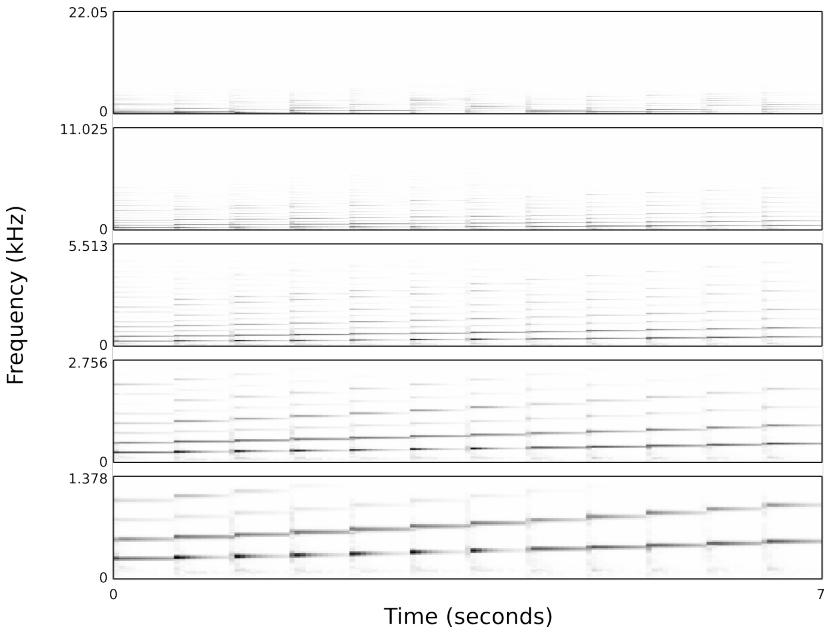

Figure 2.3: Downsampling to fractions of the original sample rate (44.1kHz), from top to bottom: 1.0 (44.1 kHz), $\frac{1}{2}$ (22.05 kHz), $\frac{1}{4}$ (11.025 kHz), $\frac{1}{8}$ (5.5125 kHz), $\frac{1}{16}$ (2.75625 kHz).

2.3 Music Preprocessing

We will now analyze the effect of the parameters of the various preprocessing stages involved in converting the raw audio into a representation suitable as input to the machine learning algorithms described in the following chapter.

2.3.1 Downsampling

In the music we will be analyzing, the information of interest lies in the lower portion of the frequency space. Therefore, to focus on the relevant region of the frequency space, we choose to downsample the audio signal prior to conversion to the frequency domain. This downsampling is done using the libsamplerate library [6]. Prior to downsampling, the library filters out frequencies higher than the Nyquist frequency of the new sampling rate using a low-pass filter. Figure 2.3 demonstrates the effect of downsampling for different fractions of the original sample rate, 44.1kHz. We note that information is lost when the new sample rate is less than one fourth of the original sample rate, therefore we will downsample by a factor of four, resulting in a new sample rate of 11.025 kHz. Therefore, we will only be concerned with frequencies in the data less than 5.5125 kHz.

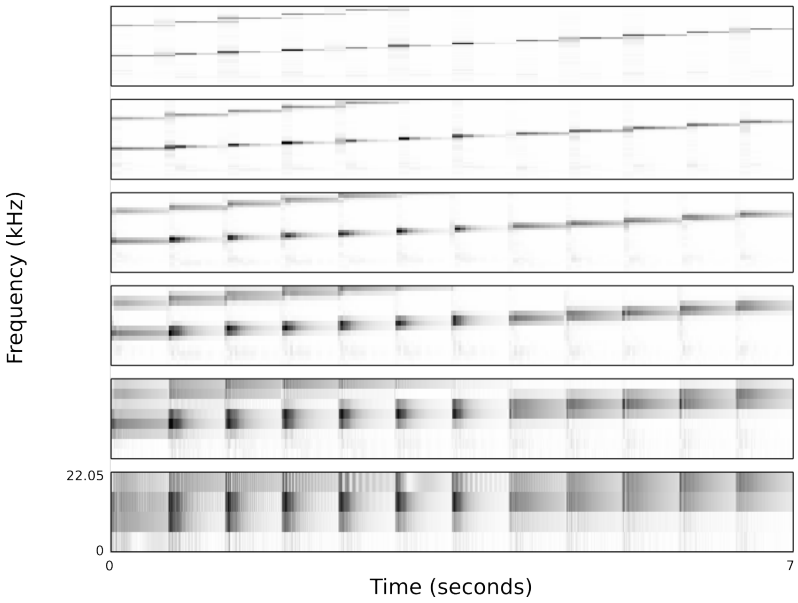

Figure 2.4: The effect of various window sizes, from top to bottom, in samples: 2048 (185.8ms), 1024 (92.9ms), 512 (46.4ms), 256 (23.2ms), 128 (11.6ms), 64 (5.8ms). Only the lowest $\frac{1}{8}$ of the frequency bins are shown.

2.3.2 Window Size

For a given sample rate, 11.025 kHz in this case, the window size determines the spectral and temporal resolutions of the frequency transform. The number of frequency bins between 0Hz and the Nyquist frequency is equal to half of the window size. There is a trade-off between temporal and frequency resolutions, the larger the window size the poorer the temporal resolution and the greater the frequency resolution, and vice versa. In figure 2.4 we demonstrate the effect of different window sizes, with no window overlap. As can be seen, as the precision of the frequency of the notes decreases as the window size decreases, while the precision of the note onsets as well as the temporal progression of the notes increases. We will use a window size of 512 samples, or 46.4ms, as for our tasks, frequency resolution will be more important than temporal resolution.

2.3.3 Window Step Size

We may improve effective temporal resolution by overlapping the sliding windows. We will define the amount of overlap as step size, the distance in samples between the start of successive windows. No overlap therefore implies a step size equal to the window size. Decreasing the step size results in an increase in the number of columns, i.e. time steps, in the resulting spectrogram. Below, we demonstrate how decreasing the step size results in a higher effective temporal resolution, where we have exaggerated the poor temporal resolution for demonstration purposes with a window size of 2048 samples, or 185.8ms. With increasing step size, we see that the onset times and temporal progressions of each note become more clear.

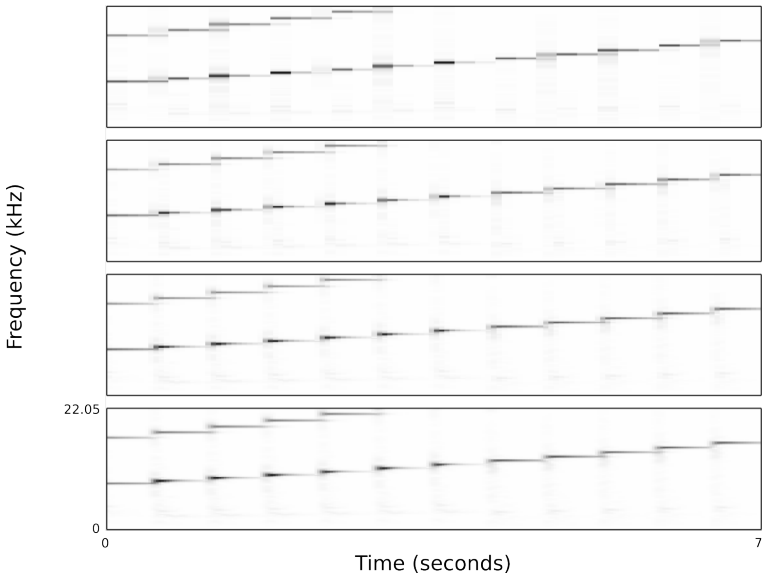

Figure 2.5: The effect of window step size on temporal resolution, for a window size of 2048, and decreasing step sizes from top to bottom: 2048 (no overlap), 1024, 512, 256.

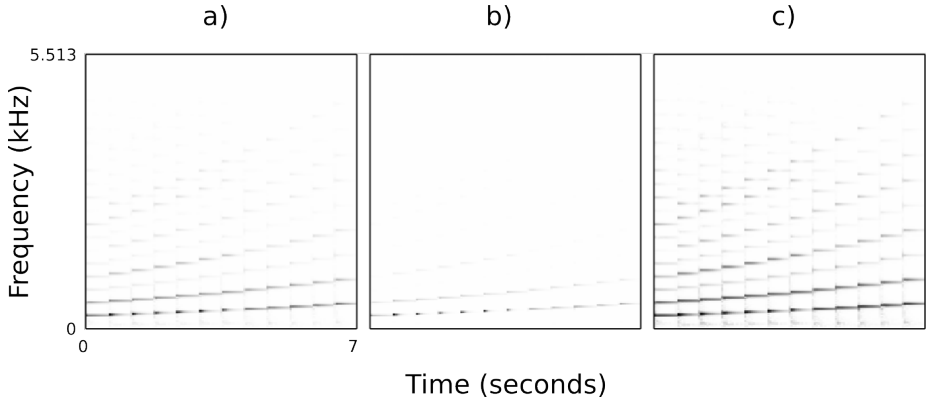

Figure 2.6: Different Spectrogram Types: a) magnitude spectrogram $|X_k|$ b) energy spectral density $|X_k|^2$ c) log magnitude spectrogram $\log|X_k|$

2.3.4 Spectrogram Type

To generate the spectrograms shown above, we took the magnitude of the Fourier transform $|X_k|$, i.e. a magnitude spectrogram. Other types are possible, including the energy spectral density $|X_k|^2$, representing the amount of energy at each time frequency bin, and the log magnitude spectrogram, $\log|X_k|$. As can be seen in figure 2.6, the log magnitude spectrogram brings out the higher frequency partials of the notes. Each spectrogram type has a different interpretation when used as input to the learning algorithms in the following chapter.

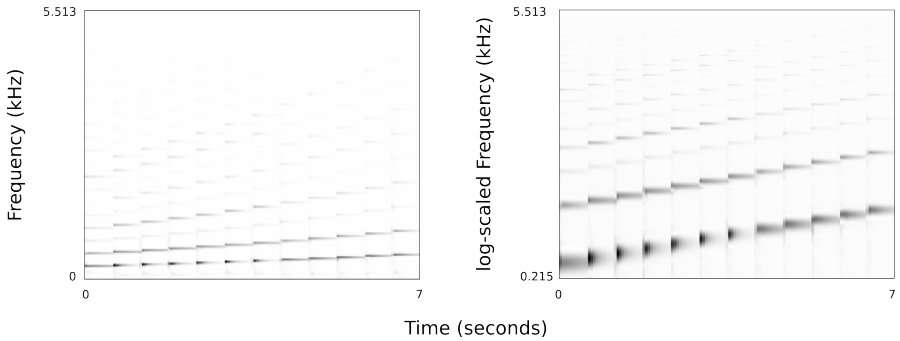

Figure 2.7: Log-scaled Spectrogram

2.3.5 Log Scaled Frequency

Due to the fact that a note's partial frequencies are positive integer multiples of the fundamental frequency, the distance in frequency between partials increases with fundamental frequency. As will be discussed in section 3.3.2, we may wish to impose shift-invariance on the frequency axis, which requires that the spacing between partials remain fixed. To address this issue, we may convert the linear-frequency transform returned by the Fourier transform into log-frequency using the Constant-Q transform [16]. The Constant-Q transform returns energy in bins whose ratios between bandwidth and center frequency are constant, and can be implemented by mapping the Fourier bins logarithmically. We make use of a version of Dan Ellis' matlab code [3], ported to Python. Figure 2.7 demonstrates the effect of this rescaling. We may note that in the log-frequency spectrogram, the distance between harmonics remains fixed as the fundamental frequency changes, while in the standard spectrogram the distance increases. While this scaling is important if we desire shift invariance on the frequency axis, it is unnecessary otherwise, and therefore will be avoided.

2.4 Dataset Generation

Dataset generation is automated in order to facilitate the creation of large quantities of audio data, and to provide a mechanism to easily generate datasets with desired properties. For example, we may wish to generate audio files from a given piece of music using many different instruments, or generate audio using a simple scale but with many different note amplitudes. In doing so, we may isolate specific characteristics of musical data, facilitating analysis of algorithms' ability to generalize over these characteristics. It would not be feasible to achieve the quantity and diversity of these datasets if the system were not automated. An overview of the data generation process can be seen in figure 2.8.

The MIDI used for dataset generation is generated using the Python programming language [10] and an in-house Python library named pygmy, which provides facilities for manipulation of MIDI, and allows

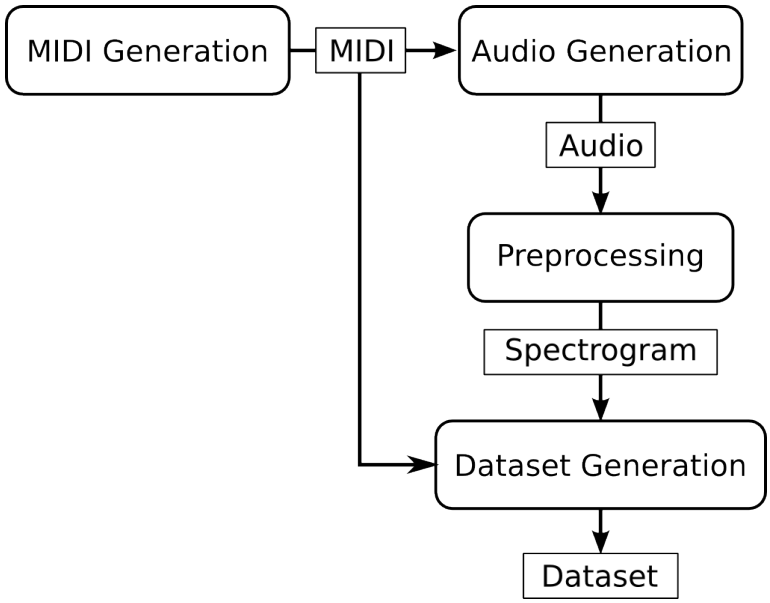

Figure 2.8: Dataset Generation Flowchart. Rounded boxes represent processing code, while rectangular boxes represent types of data.

for easy generation of monophonic and harmonized scales. Alternatively, MIDI may be acquired from other sources such as online MIDI databases. The MIDI is then converted to audio using a custom C++ tool named midi2audio, which makes use of the Core Audio library[2] and the Kontakt Audio Unit [5]. While we generate our data with Kontakt Audio Unit, the system is flexible and may use any virtual instrument with an Audio Unit [1], or SoundFont [12] interface.

Audio files are preprocessed as described above using Python code that make use of the following libraries: numpy [8], scipy [11], libsndfile [7] and libsamplerate [6]. Results are stored in HDF5 [4] data files using pytables [9].

Finally, we generate the datasets that will be used with the machine learning algorithms by generating input and ground truth matrices, with lists of corresponding train, valid and test set indexes. The input matrices correspond to spectrograms, while the ground truth matrices contain velocity and pitch information for each spectrogram frame. To apply a learning algorithm to a dataset, an input (and possibly ground truth) matrix is first read into memory, then the relevant subsets are isolated using the indexes of interest to the task.

Chapter 3

Learning Algorithms

3.1 Unsupervised Algorithms

In unsupervised learning tasks, learning is driven entirely by the input data. This is in contrast with supervised learning tasks, in which each input example has an associated target value which the algorithm learns to predict. The goal in an unsupervised learning task is to learn a model of the input data which can be used for dimensionality reduction (compression), clustering, interpretation, etc. We introduce here a family of unsupervised algorithms that approximate input data as linear combinations of a set of basis functions which may be fixed or learned. We will focus on algorithms that impose non-negativity constraints on both the basis functions and coefficients. We will discuss sparsity constraints that may be imposed on the coefficients in place of, or in addition to, the non-negativity constraints. Sparsity constraints favor sparse decompositions in which input examples may be reconstructed using only few basis functions. Depending on the data, however, sparse decompositions may be achieved using the non-negativity constraint alone.

3.1.1 Non-negative Matrix Factorization

In the Non-negative Matrix Factorization (NMF) algorithm, we define the basis functions to be vectors of the same dimensionality as the input vectors. A given input example $\mathbf{v} \in \mathbb{R}_+^{\mathrm{F}}$ is then approximated as $\boldsymbol{\lambda} \in \mathbb{R}_+^{\mathrm{F}}$, a linear combination of non-negative basis vectors $\mathbf{w}_d \in \mathbb{R}_+^{\mathrm{F}}$ with non-negative coefficients $\mathrm{h}_d \in \mathbb{R}_+$ for $d = \{1, 2, ..., \mathrm{D}\}$, i.e. $\mathbf{v} \approx \boldsymbol{\lambda} = \sum_{d=1}^{\mathrm{D}} \mathrm{h}_d \mathbf{w}_d$. We can express this approximation in matrix notation as $\boldsymbol{\lambda} = \mathbf{W}\mathbf{h}$ by defining a matrix of basis vectors $\mathbf{W} \in \mathbb{R}_+^{\mathrm{F} \times \mathrm{D}}$, whose d^{th} column corresponds to $\mathbf{w}_d$, and a coefficient vector $\mathbf{h} \in \mathbb{R}_+^{\mathrm{D}}$, whose d^{th} element corresponds to h_d. If we then define an input matrix $\mathbf{V} \in \mathbb{R}_+^{\mathrm{F} \times \mathrm{T}}$ whose t^{th} column corresponds to the t^{th} input example $\mathbf{v}_t$, and a coefficient matrix $\mathbf{H} \in \mathbb{R}_+^{\mathrm{D} \times \mathrm{T}}$ whose t^{th}

column corresponds to the coefficient vector of the t^{th} example $\mathbf{h}_t$, we can express the reconstruction of all input examples compactly as $\mathbf{\Lambda} = \mathbf{WH}$, where $\mathbf{\Lambda} \in \mathbb{R}_{+}^{\mathrm{F}\times\mathrm{T}}$ is an approximation of $\mathbf{V}$. We will refer to $\mathbf{V}$ as the input matrix, $\mathbf{\Lambda}$ as the approximated or reconstructed input matrix, $\mathbf{W}$ as the basis vector, or dictionary, matrix, and $\mathbf{H}$ as the coefficient matrix. Given an input matrix $\mathbf{V}$, our task is then to find a dictionary matrix $\mathbf{W}$ and coefficient matrix $\mathbf{H}$ that yield a good reconstruction matrix $\mathbf{\Lambda}$.

We will first present a simple application of NMF using artificial data, and then develop the details of the algorithm. We use two basis functions to generate the artificial data: a Gaussian function, and the square of an exponentially damped sinusoid (both functions are non-negative), see figure 3.1.

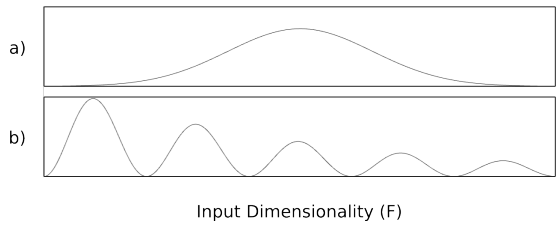

Figure 3.1: The two basis functions used to generate artificial data for the NMF Example a) Gaussian b) Square of an exponentially damped sinusoid

We generate input data by taking linear combinations of the basis functions with randomly generated coefficients, and apply NMF with D = 2. When the number of basis functions D is unknown, it becomes a hyperparameter that needs to be optimized. The learned basis functions are shown in figure 3.2. They are qualitatively indistinguishable from the ideal basis vectors.

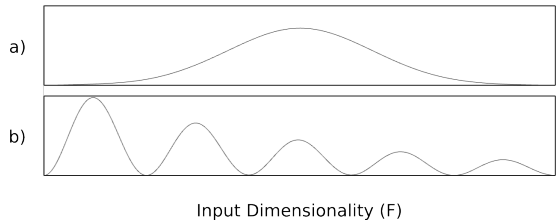

Figure 3.2: Basis functions learned by NMF from artificial data: a) learned Gaussian b) learned square of exponentially damped sinusoid

Figure 3.3 depicts several approximated input examples as well as their respective learned and ideal coefficients. The reconstructed examples and corresponding coefficients are indistinguishable from the input data.

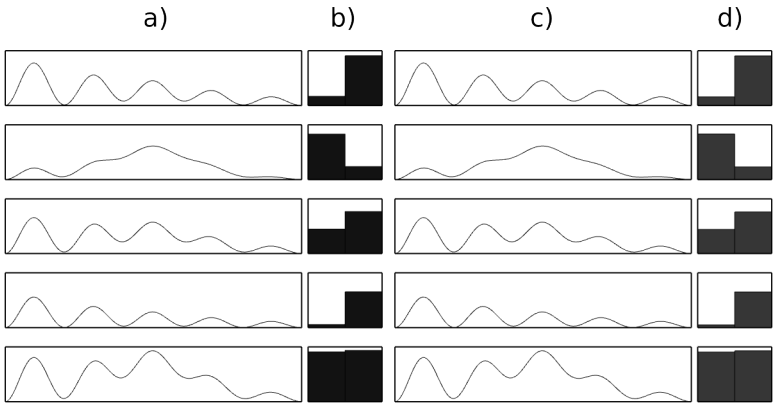

Figure 3.3: Five examples of the results of the NMF algorithm applied to artificial data described above a) input examples b) input coefficients for the Gaussian (left bar) and squared exponentially decayed sinusoid (right bar) c) corresponding reconstructed inputs d) learned coefficients. Both the input and learned coefficients are normalized to the maximum coefficient value across the entire dataset, and the bounds of their graphs are 0 to 1.

In order to find values for $\mathbf{W}$ and $\mathbf{H}$ that yield a good reconstruction $\mathbf{\Lambda}$, we define a function that measures the quality of the reconstruction. This cost function, or objective function, takes an input matrix $\mathbf{V}$ and a reconstruction matrix $\mathbf{\Lambda}$ and returns a scalar value that varies monotonically with reconstruction quality. Two frequently used cost functions are the least squares measure,

$$\frac{1}{2}\left\|\mathbf{V}-\mathbf{\Lambda}\right\|^2 = \frac{1}{2}\sum_{ij}\left(\mathbf{V}_{ij}-\mathbf{\Lambda}_{ij}\right)^2 \tag{3.1}$$

and a divergence measure,

$$\begin{aligned} D\left(\mathbf{V}\|\mathbf{\Lambda}\right) &= \sum_{ij}\mathbf{V}_{ij}\log\frac{\mathbf{V}_{ij}}{\mathbf{\Lambda}_{ij}}-\mathbf{V}_{ij}+\mathbf{\Lambda}_{ij} &(3.2)\\ &= \mathbf{V}\odot\log\left(\mathbf{V}\oslash\mathbf{\Lambda}\right)-\mathbf{V}+\mathbf{\Lambda} &(3.3)\end{aligned}$$

where $\odot$ and $\oslash$ refer to element-wise multiplication and division respectively, i.e. $\left(\mathbf{A}\odot\mathbf{B}\right)_{ij}=\mathbf{A}_{ij}\cdot\mathbf{B}_{ij}$ and $\left(\mathbf{A}\oslash\mathbf{B}\right)_{ij}=\mathbf{A}_{ij}\div\mathbf{B}_{ij}$. The divergence measure is equivalent to Kullback-Leibler (KL) divergence when $\sum_{ij}\mathbf{V}_{ij}=\sum_{ij}\mathbf{\Lambda}_{ij}=1$, such that the two matrices may be interpreted as probability distributions. $D\left(\mathbf{A}\|\mathbf{B}\right)$ is only defined if $\mathbf{A}_{ij}\neq 0$ and $\mathbf{B}_{ij}\neq 0\ \forall\, i,j$, and is not symmetric in general, i.e. $D\left(\mathbf{A}\|\mathbf{B}\right)\neq D\left(\mathbf{B}\|\mathbf{A}\right)$.

Once a cost function has been chosen, we turn to the task of optimization: find values of $\mathbf{W}$ and $\mathbf{H}$ that yield a minimal value of the cost function. A common technique used in optimization is that of gradient descent. Intuitively, the partial derivative of the cost function with respect to each parameter indicates how local changes to the parameter value affect the value of the cost function. We can therefore define rules that update the parameter values such that the value of the cost function decreases. For example, we take

the partial derivative of the least squares cost function with respect to arbitrary elements of the dictionary matrix $\mathbf{W}_{fd}$ and the coefficient matrix $\mathbf{H}_{dt}$. In the following we make use of the Kronecker delta function, $\delta_{ij} = \begin{cases} 1 & i = j \\ 0 & i \neq j \end{cases}$.

$$\begin{aligned}
\frac{\partial}{\partial \mathbf{W}_{fd}} \left(\frac{1}{2} \left\| \mathbf{V} - \mathbf{\Lambda} \right\|^2 \right) &= -\sum_{ij} \left(\mathbf{V}_{ij} - \mathbf{\Lambda}_{ij} \right) \frac{\partial \mathbf{\Lambda}_{ij}}{\partial \mathbf{W}_{fd}} \\
&= -\sum_{ij} \left(\mathbf{V}_{ij} - \mathbf{\Lambda}_{ij} \right) \frac{\partial}{\partial \mathbf{W}_{fd}} \sum_{\tilde{d}} \mathbf{W}_{i\tilde{d}} \mathbf{H}_{\tilde{d}j} \\
&= -\sum_{ij} \left(\mathbf{V}_{ij} - \mathbf{\Lambda}_{ij} \right) \delta_{if} \mathbf{H}_{dj} \\
&= -\sum_{j} \mathbf{V}_{fj} \mathbf{H}_{dj} - \mathbf{\Lambda}_{fj} \mathbf{H}_{dj} \\
&= -\left(\mathbf{V}\mathbf{H}^{\mathrm{T}} - \mathbf{\Lambda}\mathbf{H}^{\mathrm{T}} \right)_{fd}
\end{aligned}$$

$$\begin{aligned}
\frac{\partial}{\partial \mathbf{H}_{dt}} \left(\frac{1}{2} \left\| \mathbf{V} - \mathbf{\Lambda} \right\|^2 \right) &= -\sum_{ij} \left(\mathbf{V}_{ij} - \mathbf{\Lambda}_{ij} \right) \frac{\partial \mathbf{\Lambda}_{ij}}{\partial \mathbf{H}_{dt}} \\
&= -\sum_{ij} \left(\mathbf{V}_{ij} - \mathbf{\Lambda}_{ij} \right) \delta_{jt} \mathbf{W}_{id} \\
&= -\sum_{i} \left(\mathbf{V}_{it} \mathbf{W}_{id} - \mathbf{\Lambda}_{it} \mathbf{W}_{id} \right) \\
&= -\left(\mathbf{W}^{\mathrm{T}}\mathbf{V} - \mathbf{W}^{\mathrm{T}}\mathbf{\Lambda} \right)_{dt}
\end{aligned}$$

Using the standard gradient descent method, we can then define update rules as $\mathbf{W}_{fd} \leftarrow \mathbf{W}_{fd} - \eta \left(\mathbf{V}\mathbf{H}^{\mathrm{T}} - \mathbf{\Lambda}\mathbf{H}^{\mathrm{T}} \right)_{fd}$ and $\mathbf{H}_{dt} \leftarrow \mathbf{H}_{dt} - \eta \left(\mathbf{W}^{\mathrm{T}}\mathbf{V} - \mathbf{W}^{\mathrm{T}}\mathbf{\Lambda} \right)_{dt}$. The variable $\eta \in \mathbb{R}_+$ is referred to as the step size or learning rate. When η is set small enough, the update rule will decrease the value of the cost function. By iteratively updating $\mathbf{W}$ and $\mathbf{H}$ according to these update rules, we may find a local minimum of the least squares cost function. Lee and Seung popularized a multiplicative update rule that eliminates this parameter and can lead to much faster convergence [30]. The multiplicative factor is constructed using the absolute value of the negative terms of the gradient in the numerator, and of the positive terms of the gradient in the denominator, e.g. $\mathbf{W} \leftarrow \mathbf{W} \left| \frac{\left(\frac{\partial Cost}{\partial \mathbf{W}} \right)^-}{\left(\frac{\partial Cost}{\partial \mathbf{W}} \right)^+} \right|$. The multiplicative update rules for the least squares and divergence cost functions are,

$$\mathbf{W} \leftarrow \mathbf{W} \frac{\mathbf{V}\mathbf{H}^{\mathrm{T}}}{\mathbf{\Lambda}\mathbf{H}^{\mathrm{T}}} \tag{3.4}$$

$$\mathbf{H} \leftarrow \mathbf{H} \frac{\mathbf{W}^{\mathrm{T}}\mathbf{V}}{\mathbf{W}^{\mathrm{T}}\mathbf{\Lambda}} \tag{3.5}$$

and,

$$\mathbf{W} \leftarrow \mathbf{W}\frac{\frac{\mathbf{V}}{\mathbf{\Lambda}}\mathbf{H}^{\mathrm{T}}}{\mathbf{1}\cdot\mathbf{H}^{\mathrm{T}}} \tag{3.6}$$

$$\mathbf{H} \leftarrow \mathbf{H}\frac{\mathbf{W}^{\mathrm{T}}\frac{\mathbf{V}}{\mathbf{\Lambda}}}{\mathbf{W}\cdot\mathbf{1}} \tag{3.7}$$

respectively. This is equivalent to defining parameter-dependent adaptive learning rates as follows. We first define $\boldsymbol{\eta}^{\mathbf{W}} \in \mathbb{R}_{+}^{\mathrm{F}\times\mathrm{D}}$ and $\boldsymbol{\eta}^{\mathbf{H}} \in \mathbb{R}_{+}^{\mathrm{D}\times\mathrm{T}}$ such that $\boldsymbol{\eta}^{\mathbf{W}}_{ij}$ and $\boldsymbol{\eta}^{\mathbf{H}}_{ij}$ are the learning rates for parameters $\mathbf{W}_{ij}$ and $\mathbf{H}_{ij}$, respectively. We then define the additive update rules as $\mathbf{W} \leftarrow \mathbf{W} - \boldsymbol{\eta}^{\mathbf{W}} \odot \left(\mathbf{V}\mathbf{H}^{\mathrm{T}} - \mathbf{\Lambda}\mathbf{H}^{\mathrm{T}}\right)$ and $\mathbf{H} \leftarrow \mathbf{H} - \boldsymbol{\eta}^{\mathbf{H}} \odot \left(\mathbf{W}^{\mathrm{T}}\mathbf{V} - \mathbf{W}^{\mathrm{T}}\mathbf{\Lambda}\right)$. Finally, defining the learning rates as $\boldsymbol{\eta}^{\mathbf{W}} = \frac{\mathbf{W}}{\mathbf{\Lambda}\mathbf{H}^{\mathrm{T}}}$ and $\boldsymbol{\eta}^{\mathbf{H}} = \frac{\mathbf{H}}{\mathbf{W}^{\mathrm{T}}\mathbf{\Lambda}}$ results in the multiplicative update rules of least squares cost function 3.1. Lee and Seung proved that the multiplicative update rules 3.4, 3.5, 3.6 and 3.7 for the least squares and divergence cost functions, decrease the value of cost function unless at a minimum, despite the fact that the effective learning rates are not small.

3.1.2 Non-negative Matrix Factor 2D Deconvolution

Let us now consider a family of algorithms which approximates input matrix in a richer manner than simple linear combinations of basis vectors, as in NMF. The idea is the same as before, i.e. the model will approximate the input using linear combinations of basis functions, however, we introduce zero-fill shift operators $\overrightarrow{(\cdot)}^{\tau}$ and $\overset{\downarrow\phi}{(\cdot)}$ that allow the algorithm to shift the basis functions prior to combining them. This will allow the model to efficiently capture cases in which the input contains shifted versions of a small number of basis functions.

The zero-fill right-shift operator $\overrightarrow{(\cdot)}^{\tau}$ takes an input matrix and an integer shift parameter $\tau > 0$, and returns a matrix with the same shape as the input matrix. To generate the output matrix, the columns of the input matrix are shifted τ columns to the left, with the τ first columns filled with zeros, and the last τ columns being discarded. For example, we present the application of the zero-fill shift operator to a 2×3 matrix $\mathbf{A} = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix}$ with $\tau = 1$ and 2:

$$\overrightarrow{\mathbf{A}}^{1} = \begin{bmatrix} 0 & 1 & 2 \\ 0 & 4 & 5 \end{bmatrix}, \overrightarrow{\mathbf{A}}^{2} = \begin{bmatrix} 0 & 0 & 1 \\ 0 & 0 & 4 \end{bmatrix}$$

The zero-fill downward-shift operator $\overset{\downarrow\phi}{(\cdot)}$ works similarly, except that rows are shifted down instead of columns

right. For example, for ϕ =1 and 2:

$$\overset{\downarrow 1}{\mathbf{A}} = \begin{bmatrix} 0 & 0 & 0 \\ 1 & 2 & 3 \end{bmatrix}, \overset{\downarrow 2}{\mathbf{A}} = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

We will first introduce two algorithms that use one of these shift operators. Since they may both be referred to as NMFD, we will distinguish between them as NMFD$\rightarrow$ and NMFD$\uparrow$, depending on the direction in which the shifts are permitted. NMFD$\rightarrow$ was introduced by Smaragdis [41]. In the general case, Non-negative Matrix Factor 2D Deconvolution (NMF2D), shifts are permitted in both directions. NMF2D was introduced by Schmidt and Mørup [38], who also presented a sparse extension, Sparse NMF2D (SNMF2D) in [39].

3.1.2.1 NMFD$\uparrow$

In the case of the downward shift operator, the basis vectors are defined as before, i.e. $\mathbf{W} \in \mathbb{R}_{+}^{\mathrm{F} \times \mathrm{D}}$, and each input example $\mathbf{v}$ is approximated independently. The model has the ability to shift the basis vectors using the downward-shift operator described above. For each shift amount ϕ, we define a vector $\mathbf{h}^{\phi} \in \mathbb{R}_{+}^{\mathrm{D}}$ such that $\mathbf{h}_d^{\phi}$ corresponds to the coefficient of $\mathbf{w}_d$ for the shift amount ϕ. The reconstruction of an input example $\mathbf{v}$ then becomes $\boldsymbol{\lambda} = \sum_{\phi} \sum_d \overset{\downarrow \phi}{\mathbf{W}}_d \mathbf{h}_d^{\phi} = \sum_{\phi} \overset{\downarrow \phi}{\mathbf{W}} \mathbf{h}^{\phi}$. Defining $\mathbf{H}^{\phi} \in \mathbb{R}_{+}^{\mathrm{D} \times \mathrm{T}}$, whose t^{th} column corresponds to $\mathbf{h}^{\phi}$ for the t^{th} example, the reconstruction of the input matrix becomes $\boldsymbol{\Lambda} = \sum_{\phi} \overset{\downarrow \phi}{\mathbf{W}} \mathbf{H}^{\phi}$.

Let us now present a toy example of an application of this version of NMFD. First, we define a binary basis function for which the values at 15 random indexes are set to one and the rest to zero. Each input example is then generated by randomly selecting a number of shifts, and for each shift, a shift amount and a scalar multiplicative factor. The results are summed together to create a single input example. We then apply the NMFD$\uparrow$ algorithm with a dictionary size of 1, yielding the results shown in figure 3.4 below. The algorithm correctly identifies the basis function as well as the appropriate shift amounts and multiplicative factors.

3.1.2.2 NMFD$\rightarrow$

Next, we introduce the alternate version of NMFD, in which shifting is introduced along the columns of the input matrix, as opposed to the rows. In this case, we define basis matrices (as opposed to the basis vectors above), $\mathbf{w}_d^{\tau} \in \mathbb{R}_{+}^{\mathrm{F}}$ for $\tau \in \{1, 2, ..., \Upsilon\}$, where Υ determines the number of columns in the basis matrices. We then define $\mathbf{W}^{\tau} \in \mathbb{R}_{+}^{\mathrm{F} \times \mathrm{D}}$, whose d^{th} column corresponds to $\mathbf{w}_d^{\tau}$. The d^{th} basis matrix is therefore $\in \mathbb{R}_{+}^{\mathrm{F} \times \Upsilon}$, and is componsed of d^{th} columns of $\mathbf{W}^{\tau}$ for $\tau \in \{1, 2, ..., \Upsilon\}$. The coefficient matrix $\mathbf{H}$ is defined as in the original NMF, i.e. $\mathbf{H} \in \mathbb{R}_{+}^{\mathrm{D} \times \mathrm{T}}$. The reconstruction of the input matrix is then defined as $\boldsymbol{\Lambda} = \sum_{\tau} \mathbf{W}^{\tau} \overset{\rightarrow \tau}{\mathbf{H}}$.

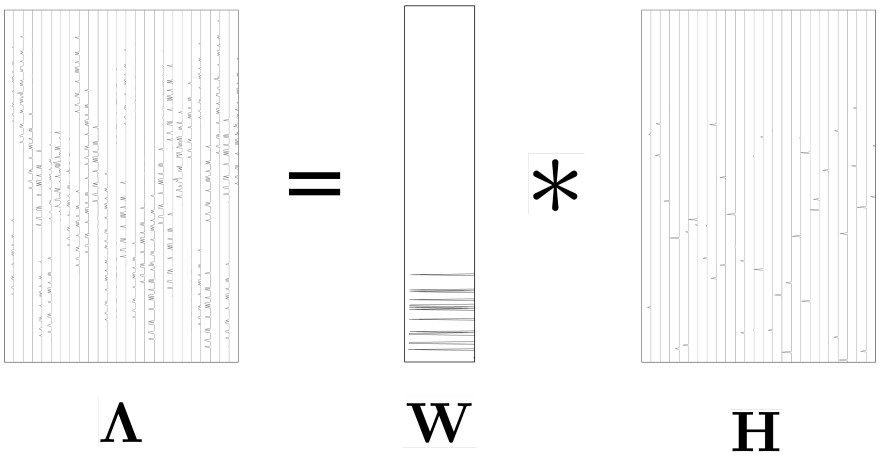

Figure 3.4: Example application of NMFD↑. The reconstruction of 25 randomly chosen examples are shown. From left to right: the reconstruction matrix $\mathbf{\Lambda}$ (F x T), computed by convolving $\mathbf{W}$ with $\mathbf{H}$, the learned basis matrix $\mathbf{W}$ (F x D), and the learned shift coefficients $\mathbf{H}$ (F x T x D). Since there is only one basis vector, D = 1. The learned basis vector and reconstruction matrices are qualitatively indistinguishable from the true basis vector and input matrix $\mathbf{V}$, respectively.

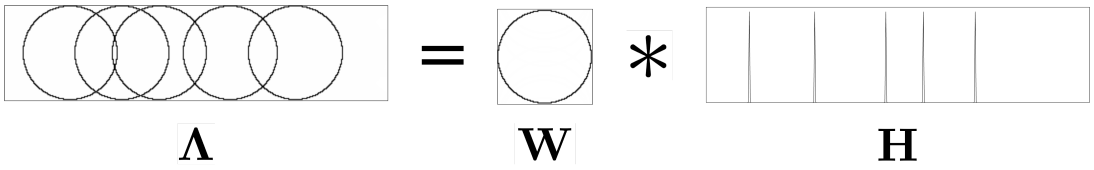

Figure 3.5: Example application of NMFD→. From left to right: the reconstruction matrix $\mathbf{\Lambda}$ (F x T), computed by convolving $\mathbf{W}$ with $\mathbf{H}$, the learned basis matrix $\mathbf{W}$ (F x τ), and the learned shift coefficents $\mathbf{H}$ (T). The learned basis matrix and reconstruction matrices are qualitatively indistinguishable from the true basis matrix and input matrix $\mathbf{V}$, respectively.

Figure 3.5 depicts the results of the application of NMFD→ to a toy example in which the single basis matrix is a circle, and the input matrix is generated by summing five random shifts of the basis matrix. The NMFD→ algorithm learns the appropriate basis matrix and shift amounts to accurately reconstruct the input.

3.1.2.3 NMF2D

If we allow shifts in both dimensions of the input matrix, the reconstruction becomes:

$$\mathbf{\Lambda} = \sum_{\tau} \sum_{\phi} \overset{\downarrow\phi}{\mathbf{W}^{\tau}} \overset{\rightarrow\tau}{\mathbf{H}^{\phi}} \tag{3.8}$$

In this case, we have basis matrices which can be shifted to any index of the reconstruction matrix. This is refered to as Non-negative Matrix Factor 2D Deconvolution (NMF2D). In figure 3.6 we present a toy

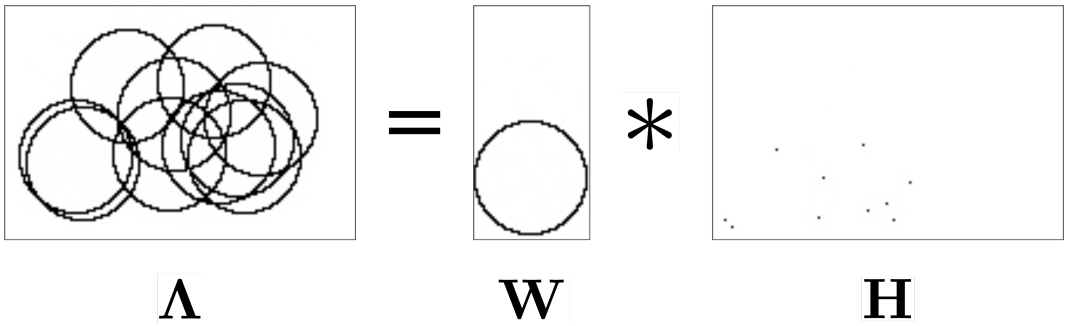

Figure 3.6: Example application of NMF2D. From left to right: the reconstruction matrix $\mathbf{\Lambda}$ (F x T), computed by convolving $\mathbf{W}$ with $\mathbf{H}$, the learned basis matrix $\mathbf{W}$ (F x τ), the learned shift coefficents $\mathbf{H}$ (F x T).

example, similar to the previous example, except that the circles may be shifted to arbitrary indexes in the input matrix.

3.1.3 Coefficient Sparsity

In all of the above examples, the coefficients were quite sparse both in generation of the data and in the learned decomposition. A sparse decomposition is one in which only a small number of the basis functions are active for a given example. In the examples above, the non-negativity constraint alone was enough to yield sparse decompositions. There are situations, however, in which the non-negativity constraint is not sufficient to yield sparse decompositions, despite the sparsity of the coefficients used to generate the data. In these cases, the resulting reconstructions may be accurate, i.e. low value of the error function, however they may yield incorrect decompositions that lack interpretability. In figure 3.7, we demonstrate this using an application of NMFD↑ for which the non-negativity constraint alone does not yield the correct decomposition. This is similar to the example shown in figure 3.4, except that the basis vector is instead the square of an exponentially damped sinusoid.

In these situations, we may introduce a sparsity constraint to the chosen cost function to encourage the algorithm to learn sparse decompositions. A common sparsity constraint is the l_1 norm $|\mathbf{H}|_1 = \sum_{i,j} |\mathbf{H}_{ij}|$. In figure below, we have added a sparsity constraint to NMFD↑, notice that $\mathbf{H}$ is now much sparser, yielding correct instantaneous, event-type activations. Multiplicative update rules similar to those presented in section 3.1.1, are presented for the least squares and KL divergence cost functions with a generalized sparsity constraint in [31].

Sparsity constraints used without a non-negativity constraint may also lead to parts-based decompositions in which the basis functions are summed together without cancellation. However, the non-negativity constraint alone may be enough to yield such parts-based decompositions depending on the statistics of the data

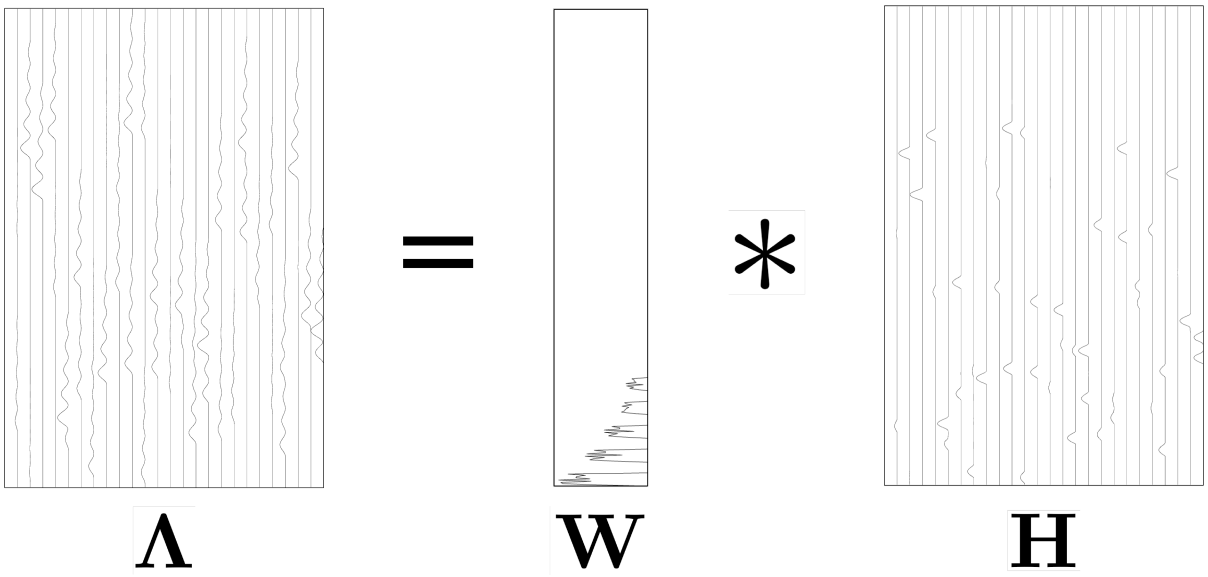

Figure 3.7: Example for which NMFD↑ does not yield sparse decompositions. Note that while the reconstructions in $\mathbf{\Lambda}$ (F x T) are accurate, the basis functions have been distributed across $\mathbf{W}$ (F) and $\mathbf{H}$ (F x T).

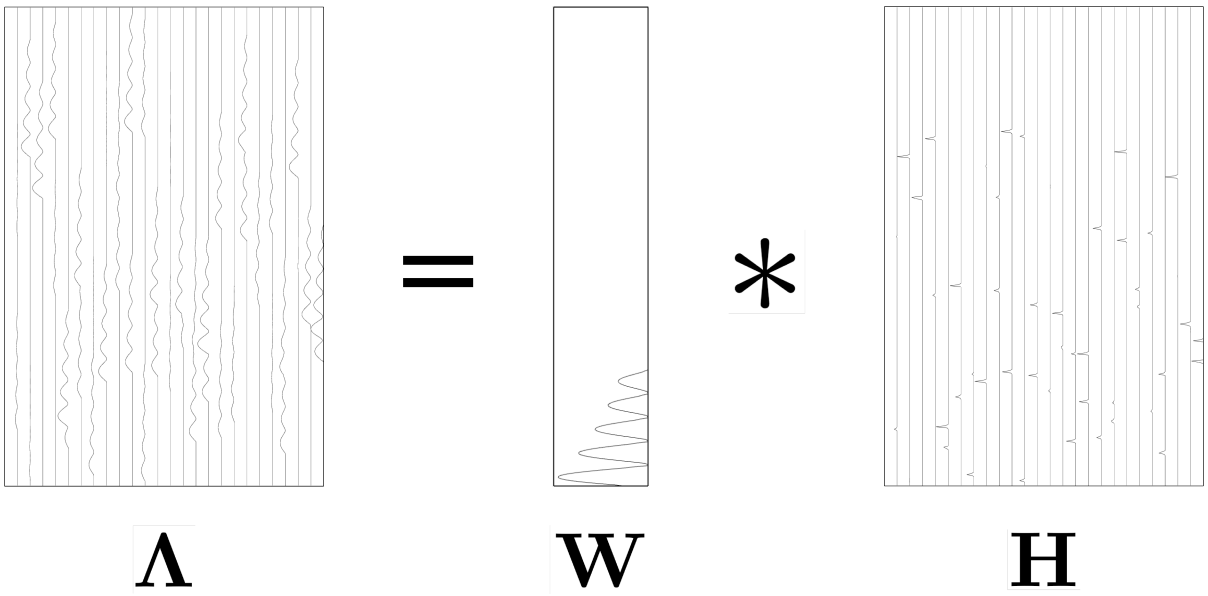

Figure 3.8: Same example as 3.7, however with the addition an l_1 sparsity constraint, which allows us to find a "correct" minimum. The basis functions are no longer distributed across $\mathbf{W}$ and $\mathbf{H}$, and $\mathbf{H}$ is correctly sparse.

and the number of basis functions [20]. For the remainder of this thesis, we will rely on the non-negativity constraint alone, and leave investigation of the effects of sparsity constraints to future work.

3.2 Semi-supervised Algorithms

Supervised learning tasks differ from unsupervised learning tasks in that for each input example, we have a target that we are to learn to predict. The target could be real-valued, in which case the task is referred to as regression, or binary, in which it is referred to as classification. We will focus here on classification tasks. In chapter 4, we will make use of logistic regression classification algorithm. The output of the logistic regression model is defined as $y = p\left(t|\mathbf{v}\right) = \sigma\left(\mathbf{w}^{\mathrm{T}}\mathbf{v}\right)$, where $\mathbf{v}$ is the input to the model, $\mathbf{w}$ are the model parameters, and t is a binary random variable whose probability we wish to predict. The training set for such an algorithm consists of pairs of input vectors $\mathbf{v}_i$ and binary targets t_i, i.e. $\{\mathbf{v}_i, t_i\}_{i=1}^{N}$, and the goal is to predict the input vectors' corresponding target values. Learning is done by minimizing the negative logarithm of the likelihood function, or the cross-entropy cost function, i.e. $-\ln p\left(\mathbf{t}|\mathbf{w}\right) = -\sum_{n=1}^{N}\left(t_n \log y_n + (1-t_n)\log\left(1-y_n\right)\right)$, using standard gradient descent. The classification problem we will study is equivalent to multiple binary classification problems for which the target is a binary vector in which each element may be either zero or one. Logistic regression may be extended to handle this kind of classification simply by training several logistic regressors in parallel.

Semi-supervised learning involves a combination of supervised and unsupervised learning. In the algorithms we present here, there are two separate training phases: an unsupervised training phase followed by supervised training phase. First we train an NMF model in the standard way, learning a sparse representation of the preprocessed input. We then use the supervised logistic regression to perform a classification task using the learned representation as input. See figure 3.9 for a depiction of the architecture.

3.3 Applications to Music Audio

In recent years, there has been significant interest in the application of sparse and non-negative matrix decomposition algorithms to music audio [21, 22, 23, 26, 27, 28, 29, 32, 35, 38, 39, 40, 42, 41, 43, 37, 44, 45, 46, 47, 48]. The primary goal has been in polyphonic pitch detection tasks, where we wish to determine the pitch of each note present in a given frame of audio. In this section we will demonstrate how the factorization algorithms discussed above have been applied to music audio in the literature.

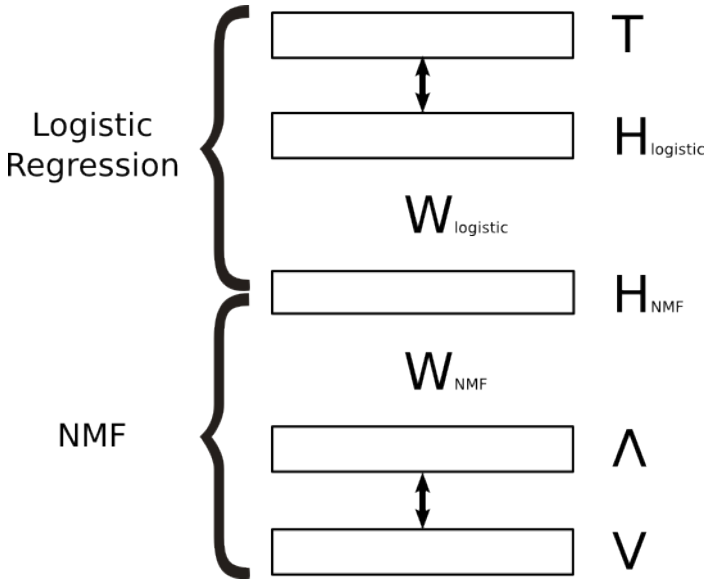

Figure 3.9: Semi-supervised NMF: NMF unsupervised training, followed by logistic regression supervised training. Note that the parameters are not shared between the two phases.

3.3.1 Spectrogram Factorization

As discussed in section 2.1, a spectrogram is a two dimensional, i.e. time-frequency, representation of audio. A spectrogram may be represented digitally as a matrix whose rows correspond to frequency bin and whose columns correspond to time index. Such spectrograms may be used as input matrices $\mathbf{V} \in \mathbb{R}_{+}^{\mathrm{F} \times \mathrm{T}}$ to the NMF algorithms described above, as their rows represent input dimensionality, and their columns the example index. In the following chapter, we will conduct various experiments to determine the qualitative effects of different parameters of spectrogram factorization algorithms, as well as an evaluation of their use in monophonic and polyphonic pitch detection.

3.3.2 Spectrogram Deconvolution

There are two important limitations in using non-convolutive NMF to separate music sources. First, since source mixing is instantaneous, the algorithms cannot model a note's temporal evolution. Second, due to lack of shift-invariance on the frequency axis, notes from a given instrument will invariably be modeled as separate basis functions. If instrument identification is desired, some form of clustering must therefore be performed after the fact, in order to group basis functions belonging to the same instrument. NMF2D algorithms discussed in section 3.1.2 have been proposed to resolve these two limitations.

In applying NMFD$\rightarrow$, the temporal progression of notes may be modeled. However, this introduces a new hyperparameter, namely the durations of the basis matrices, whose ideal values would be the lengths of the notes present in the input. If two notes having the same pitch but different durations are present in the

input, two basis functions would be required to model the two different notes.

In applying NMFD↑, and log-scaling the input's frequency axis as described in section 2.3.5, we expect to be able model notes played by a single instrument using a single basis vector. However, this relies on the assumption that the relative amplitudes of an instruments harmonics are pitch invariant, which is typically only the case for small pitch differences. This can be understood by considering the source-filter model of sound production. According to this model, acoustic sources can be separated into a source, which generates harmonic frequencies at a given pitch (such as the vocal cords or a guitar string), and a filter, which colours the sound (such as the shape of the mouth or the guitar body). While the sources are shift-invariant in frequency, the filters are not, and the application of NMFD↑ assumes frequency invariance in both. NMF can been adapted to incorporate the source-filter models as can be seen in [47].

NMF2D have also been applied to take into account both limitations [38, 39], however they are very sensitive to the choice of dictionary size parameters. Higher order tensor factorizations have also been introduced to account for stereo recordings in which a source signal appears in more than one channel with different gain [21].

3.3.3 Phase-aware Factorization

Perhaps the most fundamental problem of the NMF approach to spectrogram factorization is in its application to polyphonic audio in which there is overlap between sources in the time-frequency plane. These overlaps are very common in music, as notes whose fundamental frequencies have ratios of low integer values sound pleasing, or consonant, and are at the root of harmony. The NMF algorithm makes the assumption that addition of sources' contributions is linear. While addition in the time domain is indeed linear, in the spectrogram domain, it is not. This is due to the non-linearity imposed by the absolute value or squaring functions that discard phase information. In the worst case, the sum of two sinusoids having equal frequencies and amplitudes, and a phase difference of π, will result in silence. In practice, sources' frequencies are rarely exactly equal, which leads to an effect referred to as beating, where the amplitude of the sum repeatedly grows to maximum then shrinks to a minimum at a frequency equal to the difference between the two source frequencies.

Two research groups have attempted to incorporate knowledge of this interference into NMF-type models, arriving at a third cost function which we will refer to as the phase-aware cost function, or PA for short. Abdallah and Plumbley propose a probabilistic model with multiplicative noise that "is physically accurate for the composition of power spectra arising from the superposition of phase-incoherent Gaussian processes" [13]. The resulting divergence error measure was also found by Parray and Essa using a maximum likeli-

hood approach by representing phase as a uniform random variable combined with the false assumptions of independent time-frequency measures, and a large number of overlapping components [34]. The resulting divergence function,

$$\mathrm{PA}\left(\mathbf{V},\mathbf{\Lambda}\right) = D\left(1\|\mathbf{V}/\mathbf{\Lambda}\right) = \sum_{ij} \frac{\mathbf{V}_{ij}}{\mathbf{\Lambda}_{ij}} - 1 + \log\left(\frac{\mathbf{\Lambda}_{ij}}{\mathbf{V}_{ij}}\right) \tag{3.9}$$

results in the following update rules,

$$\mathbf{W} \leftarrow \mathbf{W}\frac{\frac{\mathbf{V}}{\mathbf{\Lambda}^2}\mathbf{H}^{\mathrm{T}}}{\frac{1}{\mathbf{\Lambda}}\mathbf{H}^{\mathrm{T}}} \tag{3.10}$$

$$\mathbf{H} \leftarrow \mathbf{H}\frac{\mathbf{W}^{\mathrm{T}}\frac{\mathbf{V}}{\mathbf{\Lambda}^2}}{\mathbf{W}^{\mathrm{T}}\frac{1}{\mathbf{\Lambda}}} \tag{3.11}$$

where $\frac{1}{\mathbf{A}}$ refers to element-wise inverse, and $\mathbf{A}^2$ to element-wise squaring. Comparisons of this cost function with the previously used LS and KL when applied to music data are notably absent from the literature. We will therefore present such comparisons in the following chapter.

Chapter 4

Experiments

4.1 Introduction

Throughout this chapter, we will investigate the learning and generalization behaviours of the NMF algorithm with the three principle cost functions discussed in chapter 3, when applied to monophonic and harmonized scales. In sections 4.2 through 4.6, we will study different model variants, such as dictionary initialization and inference techniques, and model parameters such as dictionary size. We will then study the effect of data parameters in section 4.7, for example instrument type, on the algorithms' abilities to generalize to out-of-sample parameter values.

We will start our analysis of model variants with the choice of cost function (section 4.2), followed by dictionary initialization techniques (section 4.3). We will then study different dictionary labeling techniques (section 4.4) that will allow the factorization algorithms to be used for polyphonic pitch classification. We then analyze the effect of dictionary size on optimization and interpretability (section 4.5). Two different inference techniques will then studied (section 4.6), i.e. algorithms that allow us to determine the value for $\mathbf{H}$ given an input matrix $\mathbf{V}$ and a dictionary $\mathbf{W}$. Finally, in section 4.7, we will turn our attention to data parameters. Here, we will study the ability of the different learning algorithms and labeling algorithms to generalize to unseen note intensities (section 4.7.1.1), note durations (section 4.7.1.2), and instruments (section 4.7.1.3).

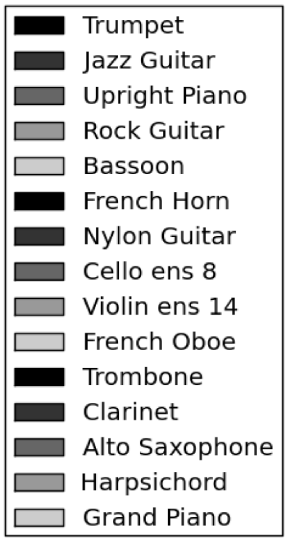

Figure 4.1: Legend for Figures 4.2 and 4.10.

4.2 Cost Functions

As discussed in chapter 3, there are three principal cost functions used for non-negative spectrogram factorization:

$$\mathrm{LS}\left(\mathbf{V},\mathbf{\Lambda}\right) = \left\|\mathbf{V}-\mathbf{\Lambda}\right\|^2 = \frac{1}{2}\sum_{ij}\left(\mathbf{V}_{ij}-\mathbf{\Lambda}_{ij}\right)^2 \tag{4.1}$$

$$\mathrm{KL}\left(\mathbf{V},\mathbf{\Lambda}\right) = D\left(\mathbf{V}\|\mathbf{\Lambda}\right) = \sum_{ij}\mathbf{V}_{ij}\log\left(\frac{\mathbf{V}_{ij}}{\mathbf{\Lambda}_{ij}}\right) - \mathbf{V}_{ij} + \mathbf{\Lambda}_{ij} \tag{4.2}$$

$$\mathrm{PA}\left(\mathbf{V},\mathbf{\Lambda}\right) = D\left(1\|\mathbf{V}/\mathbf{\Lambda}\right) = \sum_{ij}\frac{\mathbf{V}_{ij}}{\mathbf{\Lambda}_{ij}} - 1 + \log\left(\frac{\mathbf{\Lambda}_{ij}}{\mathbf{V}_{ij}}\right) \tag{4.3}$$

In this section, we will study how the values of these cost measures evolve during optimization for an A major scale played at medium volume. The progressions of each of the three cost functions during the optimization of each of the three algorithms are depicted in figure 4.2. The results are organized in a 3 x 3 grid whose columns correspond to the algorithms, and whose rows correspond to the cost functions: element ij of the grid corresponds to the evolution of cost i as we optimize algorithm j. Each graph depicts the evolution of the cost of 15 different instruments, starting from 20 different random initializations, i.e. each band is a set of 20 trajectories for a given instrument. The histograms at the left of the graphs show the final values for trajectories. Instruments are ordered from left to right according to decreasing average least squared error measure after the first iteration of optimization. Thus in the top left graph, the error value at the first epoch will always decrease from left to right.

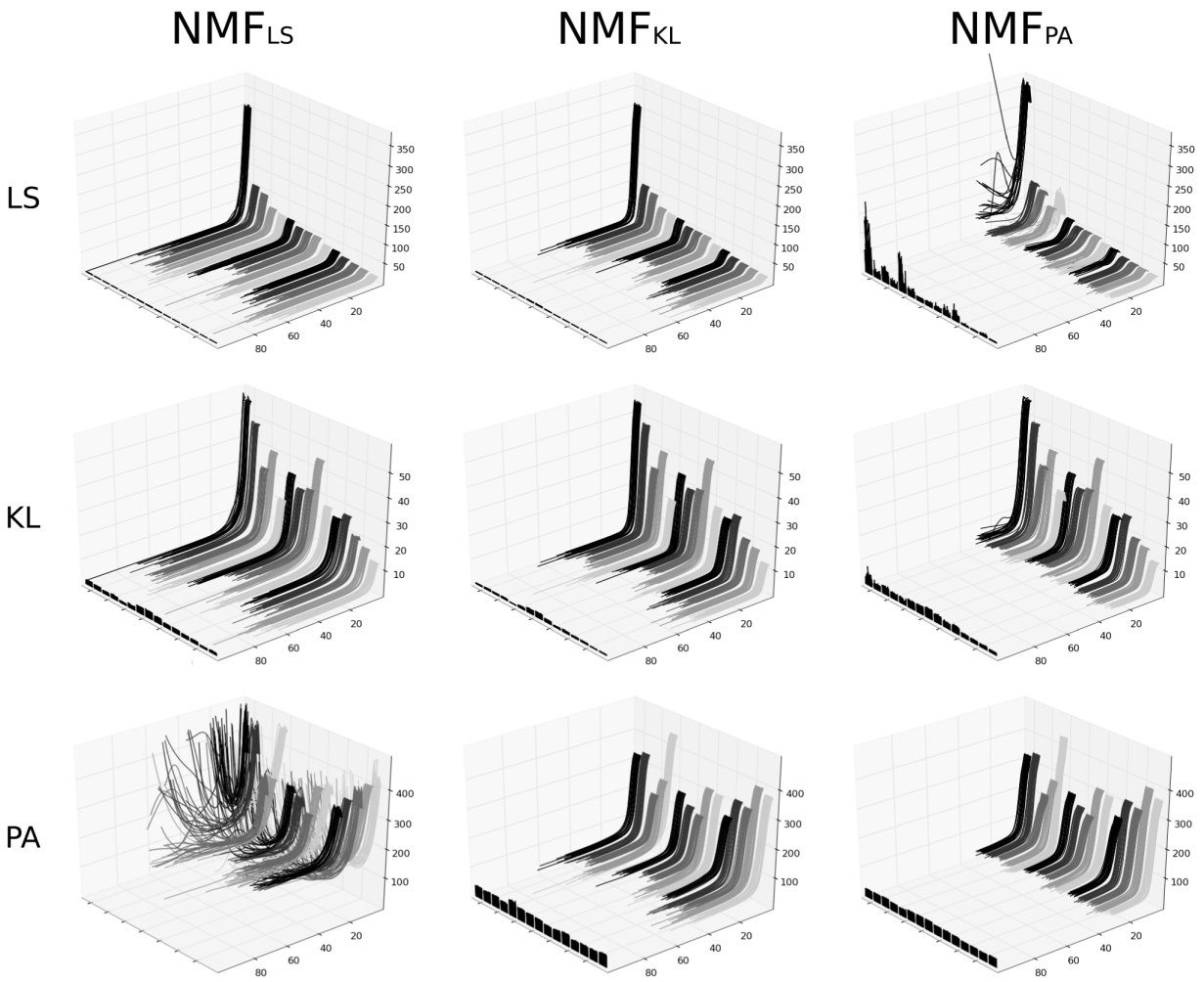

Figure 4.2: Training error curves using different error functions, with a dictionary size of 25. 3 x 3 grid showing minimization of each error (rows) by each algorithm (columns). Vertical axis represents the cost of the corresponding column, labeled horizontal axis is number of epochs. The histograms at the left of each graph show the trajectories' final values (lower-left histogram is omitted since the majority of trajectories diverged). See Figure 4.1 for a legend of the instruments used.

Looking at the graphs along the diagonal of the grid, we see the value of each cost function as we optimize that same cost function decreases monotonically. This is expected in the case of LS and KL, as the update rules have been proven to decrease the value of the error unless at a local minimum [30], and it is interesting to note that this appears to be the case for PA as well. The errors decrease quite quickly at first, within the first 10 to 20 epochs, then slowly approach an asymptote. We note that the optimization is fastest for the PA cost function, followed by KL, and finally LS. It takes approximately twice as long for the LS optimization to reach convergence than the PA optimization.

Investigating the off-diagonal elements, we may gain insight into how the other costs evolve as we optimize a given cost function, for example the top-center graph depicts how the LS varies as we minimize KL. We

note that the relative errors between different instruments have the same pattern for KL and LS, regardless of whether we optimize KL or LS, i.e. in comparing the top-left and top-center, as well as in comparing the center-left and center-center. Looking at the bottom-left graph, we note that as we optimize LS, the PA cost function often diverges. It is interesting to note that this seems to affect specific instruments, as a few instruments do converge consistently. We hypothesize that this is due to the assumption of multiple interacting partials made in the PA model, which may be more or less violated depending on the instrument. For example, when a single key is played on a piano, multiple strings are struck, each producing a complex tone whose pitch is very near to the others'. We will see more evidence supporting this hypothesis in section 4.7, as we note an important difference in the performance of PA between monophonic and polyphonic pitch detection tasks, however a thorough analysis is left to future work. Finally, we note that LS cost function may also diverge as we optimize PA, however this happens much less frequently.

Optimization of the KL cost function monotonically minimizes each of the three cost functions. Although it does not result in the fastest convergence, the results are consistently reliable and will therefore be used for the experiments in following sections for which we use a single cost function.

4.3 Dictionary Initialization

As we saw in section 4.2, the three NMF algorithms are quite robust to different random initializations for the inputs we studied, i.e. the local minima achieved were approximately equivalent in terms of cost, regardless of initialization. In this section, we will investigate an alternative to random initialization that incorporates domain knowledge of the true basis functions, and yields dictionaries that are more easily interpretable.

First, let us take a look at a simple example using a major scale generated using the Grand Piano instrument. In figure 4.3 below, we show the results of five different random initializations, using a dictionary size of 10 (slightly larger than the number of notes, 7). We can see that the *order* of the encoded notes differs significantly between initializations, this is perhaps most evident by looking at the coefficients matrices $\mathbf{H}$, where for each note (approximately one seventh of the duration of the horizontal axis), we note which dictionary elements are active.

An alternative to random initialization is to initialize the dictionary using domain knowledge of sounds generated by musical instruments. As described in section 2.2.1, notes played by musical instruments are typically harmonic in nature, i.e. they contain energy at integer multiples of the note's fundamental frequency. Therefore, we may choose to initialize the dictionary elements with spectra that contain energy at expected frequencies of the underlying notes, which has been referred to in the literature as Harmonic NMF [36]. Since only harmonic frequencies are present in the initial dictionary elements, the algorithm needs only to adjust

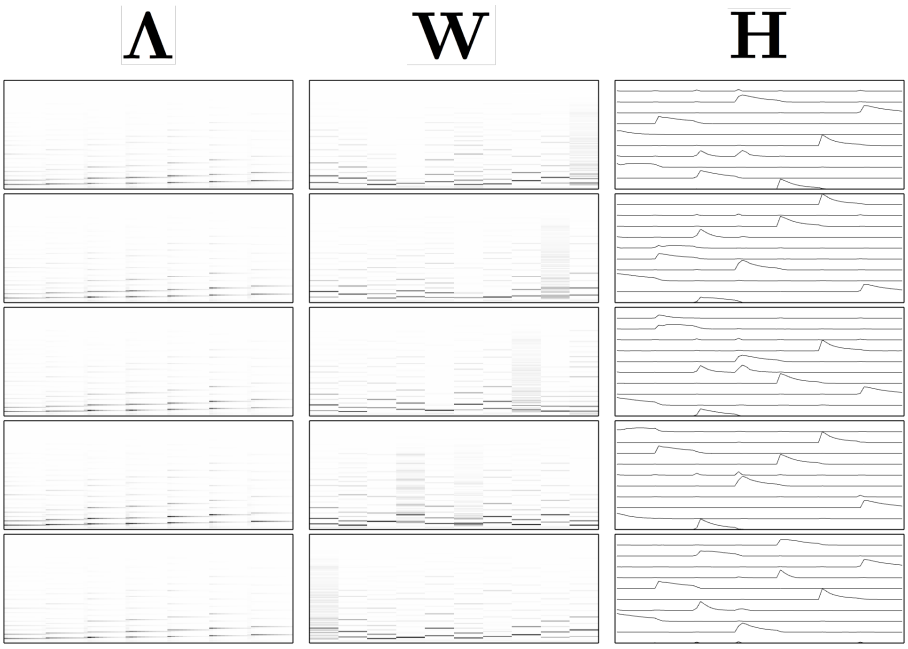

Figure 4.3: Dictionary Initializations: while different initializations yield similar reconstruction matrices $\mathbf{\Lambda}$ (F x T), the ordering of the dictionary matrices $\mathbf{W}$ (F x D) and coefficient matrices $\mathbf{H}$ (D x T) are completely different.

the frequencies' heights. As well, since the updates are multiplicative, only the non-zero values are modified during updates. This can be interpreted as providing the algorithm with the expected pitch information, leaving it to learn the timbre information. For a comparison of a randomly initialized dictionary and a harmonically initialized dictionary, see figure 4.4.

In figure 4.5, we show the results of the same five coefficient initializations as in figure 4.3, this time using harmonic dictionary initialization. As can be seen in the coefficient matrix, the five initializations result in very similar dictionary and coefficient matrices, and the general structure of the ascending scale is evident in each coefficient matrix.

In addition to providing qualitative interpretability of the learned dictionaries, harmonic initialization provides a simple way to interpret the dictionary elements quantitatively, as we may label the dictionary elements as having the pitch with which they are initialized. This approach is susceptible to error however, as can be seen, for example as some of the elements do not converge to harmonic spectra, but rather to note attacks. It is also prone to what are known as octave errors. Since notes an octave apart, i.e. one being two times the frequency of the other, share half of the partial frequencies, it is possible for an initialized vector to converge to a pitch an integer number of octaves below the intended pitch. In the following section (4.4), we will investigate two alternative approaches to dictionary interpretation, one through analysis of the learned dictionary matrix, the other through analysis of the learned coefficient matrix.

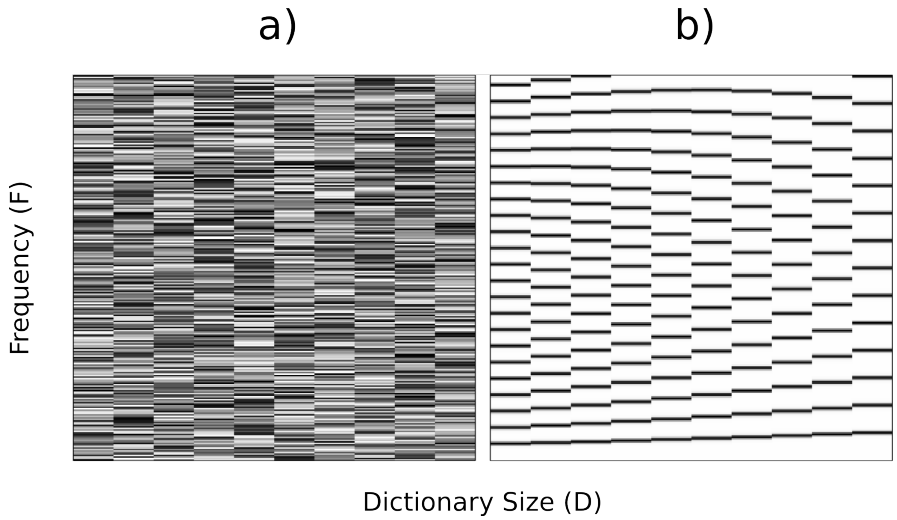

Figure 4.4: a) Randomly initialized dictionary b) Harmonically initialized dictionary

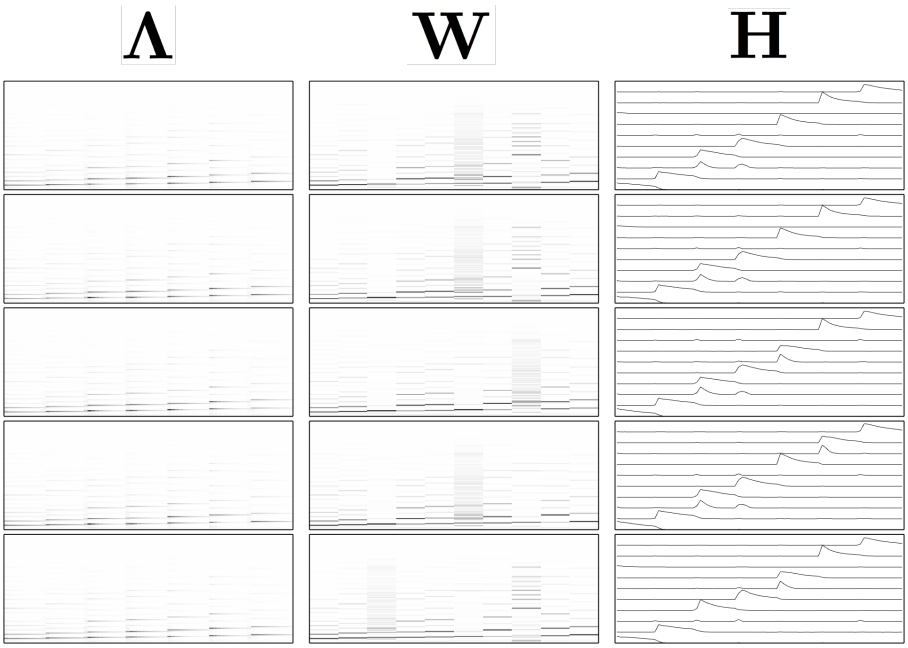

Figure 4.5: Harmonic Dictionary Initialization. The reconstruction matrices $\mathbf{\Lambda}$ (F x T) remain accurate as before, however now the dictionary matrices $\mathbf{W}$ (F x D) and coefficient matrices $\mathbf{H}$ (D x T) are qualitatively very similar. The general structure of the ascending scale is evident in each coefficient matrix.

4.4 Dictionary Labeling

As we saw in section 4.3, initializing the dictionary matrices randomly will result in an unordered dictionary after learning: there is in general no correspondence between dictionary index and pitch height. In order to be useful for the pitch detection task we will study in section 4.7, however, we will need to be able to determine a mapping from dictionary elements to pitch. The simplest is a many-to-one mapping in which we label each dictionary element as corresponding to a single note. We will refer to this as hard dictionary labeling, as we make the assumption that each dictionary element corresponds to one and only one note. An example of this kind of mapping was given above in section 4.3, where we may label the harmonically initialized dictionary prior to learning. In this section, we will demonstrate two alternative hard labeling methods with which we label dictionary elements after learning. The first approach involves analysis of the dictionary using knowledge of musical sounds, and is the commonly used technique in the literature. The second approach involves analysis of the learned coefficients, and is a novel contribution. An alternative to hard dictionary labeling is a soft labeling approach, in which the mapping from dictionary elements to notes is many to many. In this case, we forgo the assumption that each dictionary element corresponds to a single note, and adopt the assumption that different notes are in fact encoded using different distributions over the dictionary elements.

The first hard labeling technique is purely unsupervised and makes use only of knowledge of musical sounds. Based on the assumption that each learned dictionary element will represent a single note, we may compare the learned dictionary elements with an ideal dictionary whose elements are harmonic spectra of the expected notes. In the example below, the ideal dictionary only contains spectra of notes we know to be in the input, however in the general case would contain all notes within the musical range. The cosine distance similarity measure is used to compare the learned dictionary elements to the ideal dictionary elements, and since both the learned and ideal dictionary elements are normalized, this reduces to a dot product. For an example of hard dictionary labeling, see figure 4.6.

The second hard labeling technique requires knowledge of which notes are active in each frame of the input, and relies on an analysis of the coefficient matrix instead of the dictionary matrix. Unlike the analysis of the learned dictionary, this is a supervised technique. We treat each coefficient vector as measure of the likelihood that the corresponding note is present over the course of the input, and compare it with ground truth coefficient vectors to determine the most likely corresponding note. We choose to use the F-score measure to determine the most likely corresponding note. The F-score measure is defined as $2 \cdot \frac{precision \cdot recall}{precision + recall}$, where $precision = \frac{tp}{tp+fp}$, $recall = \frac{tp}{tp+fn}$ and tp is the number of true positives, fp the number of false positives and fn the number of false negatives. To use the F-score measure, we must threshold the coefficient vectors such

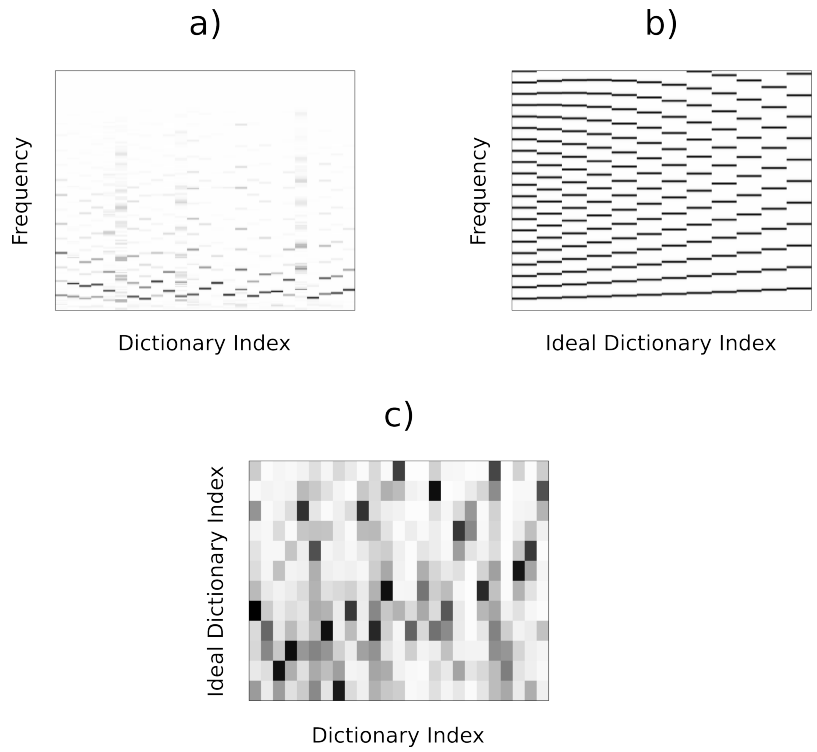

Figure 4.6: Example of hard labeling of dictionary elements using max correlation with initialized dictionary. a) Dictionary matrix b) Ideal dictionary matrix c) resulting similarity matrix

that they may be interpreted as a classification: for each frame, a value of 0 means the note is not present, while a value of 1 means it is present. Comparing the coefficient vectors in this way results in a sparse similarity matrix as can be seen in figure 4.7, which presents an example of hard labeling of the coefficient matrix.

Finally, we introduce a soft labeling technique as an alternative to the hard labeling approaches discussed above. Here, we make the assumption that learned dictionary elements do not represent individual notes, but rather that each note is modeled using a distribution over dictionary elements. We train a logistic regression model using the learned coefficients for each frame as input, and the binary ground truth note activations for each frame as the target. We expect this model to learn to predict which notes are active in a given frame, given the dictionary coefficients calculated using NMF. Figure 4.8 depicts an example of the weights learned by the logistic regression algorithm using the hidden activations seen in figure 4.7.

Once the dictionary has been labeled using one of the techniques above, the results can be used to perform pitch tracking. For the hard labeling algorithms, we take the activation of a given note to be the sum of all coefficients that correspond to that note. For the soft labeling algorithm, we take the activation of a given note to be the activation of the output unit corresponding to that note. We briefly demonstrate such results in figure 4.9, using the same input as above, however with dictionary size of 50 to demonstrate how the

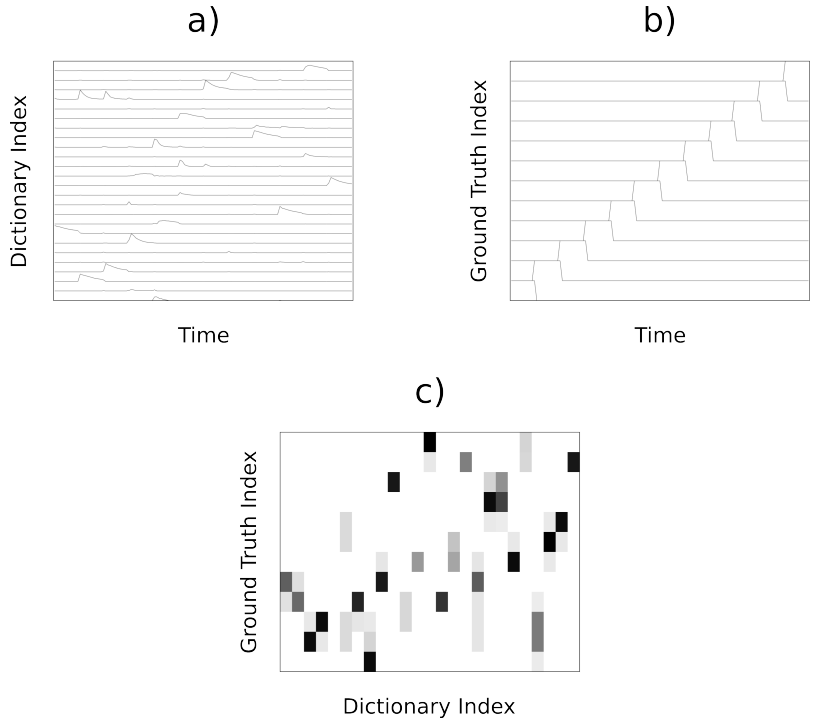

Figure 4.7: Example of hard labeling of dictionary elements using max correlation with ground truth, with activation threshold percentage of 0.01, for a threshold value 0.077.

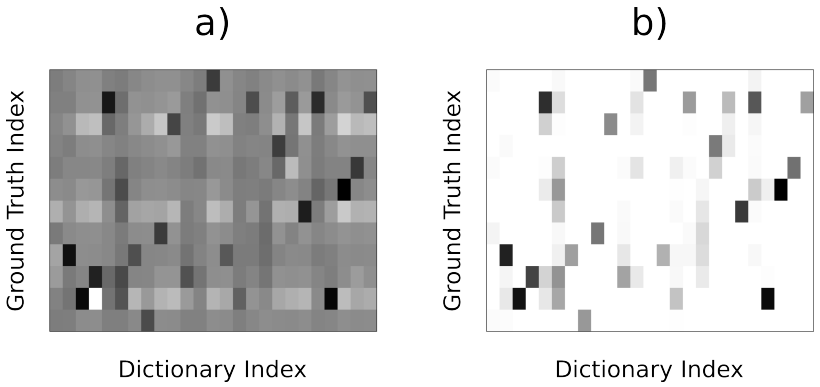

Figure 4.8: Example of soft labeling using logistic regression. a) weights normalized such that smallest weight is white, largest weight is black. b) same as in a), but with negative weights set to zero.

dictionary elements are combined by the hard labeling algorithms. We may note that the two hard labeling techniques are qualitatively similar, while the logistic labeling technique results in activations that are almost binary. When we will apply these techniques to the pitch classification problem in section 4.7, all activations will be thresholded to yield binary classification results. These threshold will be determined by maximizing the F-score on a validation set. We may also note that all algorithms seem to suffer from confusion at note boundaries, in all likelihood due to our decision to sacrifice temporal resolution for increased spectral resolution. This issue could be addressed by decreasing the window size or by decreasing the step size as discussed in section 2.3.3.

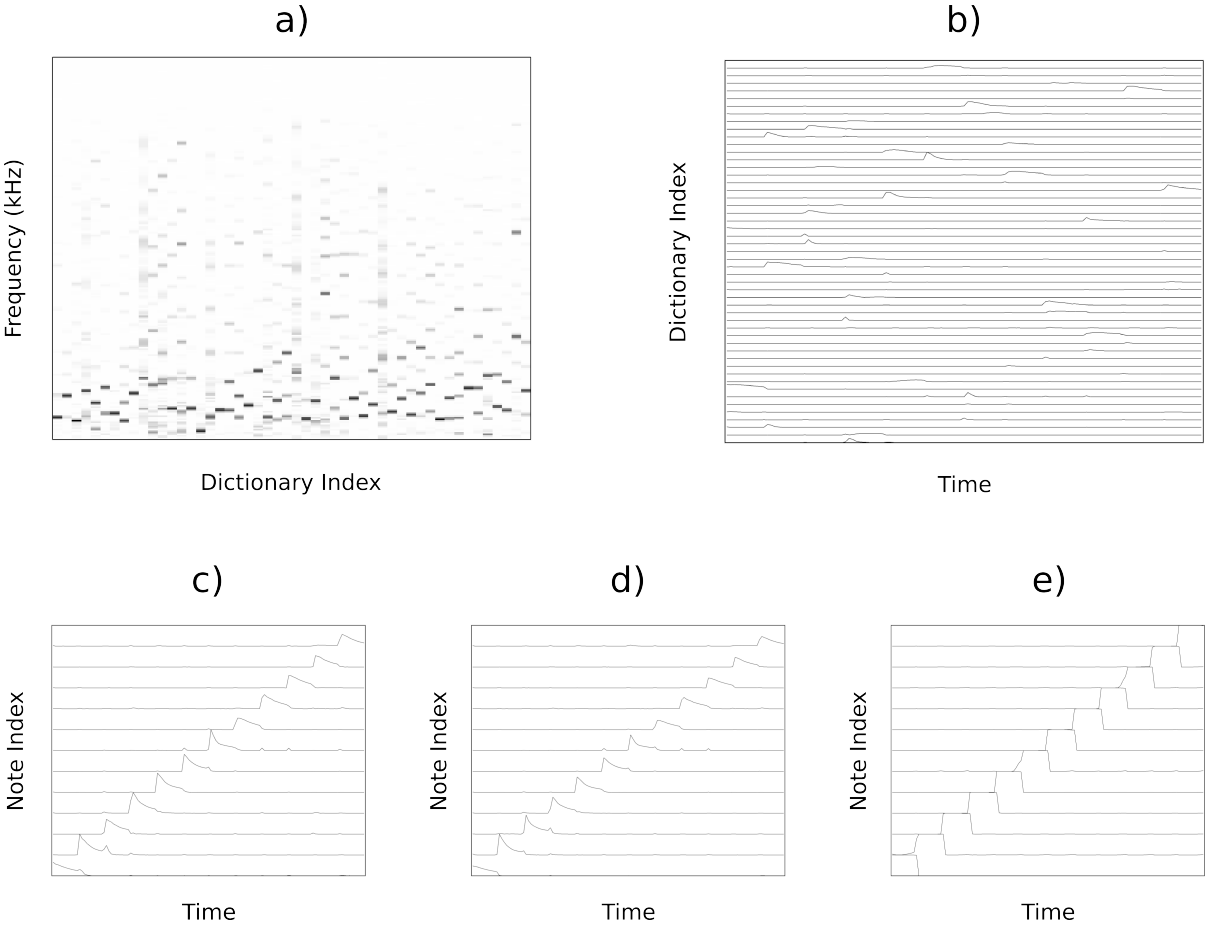

Figure 4.9: Activations of different labeling algorithms a) dictionary matrix b) coefficient matrix c) hard dictionary labeling activations d) hard coefficient labeling activations e) logistic activations

4.5 Dictionary Size

We will now investigate the effect of dictionary size on optimization of the NMF algorithm, as well as on the information modeled by the learned dictionaries. In figure 4.10 below, we show the progression of the three

cost functions for increasing dictionary size.

As we increase the dictionary size, we note that the number of epochs necessary before convergence also increases. This is to be expected as the total number of parameters, $F \cdot T \cdot D^2$, increases with dictionary size D. However, the increase in dictionary size seems to result in a smaller and smaller increases in number of epochs. The dictionary sizes presented are {6, 8, 12, 25, 50, 100}, where the increases between subsequent graphs grows smaller and smaller.

Figure 4.11 shows histograms of final costs for the same dictionary sizes as above. We note that the final costs after optimization also decrease with dictionary size. This is to be expected as increasing dictionary size results in a model with increased capacity, allowing the model to more precisely model the input data.

Finally, we take a qualitative look at the kind of information that is modeled as we increase the dictionary size. Let us first consider the monophonic case depicted in figure 4.12. For dictionary sizes less than the number of notes, we note that multiple notes are modeled by single dictionary elements, however this is not of interest in terms of interpretability. As we increase the number of dictionary elements beyond the number of notes, we note two interesting phenomena. First, a single note may be modeled by more than one dictionary element, for example one dictionary element may be responsible for the beginning of the note, while another may be responsible for the end. Thus the algorithms are able to model notes' temporal progressions using multiple dictionary elements to encore a single note. Second, we note that dictionary elements may correspond to the notes' percussive attacks or onsets. These attacks are short in duration, and the same dictionary element may be used to encore an attack, independent of pitch. As the number of dictionary elements grows significantly larger than the number of notes, we may note attacks particular to certain pitch, i.e. the attack dictionary elements become more specialized and only model the onsets of certain notes. Another phenomenon that occurs with large dictionaries is that a larger percentage of the dictionary elements is used to encode transient attacks.

In comparing the dictionaries learned by the different cost functions, we note that the tendency of the algorithm to use dictionary elements to encode transients versus steady states differs significantly, increasing from LS to KL to PA. For the largest dictionary (100), the PA algorithm uses approximately half of the elements to encode transients.

We will now look at the dictionaries learned on polyphonic audio, i.e. a harmonized A major scale, depicted in figure 4.13. We first train the models as before, but on polyphonic input data. The resulting dictionary matrices are displayed in the left columns, however plotting the resulting coefficients would not be easy to interpret due to the polyphonic nature of the input. Therefore, we use the NMF-style inference described in the following section (4.6), to encode the monophonic audio from the previous example. This way, we are able to interpret the function of the dictionary elements as before. We may note that a significant

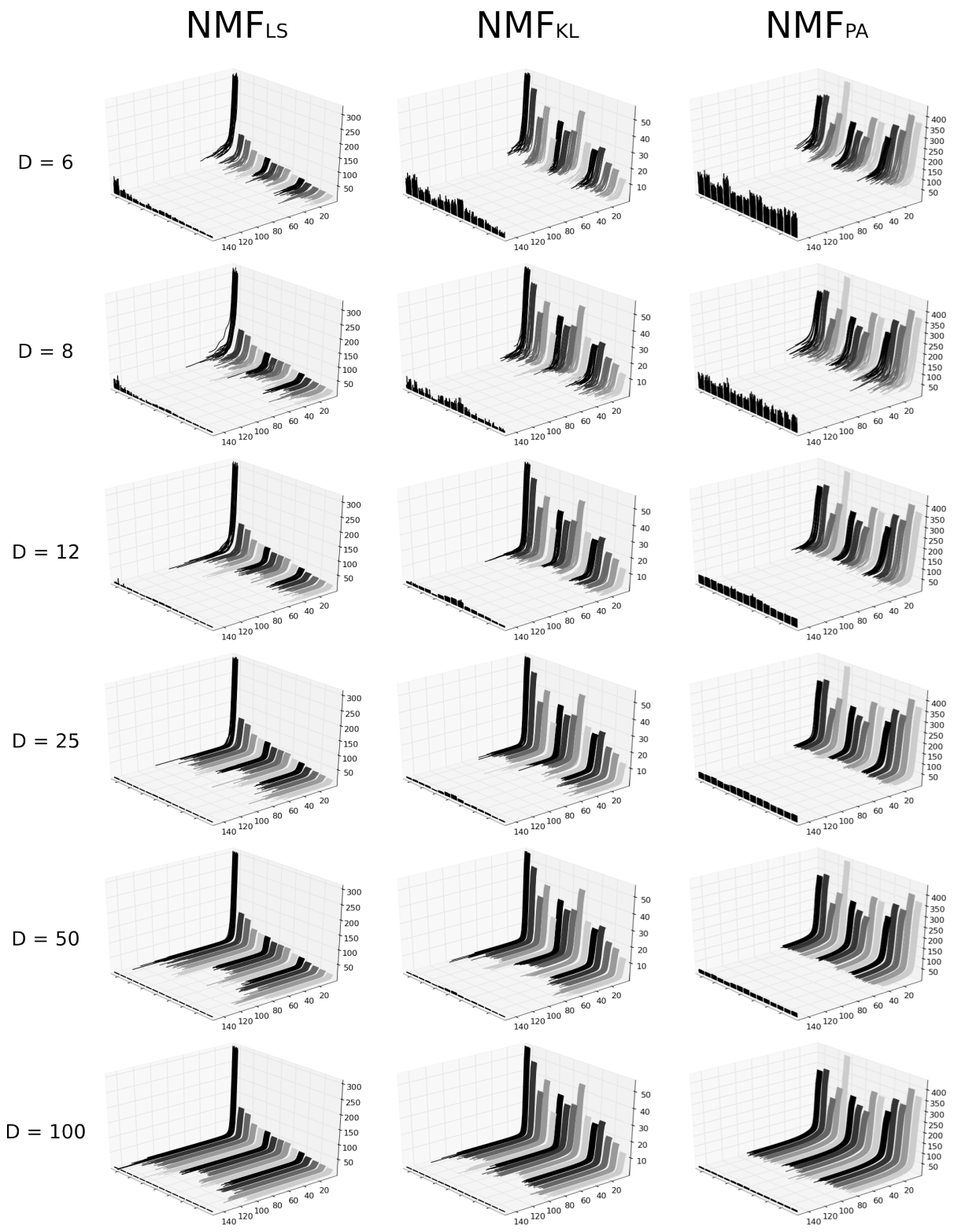

Figure 4.10: Evolution of cost over the course of optimization for various dictionary sizes. Columns for each cost function, vertical axes represents appropriate cost. Indexed horizontal axis represents epoch index. Dictionary size indicated on the left. See Figure 4.1 for a legend of the instruments used.

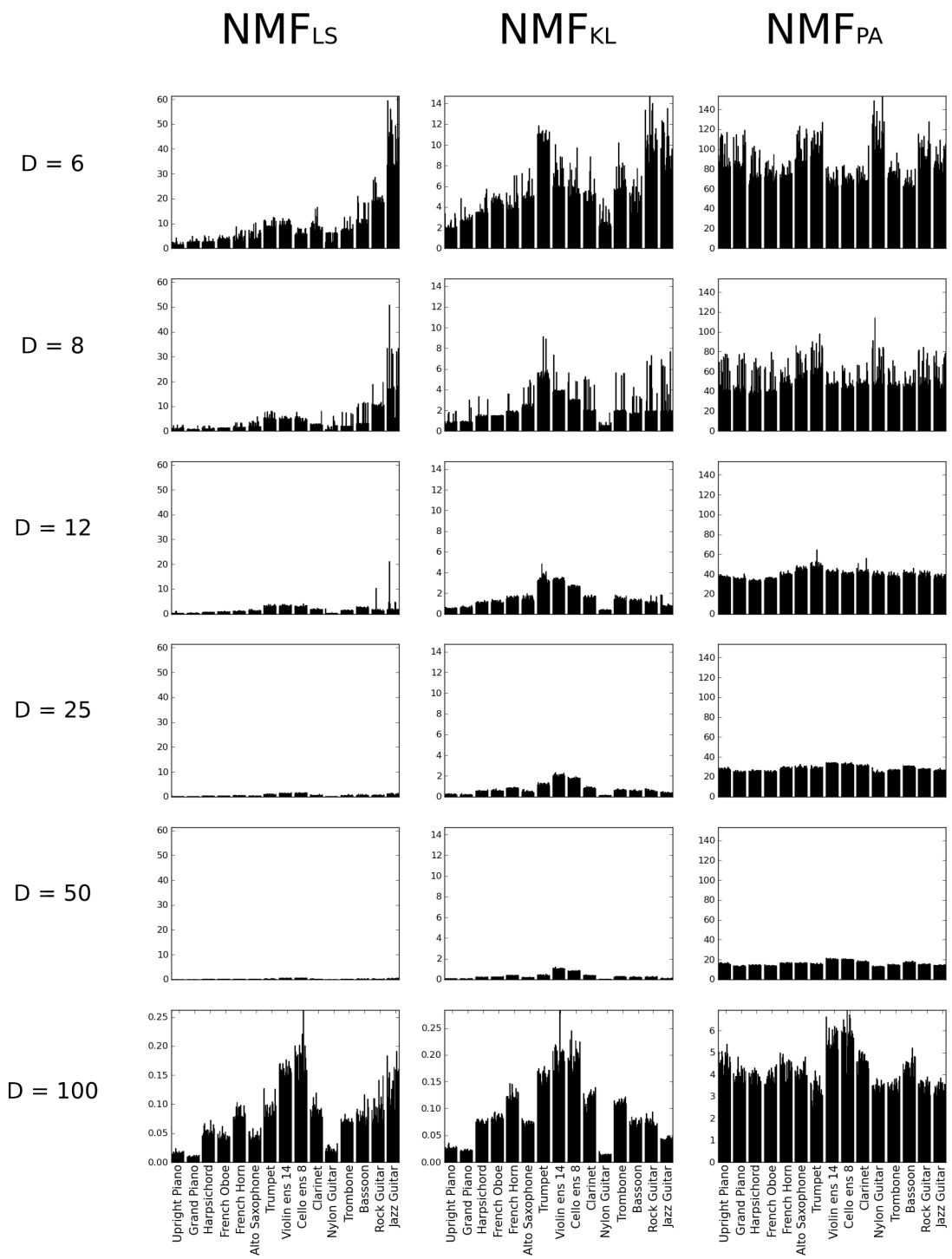

Figure 4.11: Final error histograms for the three optimizations. Note that the scale for $D = 100$ is enlarged to show differences between instruments.

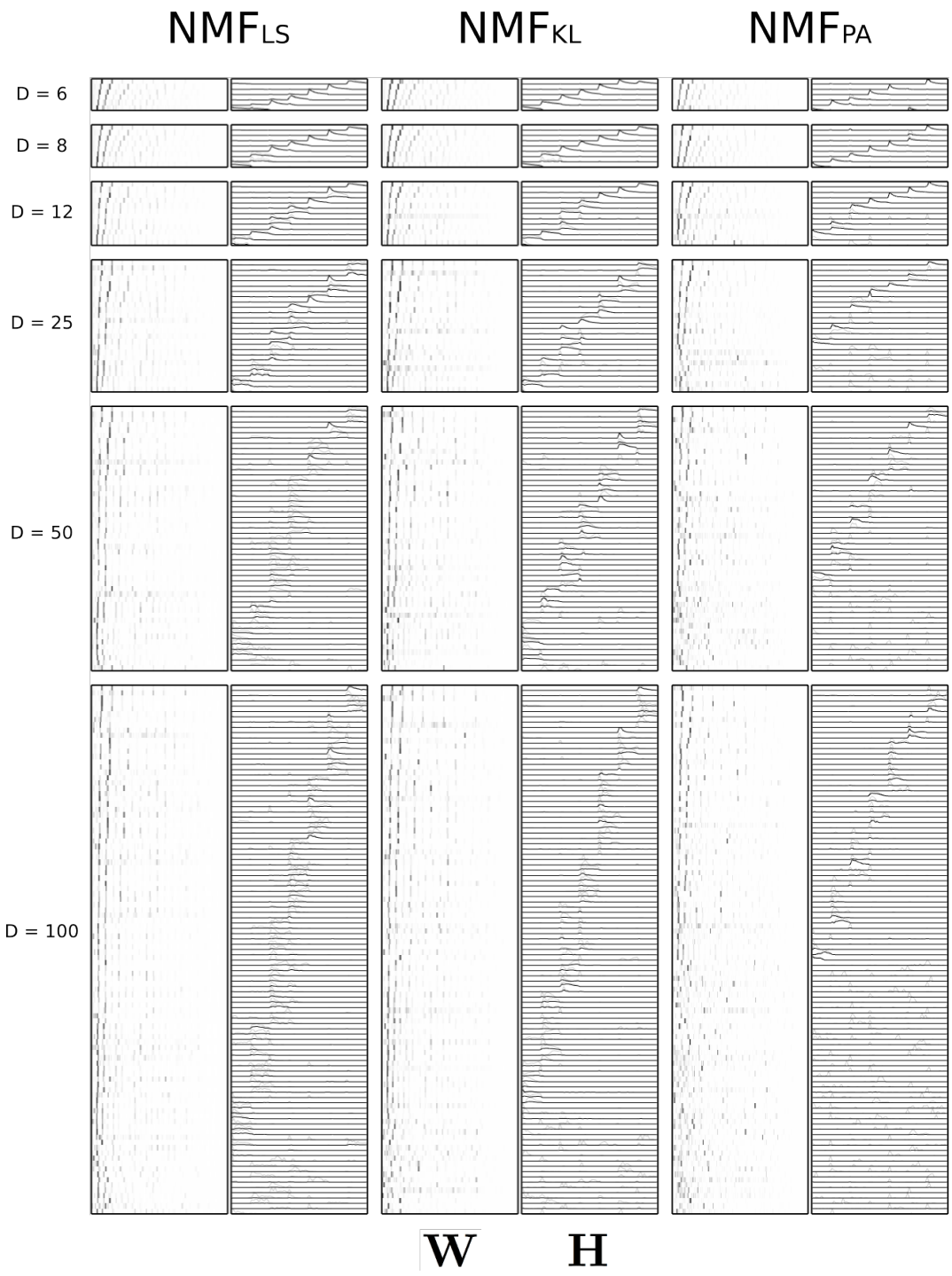

Figure 4.12: Example showing dictionary interpretability on monophonic data, for different dictionary sizes and each cost function. Input consists of an ascending single-octave A major scale, played by the Grand Piano instrument.

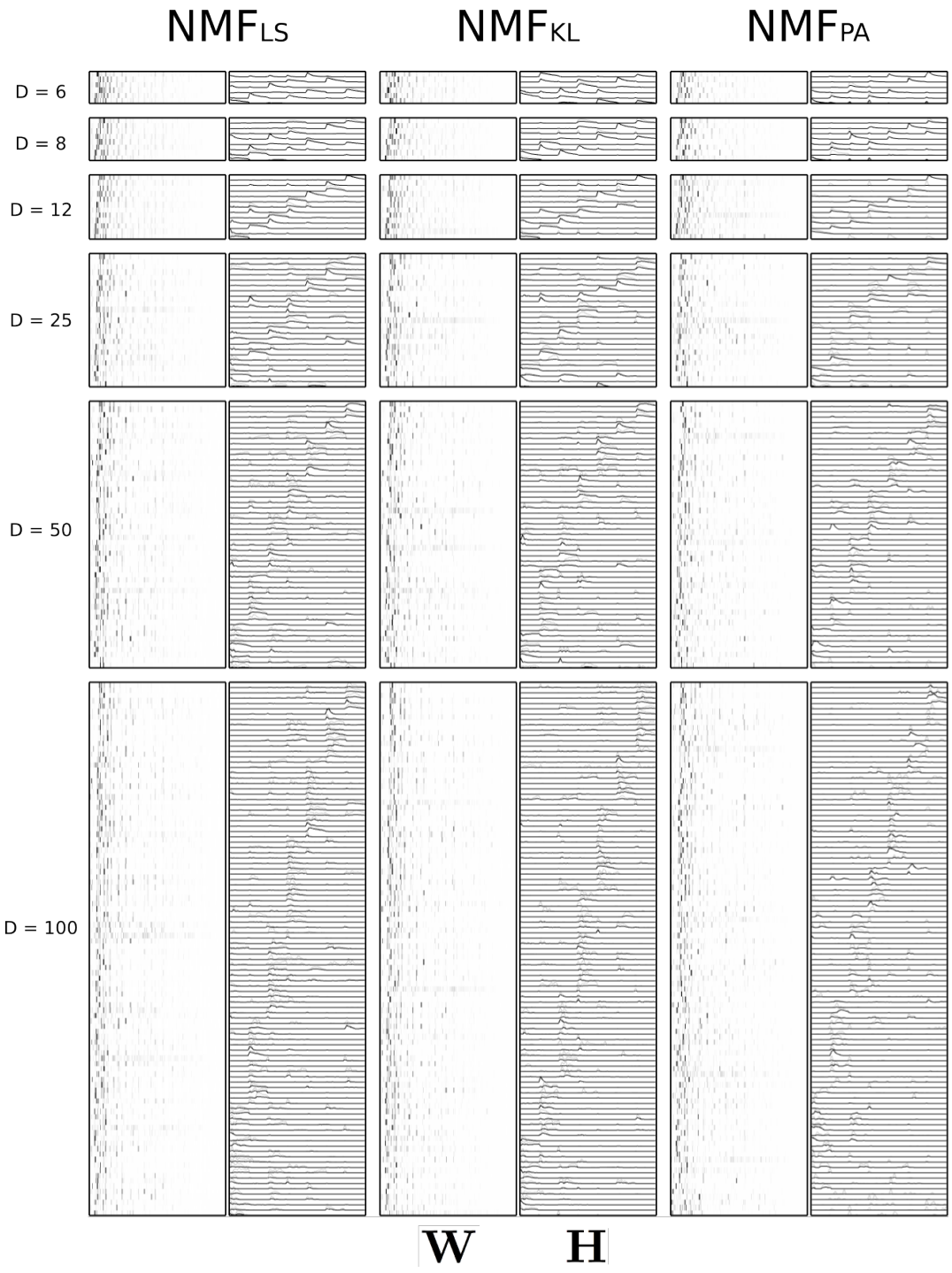

Figure 4.13: Example showing dictionary interpretability on polyphonic data, for different dictionary sizes and each cost function. Input consists of an ascending single-octave harmonized A major scale (see figure 4.20), played by the Grand Piano instrument.

number of the dictionary elements seem to be used to encode several different notes, unlike in the monophonic case where most basis functions were only active for one note.

4.6 Inference

Up until this point, we have been focused on simultaneously learning the dictionary matrix $\mathbf{W}$ and the corresponding coefficients matrix $\mathbf{H}$. In section 4.7, however, we will be concerned with the capacity of the dictionaries learned by these algorithms to be used to encode new input data. This is central to the idea of generalization: we wish to learn dictionaries that will allow us to accurately encode unseen, out-of-sample data. We must therefore be able to determine the coefficient matrix $\mathbf{H}$ for a novel input matrix $\mathbf{V}$, given a learned dictionary matrix $\mathbf{W}$. In this section, we investigate two methods for doing so.

The first method involves using the pseudoinverse of the dictionary $\mathbf{W}$, i.e. $\mathbf{W}^+$. Given the equation for the reconstruction of the novel input matrix, $\mathbf{V} \approx \mathbf{WH}$, we may approximate $\mathbf{H}$ using $\mathbf{H} \approx \mathbf{W}^+\mathbf{V}$. The second method uses a similar technique to NMF by initializing the W matrix to the learned dictionary, and repeatedly updating the H matrix according to the update rules 3.5, 3.7, and 3.11. The W matrix remains unchanged during this process, and the H matrix to converges to a local minimum of the appropriate cost function. We will now present results of these two inference techniques. In order to be able to compare the results against the "ideal" coefficients, we will apply the two approaches above to the results of a standard optimization, i.e. we learn $\mathbf{W}$ and $\mathbf{H}$ for a given $\mathbf{V}$, and then use the two inference techniques above to learn a new $\mathbf{H}$, given the same $\mathbf{V}$. This differs from the approach we will use in practice, where we will use the inference techniques to learn an $\mathbf{H}$ for a new $\mathbf{V}$, using the learned $\mathbf{W}$. Figure 4.14 displays the results of the two inference techniques for each of the three cost functions, using a dictionary size of 15. For each cost function, we show the true $\mathbf{H}$, $\mathbf{H}$ found using the pseudo-inverse method, and $\mathbf{H}$ found using the NMF update function.

The two inference techniques result in coefficients that capture the general shape of the ideal $\mathbf{H}$, as can be seen as all the coefficients follow the same general scale pattern. However, there an important exception with the pseudo-inverse method. Since the pseudo-inverse does not have a non-negativity constraint, coefficients may take on negative values. This can be seen in the second columns under each cost function, and is most prominent in the case of the PA cost function. While this method may find coefficients that yield low reconstruction error, the coefficients may loose the interpretability provided by the non-negativity constraint. One option to work around this is to set all negative values of the coefficient matrix to zero, however, as we will see in the following example, this may result in significant error. To get a better idea of the approximation errors the two inference methods make, we show the squared reconstruction errors between the approximated

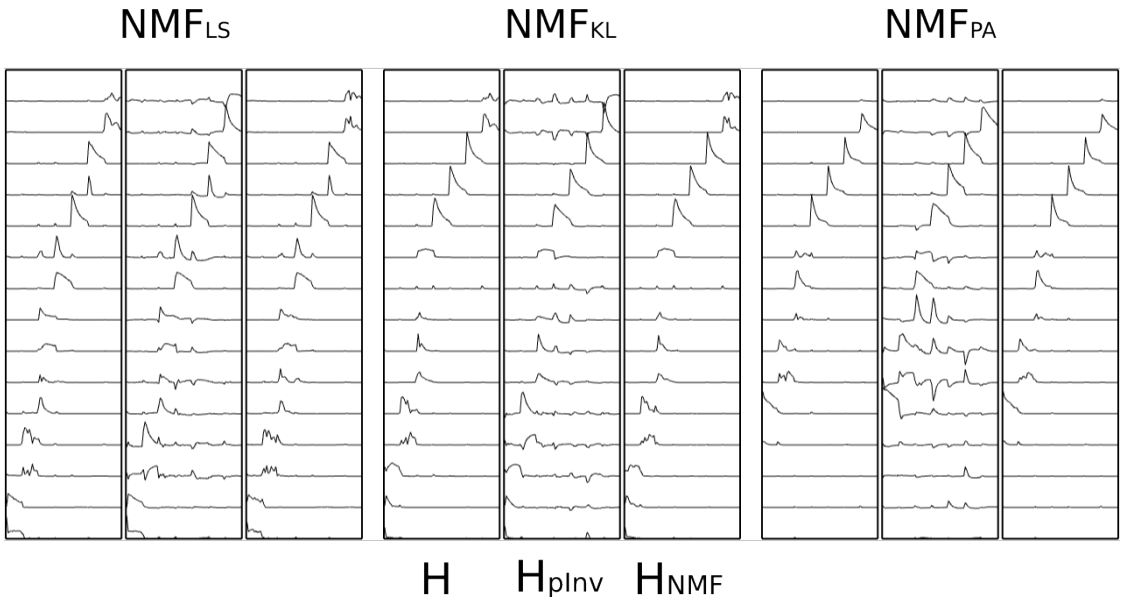

Figure 4.14: Inference results for each cost function, using the two inference techniques (pseudo inverse and NMF). Left column contains the ideal coefficients.

and ideal coefficient vectors in figure 4.15.

We note that both inference techniques result in large errors that are isolated in time. The errors made using the NMF technique appear to come primarily in groups of two or more coefficients, whose errors occur simultaneously and have the same general shape. It seems that since the dictionary matrix $\mathbf{W}$ contains multiple basis vectors for a single note, the algorithm may use various combinations thereof to achieve the same result. While this would result in noticeable errors between the estimated and ideal coefficients, the reconstruction of the input would still be accurate. For the pitch classification experiments in the following section (4.7), we will sum the activity of dictionary elements that correspond to the same note, therefore the grouped errors should not significantly affect the results. For this reason, a simple quantitative comparison of the inference errors would not suffice to determine the best technique for the pitch detection tasks. An alternative quantitative comparison in which the coefficient activities of similar basis vectors are summed prior to error calculation would be much more accurate, and is left to future work. We proceed with the assumption that grouped errors have a significantly smaller impact on the pitch detection scores than non-grouped errors, and note that while the pseudo-inverse technique also results in such grouped errors, it also results in noticeably more non-grouped errors. For this reason, we will use the NMF technique exclusively for the experiments in section 4.7. Finally, we note that while the NMF method is significantly less computationally efficient than the pseudo-inverse method, as it requires an iterative optimization procedure as opposed to a single matrix-matrix multiplication, its accuracy is considered to be more important than its computational complexity.

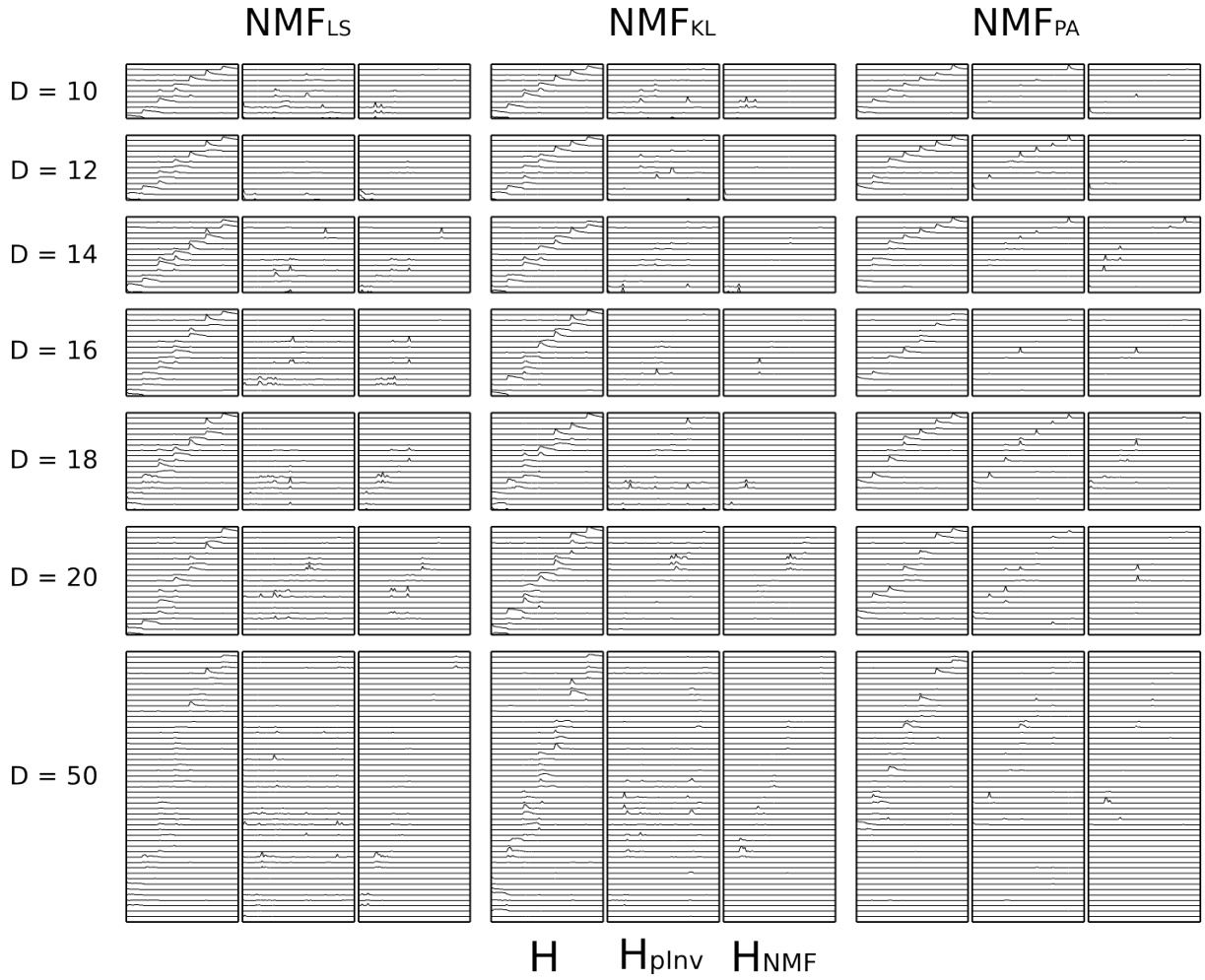

Figure 4.15: Comparison of squared errors between the ideal coefficients and inferred coefficients for each cost function. The left columns depict the ideal coefficient matrices $\mathbf{H}$, while the center and right columns depict the reconstructed coefficient matrices using the pseudo-inverse and NMF methods respectively.

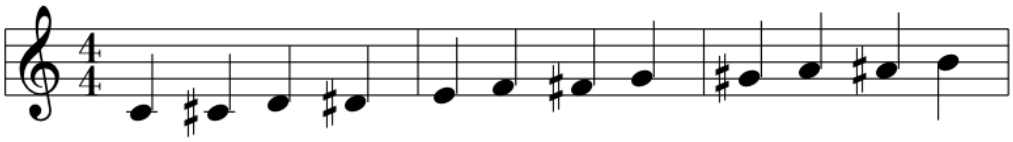

Figure 4.16: Musical notation of monophonic data used in section 4.7.1.

4.7 Musical Parameters

In this section we will analyze the ability of the learning and labeling algorithms to generalize across three factors of variation common in musical data: note duration, note velocity and instrument type. The task is to predict which pitches are present in the input, given a single spectrogram frame. The target is a binary vector with each element representing a note, where a value of 1 implies that the pitch is present in the input, and a value of 0 implies that it is not. We will use the F-score error measure to compare the different algorithms and labeling types. Since the model outputs are not binary, they must be thresholded to be interpreted as a classification. The threshold value is optimized in order to maximize the F-score performance on the validation set. We will first analyze the results on monophonic data (a chromatic scale starting on C, see figure 4.16), followed by polyphonic data (a harmonized A major scale, see figure 4.20).

4.7.1 Monophonic Data Results

4.7.1.1 Velocity

The first musical parameter we study is note velocity, i.e. how hard or soft the note is played. This is an important parameter, due to the fact that a note's timbre differs depending on how loud it is played, i.e. the spectrogram of a note played loudly is not simply a multiple of the spectrogram of the same note played softly. We wish to determine if the algorithms are able to generalize to notes played at a different volume than notes that were seen during training. We generate an input matrix for each of the 20 instruments, which consists of chromatic scales (see figure 4.16) played at 100 different amplitudes (MIDI velocities 25-124 inclusive). The training set contains velocities 45 to 64, a quiet test set contains velocities 25 to 44 and a loud test set contains velocities between 105 and 124. Velocities between 65 and 104 are split randomly into a validation set for hyperparameter selection, and parallel test set, this test set will allow us to determine how sensitive the algorithm is to this choice of validation.

Figure 4.17 shows the classification performance of 10 different algorithms: the three NMF variants (LS, KL, PA), each with the three labeling algorithms (dictionary hard labeling, coefficient hard labeling, and logistic soft labeling), and finally a logistic regression applied directly to the input. Coefficient labeling on the KL activations outperforms all other labeling and cost functions on all datasets including train,

validation and test sets. However, not all such comparisons are statistically significant. Larger datasets with more instruments may be used in future tests to decrease the variability in the results. It is particularly interesting to note that this hard labeling approach results in better classification results than the supervised logistic regression soft labeling approach, as well as logistic regression applied directly to the input. It is also interesting that the PA cost function results in noticeably lower performance than the others, and that there is significant variability in these results. This may be related to the behaviour observed in section 4.2, where we noticed that the PA cost diverged for certain instruments as we optimized LS. Here, it appears as though optimization of PA is sensitive to the input instrument as well. Further experiments are necessary to determine the nature of this dependence, and are left to future work. In general, the algorithms generalize well to unseen note velocity values.

4.7.1.2 Duration

The second music parameter we study is note duration. Notes' temporal evolutions are not static, for example as different frequencies decay at different rates due to the acoustic properties of the instrument. The spectrum of the end of a long note may therefore be very different than the middle of a short note. The train set contains chromatic scales played with notes of duration 250 and 500, while the validation set has duration 1000. Finally, we have two test sets: a short note test set with duration 125, and a long note test set with duration 2000.

Figure 4.18 demonstrates the classification results of the same 10 algorithms as above, on the different duration datasets. First, we note that the performance on the first test set, i.e. that which contains short notes, is noticeably lower than all other test sets. This is due to the large percentage of examples that are affected by the confusion introduced at the note boundaries by the low temporal resolution of the preprocessing, however it is interesting to note that the logistic soft labeling algorithm handles this case better than the other labeling algorithms. Again in our duration test sets, the KL cost function is the winner, although not always statistically significantly so, using logistic soft labeling in the short note test set, and hard coefficient labeling in the long note test set.

4.7.1.3 Instrument

Finally, we will study how well the learning and labeling algorithms are able to generalize across instruments. Instrument timbres vary significantly, and it is therefore of interest to determine how well a dictionary learned using one set of instruments may be used to encode other instruments. The following train, validation and test sets were generated randomly: train {French Horn, Grand Piano, Rock Guitar, Upright Piano, French Oboe, Violin Ensemble, Nylon Guitar, Trumpet}, validation {Clarinet, Alto Saxophone}, and test {Harpsichord,

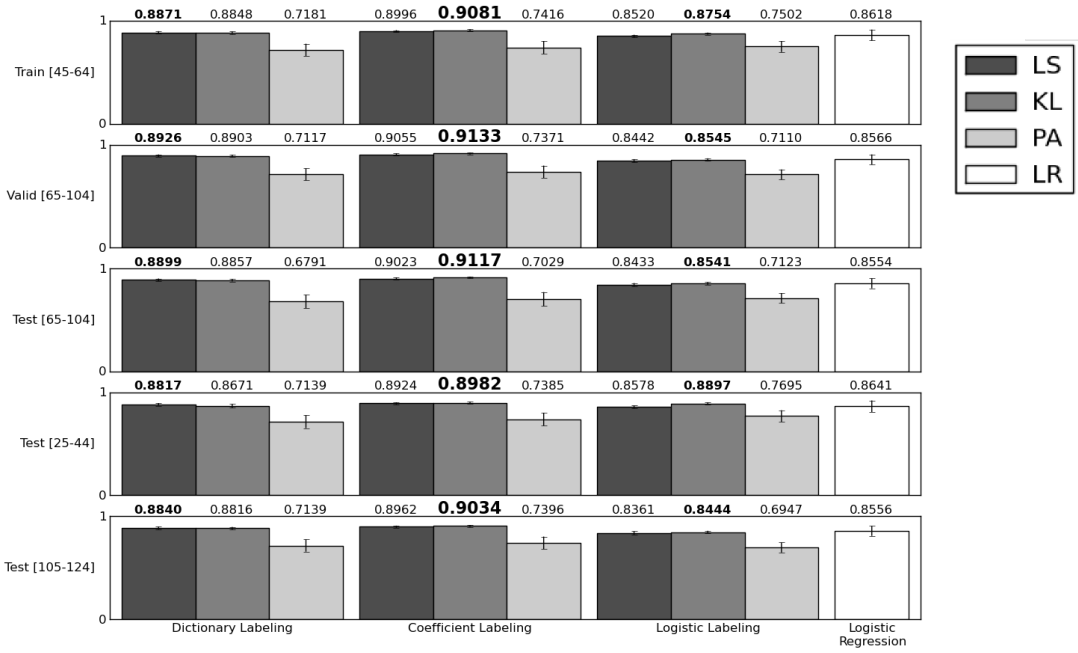

Figure 4.17: Mean F-score ± standard error values for monophonic velocity task, for the different NMF algorithms, and different labeling algorithms.

Cello Ensemble}.

Figure 4.19 depicts the classification results on the instrument datasets. First, we note that overall, the performance is significantly worse than on the velocity and duration test datasets, even on the training set. This is to be expected, however, as we are trying to model a much larger portion of the musical space. We note that the PA algorithm performs very poorly, again showing its sensitivity to the type of input instrument. Here the hard dictionary labeling algorithm using the LS cost function comes out on top.

4.7.2 Polyphonic Data Results

We will now shift our focus to the performance results of the same three factors of variation presented in the previous section (4.7.1), this time using polyphonic data. For these experiments we use a harmonized A major scale (see figure 4.20), with a degree polyphony of three, i.e. all frames contain three notes played simultaneously. Figures 4.21, 4.22, and 4.23 depict the results for the polyphonic velocity, duration and instrument tests respectively. The first thing we note is that the differences between the three labeling types are quite different in the polyphonic case when compared to the monophonic case. Here, the hard dictionary labeling does quite poorly on all three tasks, the hard coefficient labeling does noticeably better, the logistic labeling and logistic regression applied directly to the input significantly outperforming them. The likely

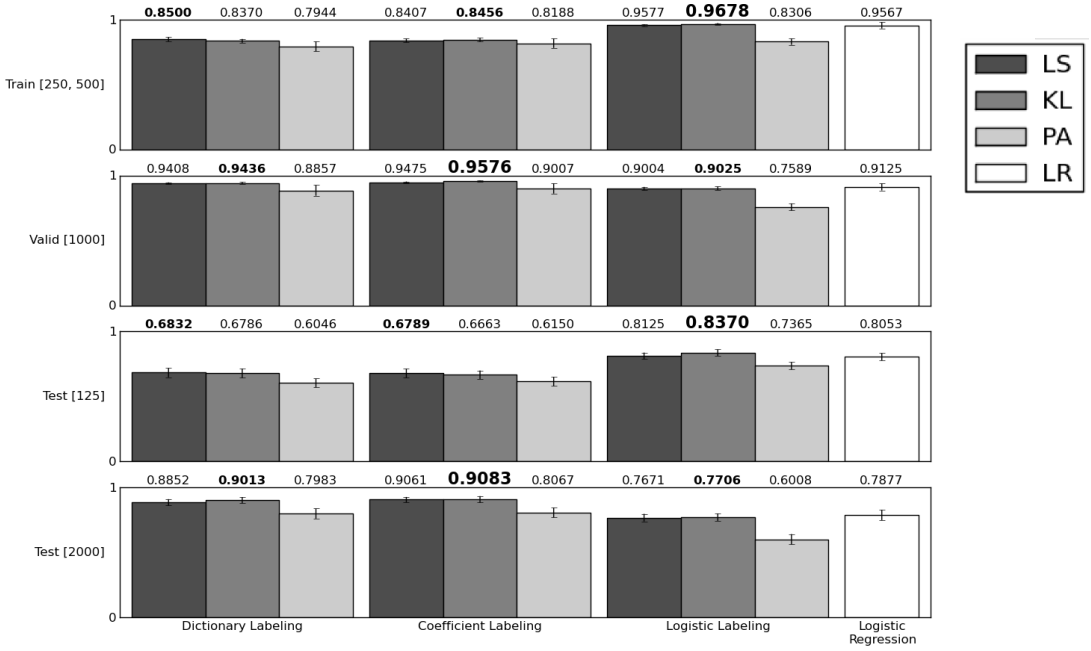

Figure 4.18: Mean F-score ± standard error values for monophonic duration task, for the different NMF algorithms, and different labeling algorithms.

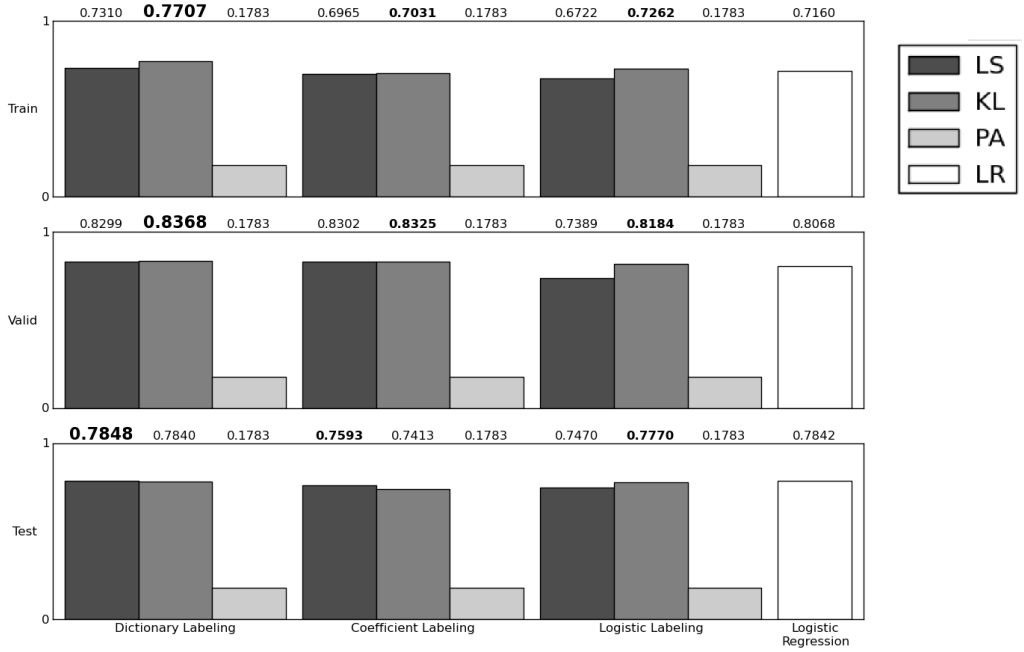

Figure 4.19: F-score values for instrument task, for the different NMF algorithms, and different labeling algorithms.

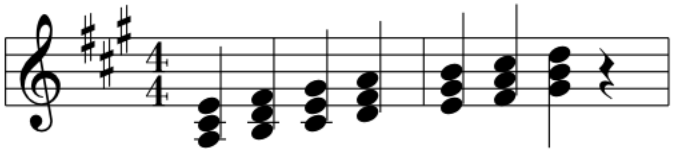

Figure 4.20: Musical notation of polyphonic data used in section 4.7.2.

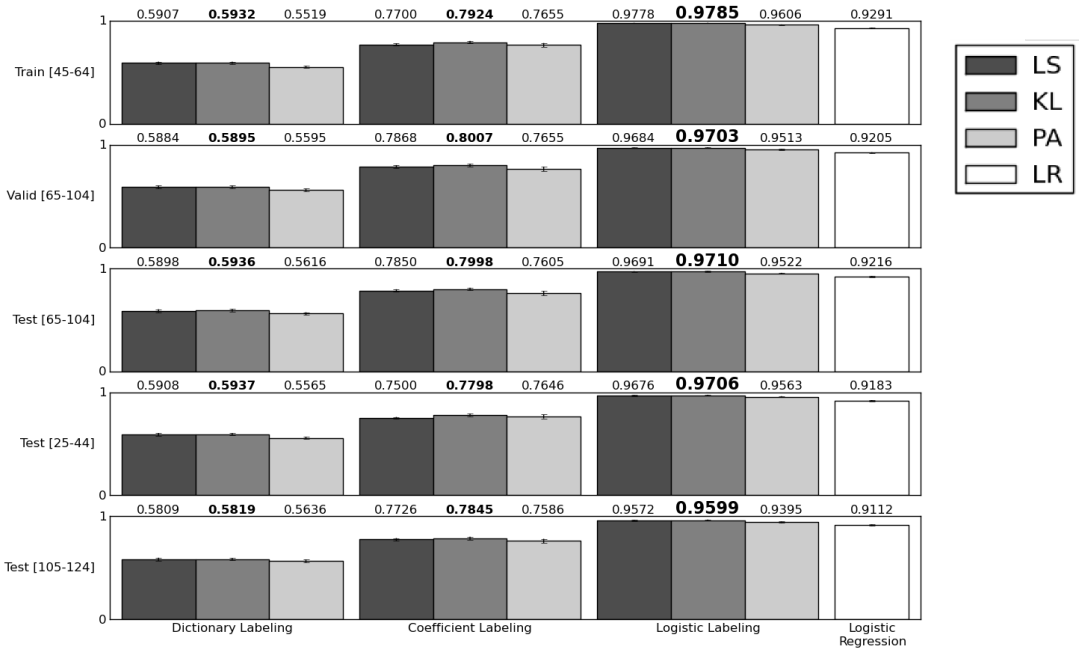

Figure 4.21: Mean F-score ± standard error values for polyphonic velocity task, for the different NMF algorithms, and different labeling algorithms.

reason for this can be traced to section 4.5 where we discussed dictionary interpretability. In the polyphonic case, we saw that the basis vectors were used to encode several notes. Since this goes against the assumption that mapping between dictionary elements and notes is one to one, it makes sense that the hard labeling algorithms would perform poorly. Second, we note that the PA cost function now performs comparatively with LS and KL, perhaps most apparent in comparing the logistic labeling results for the monophonic and polyphonic duration tasks in figures 4.18 and 4.22. We suspect this is due to the fact that in the polyphonic case, there are many overlapping partials with cancellation, bringing the data closer to the assumptions made by the PA cost function.

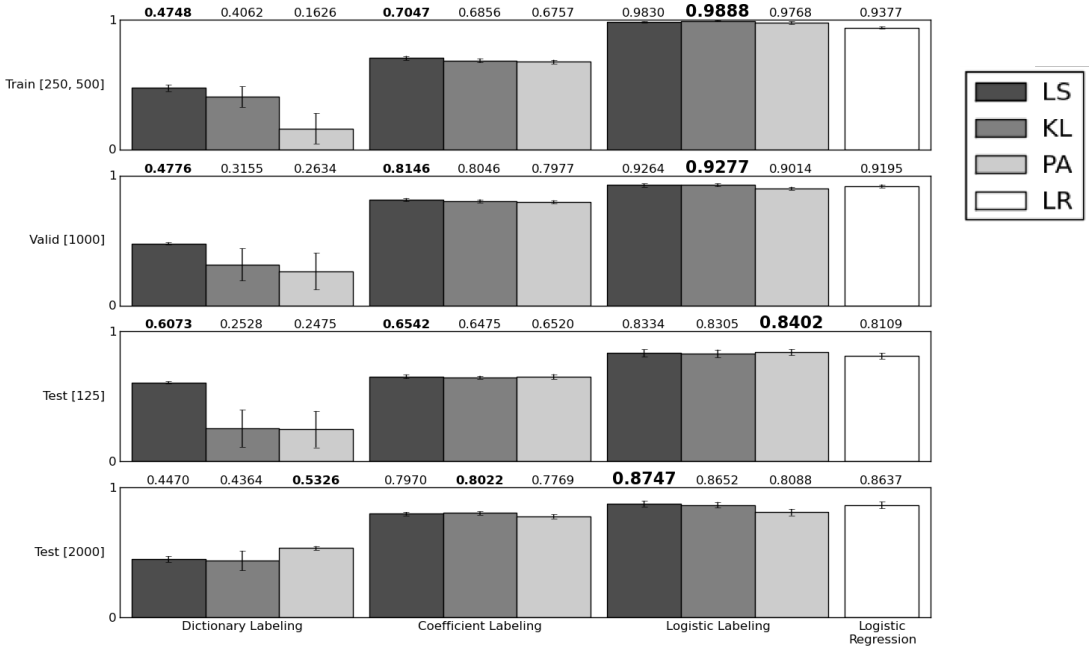

Figure 4.22: Mean F-score ± standard error values for polyphonic duration task, for the different NMF algorithms, and different labeling algorithms.

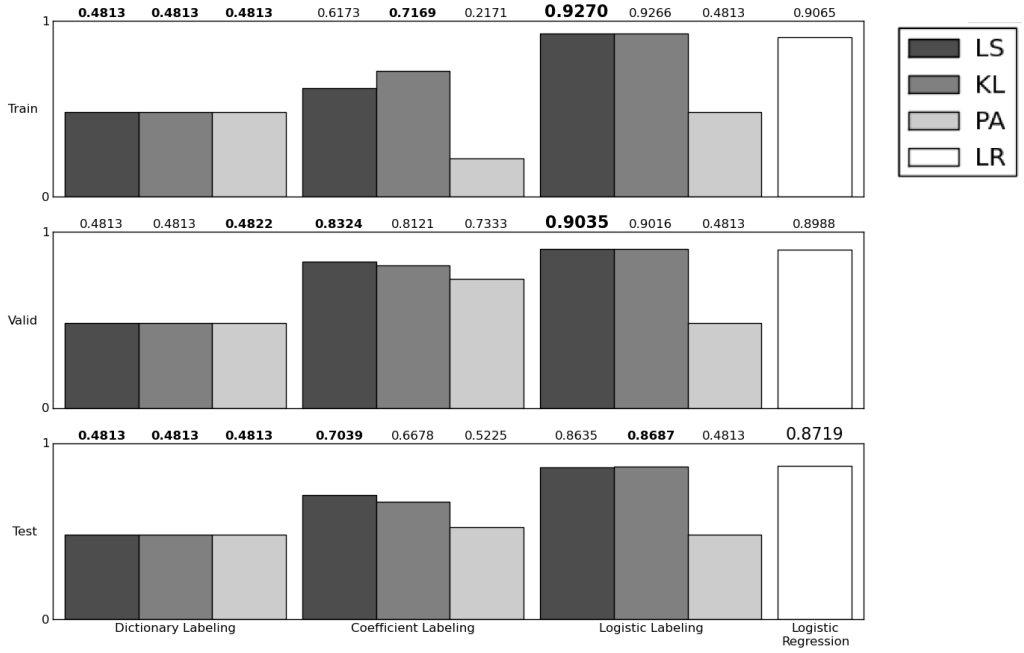

Figure 4.23: F-score values for polyphonic instrument task, for the different NMF algorithms, and different labeling algorithms.

Chapter 5

Conclusions and Future Work

5.1 Conclusions

Over the course of this thesis, we have presented the application of Non-negative Matrix Decomposition algorithms to time-frequency representations of music audio. We first discussed signal processing methods for preprocessing raw audio, transforming it into a representation suitable for the application of matrix decomposition algorithms. We then introduced the algorithms themselves, and discussed how they have been applied to music audio in the literature. We proceeded to analyze the effect of various model variants and parameters including different cost functions and dictionary sizes, on the resulting decompositions.

In the literature, decompositions have been typically analyzed using musical domain knowledge to interpret the learned basis functions, with the goal of applying the algorithms to the task of pitch detection. We introduced novel supervised dictionary labeling methods, including hard coefficient labeing and soft logistic labeling, that remove the dependence on this domain knowledge. A second constribution is the analysis of the generalization performance of the learned dictionaries, notably absent from the literature. We studied the ability of the algorithms to generalize across three common factors of variation in musical data: note amplitude, note duration and instrument type. Our novel labeling algorithms, hard coefficient labeling and soft logistic labeling, outperform the previous labeling approach (hard dictionary labeling) on the majority of our datasets (monophonic instrument generalization being the only exception).

In comparing the soft logistic labeling algorithm with a logistic regression performed directly on the input, we found the representations learned by NMF to be beneficial when used as input for both monophonic and polyphonic pitch detection tasks.

Finally, our tests allows for a comparison of the recently introduced "phase-aware" (PA) cost function with the two previously used costs, when applied to music audio. While more theoretically sound, the PA cost

function does not yield as good results as the other cost functions on our datasets. However, we found that the differences in performance between PA and the other cost functions decreased when applied to polyphonic as opposed to monophonic music.

5.2 Future Work

In recent years, there has been a significant amount of interest in the application of the algorithms we studied to music audio, primarily for the purpose of pitch extraction. We have attempted to take the first steps towards a more thorough analysis of the effects of the model variants and parameters as well as their ability to generalize to unseen data. There are many more steps to take. First, our generalization tests were done using a fixed dictionary size of 50, and while this allowed us to compare models of equal complexity, it is entirely possible that some models perform better with larger or smaller dictionaries. An analysis of how dictionary size affects the generalization performance would be an important first step. An important next step would be to increase the complexity of the MIDI data. While scales have been an excellent starting point, and have proved very useful for interpretability of the results, there are musical complexities that we have not tested as a result. Second, we have limited our analysis to mono-timbral audio, i.e. audio that contains a single instrument. Real music audio typically contains many different instruments played simultaneously, multi-track MIDI files would be a sensible starting point to generate such data. Other factors of variation should be tested including the instrument playing style, e.g. staccato, legato, etc. As well, harmonized scales using stacking intervals other than thirds, e.g. seconds, fourths, etc. would allow us to study the algorithms' abilities to generalize to new tonalities and to unseen intervals. Finally, a more thorough analysis of the phase-aware cost function when applied to music is necessary, specifically with regards to the instrument-specific behaviour noticed in sections 4.2 and 4.7, and to the difference in performance between monophonic and polyphonic audio noticed in section 4.7.

Bibliography

[1] Audio unit. http://en.wikipedia.org/wiki/Audio_Units.

[2] Core audio. http://developer.apple.com/audio.

[3] Dan ellis' log-frequency spectrogram code. http://labrosa.ee.columbia.edu/matlab/sgram.

[4] Hdf5. http://www.hdfgroup.org/HDF5.

[5] Kontakt sampler. http://en.wikipedia.org/wiki/Kontakt_(software).

[6] libsamplerate. http://www.mega-nerd.com/SRC.

[7] libsndfile. http://www.mega-nerd.com/libsndfile.

[8] numpy. http://numpy.scipy.org.

[9] pytables. http://www.pytables.org.

[10] Python. http://www.python.org.

[11] scipy. http://www.scipy.org.

[12] Soundfont. http://en.wikipedia.org/wiki/SoundFont.

[13] S.A. Abdallah and M.D. Plumbley. Polyphonic music transcription by non-negative sparse coding of power spectra. In *Proceedings of the Fifth International Conference on Music Information Retrieval (ISMIR 2004)*, pages 10–14. Citeseer.

[14] S.A. Abdallah and M.D. Plumbley. Unsupervised analysis of polyphonic music by sparse coding. *IEEE Transactions on Neural Networks*, 17(1):179–196, 2006.

[15] T. Blumensath and M. Davies. Sparse and shift-invariant representations of music. *IEEE Transactions on Speech and Audio Processing*, 14(1):50, 2006.

[16] J.C. Brown. Calculation of a constant Q spectral transform. *Journal of the Acoustical Society of America*, 89(1):425–434, 1991.

[17] C.K. Chui. *An introduction to wavelets*. Academic Pr, 1992.

[18] A. de Cheveigné and H. Kawahara. YIN, a fundamental frequency estimator for speech and music. *The Journal of the Acoustical Society of America*, 111:1917, 2002.

[19] J.L. Durrieu, G. Richard, and B. David. Singer melody extraction in polyphonic signals using source separation methods. In *Proc. IEEE Int. Conf. on Acoustics, Speech and Signal Processing (ICASSP*, pages 169–172, 2008.

[20] J. Eggert and E. Korner. Sparse coding and NMF. In *2004 IEEE International Joint Conference on Neural Networks, 2004. Proceedings*, volume 4, 2004.

[21] D. FitzGerald, M. Cranitch, and E. Coyle. Sound source separation using shifted non-negative tensor factorisation. *Update*, 1(3):1–3.

[22] D. FitzGerald, M. Cranitch, and E. Coyle. Shifted 2D non-negative tensor factorisation. In *Proceedings of the Irish Signals and Systems Conference*. Citeseer, 2006.

[23] D. FitzGerald, M. Cranitch, and E. Coyle. Extended nonnegative tensor factorisation models for musical sound source separation. *Computational Intelligence and Neuroscience*, 2008:872425, 2008.

[24] T.J. Gardner and M.O. Magnasco. Sparse time-frequency representations. *Proceedings of the National Academy of Sciences*, 103(16):6094, 2006.

[25] D. Gerhard. Pitch extraction and fundamental frequency: History and current techniques. *University of Regina Technical Report TR-CS*, 6, 2003.

[26] R. Grosse, R. Raina, H. Kwong, and A.Y. Ng. Shift-invariant sparse coding for audio classification. *cortex*, 9:8.

[27] M. Helén and T. Virtanen. Separation of drums from polyphonic music using non-negative matrix factorization and support vector machine. In *Proc. EUSIPCO*, volume 2005. Citeseer, 2005.

[28] H. Kameoka, N. Ono, K. Kashino, and S. Sagayama. Complex NMF: A new sparse representation for acoustic signals. In *Proceedings of the 2009 IEEE International Conference on Acoustics, Speech and Signal Processing-Volume 00*, pages 3437–3440. IEEE Computer Society, 2009.

[29] J. Le Roux, A. de Cheveigne, and L.C. Parra. Adaptive template matching with shift-invariant semi-NMF. In *Proc. NIPS*, volume 8, 2008.

[30] D.D. Lee and H.S. Seung. Algorithms for non-negative matrix factorization. *Advances in neural information processing systems*, pages 556–562, 2001.

[31] M. Mørup and M. N. Schmidt. Sparse non-negative matrix factor 2-D deconvolution. Technical report, Technical University of Denmark, 2006. `http://www2.imm.dtu.dk/pubdb/p.php?4116`.

[32] M. Mørup, M.N. Schmidt, and L.K. Hansen. Shift invariant sparse coding of image and music data. *Submitted to Journal of Machine Learning Research*, 2008.

[33] A.V. Oppenheim, A.S. Willsky, and S. Hamid. Signals and systems. 1997.

[34] R.M. Parry and I. Essa. Phase-aware non-negative spectrogram factorization. *Lecture Notes in Computer Science*, 4666:536, 2007.

[35] M.D. Plumbley, S.A. Abdallah, T. Blumensath, and M.E. Davies. Sparse representations of polyphonic music. *Signal Processing*, 86(3):417–431, 2006.

[36] S.A. Raczynski, N. Ono, and S. Sagayama. Multipitch analysis with harmonic nonnegative matrix approximation. In *Proc. Int. Conf. on Music Information Retrieval (ISMIR*, pages 381–386, 2007.

[37] S.A. Raczynski, N. Ono, and S. Sagayama. Multipitch analysis by harmonic nonnegative matrix approximation and hierarchical hidden markov models. Technical report, IPSJ Technical Report, 2008.

[38] M.N. Schmidt and M. Mørup. Nonnegative matrix factor 2-D deconvolution for blind single channel source separation. *Lecture Notes in Computer Science*, 3889:700, 2006.

[39] M.N. Schmidt and M. Mørup. Sparse non-negative matrix factor 2-d deconvolution for automatic transcription of polyphonic music. In *Proc. 6th International Symposium on Independent Component Analysis and Blind Signal Separation*, 2006.

[40] F. Sha and L.K. Saul. Real-time pitch determination of one or more voices by nonnegative matrix factorization. *Advances in Neural Information Processing Systems*, 17:1233–1240, 2005.

[41] P. Smaragdis. Non-negative matrix factor deconvolution; extraction of multiple sound sources from monophonic inputs. *Lecture Notes in Computer Science*, pages 494–499, 2004.

[42] P. Smaragdis and J.C. Brown. Non-negative matrix factorization for polyphonic music transcription. In *IEEE Workshop on Applications of Signal Processing to Audio and Acoustics*, pages 177–180. Citeseer, 2003.

[43] S. Sophea and S. Phon-Amnuaisuk. Determining a Suitable Desired Factors for Nonnegative Matrix Factorization in Polyphonic Music Transcription. pages 166–170, 2007.

[44] E. Vincent, N. Bertin, and R. Badeau. Two nonnegative matrix factorization methods for polyphonic pitch transcription. *Proc. Music Information Retrieval Evaluation eXchange (MIREX)*, 2007.

[45] E. Vincent, N. Bertin, and R. Badeau. Harmonic and inharmonic nonnegative matrix factorization for polyphonic pitch transcription. In *Proc. IEEE Int. Conf. on Acoustics, Speech and Signal Processing (ICASSP*, pages 109–112, 2008.

[46] T. Virtanen. Separation of sound sources by convolutive sparse coding. In *ISCA Tutorial and Research Workshop (ITRW) on Statistical and Perceptual Audio Processing*. Citeseer, 2004.

[47] T. Virtanen and A. Klapuri. Analysis of polyphonic audio using source-filter model and non-negative matrix factorization. In *Advances in Models for Acoustic Processing, Neural Information Processing Systems Workshop*. Citeseer, 2006.

[48] B. Wang and M.D. Plumbley. Musical audio stream separation by non-negative matrix factorization. In *Proc. DMRN Summer Conf*, pages 23–24, 2005.