Atomic Decomposition of Audio with High Dimensional Distributed Representations

Christian Yost

Department of Music Research
Schulich School of Music
McGill University
Montréal, Québec, Canada

April 2022

A thesis submitted to McGill University in partial fulfillment of the requirements for the degree of Master of the Arts.

Abstract

Atomic decomposition of audio describes sound as a linear combination of elementary waveforms (atoms) from which high-quality coding, source separation, and signal transformation can be achieved. When the desired output is produced using only a few atoms, the decomposition is *sparse*. Existing atomic decomposition algorithms rely on computing many correlations between a time-signal and a redundant dictionary of atoms causing them to become increasingly memory/computationally intensive as the signal length grows and/or the atoms become more complex.

In this thesis we approach atomic decomposition through the lens of *vector function architecture* (VFA), a non-traditional computing paradigm which produces a fixed dimensional representation, consequently overcoming the burden of long duration signals. Using VFA we show how to systematically encode atoms, and from these encodings build an entirely new atomic decomposition system, called Hyperdimensional Atomic Decomposition (HD-AD), that avoids time-domain correlations altogether. We show that our system scales with the sparsity of the signal, rather than its length in time. As a result, it often produces atomic decompositions much faster than real-time – a speed unknown to existing atomic decomposition methods. Encoding with a VFA involves a non-trivial decoding step which we present a solution based on Newton's method.

We show that HD-AD can also be used as a pre-processing step to traditional atomic decomposition algorithms like matching pursuit and its variants. We call this system Neural Network accelerated Matching Pursuit (NN-MP). NN-MP greatly increases the tractability of matching pursuit with asymmetric atoms, allowing it to run on machines with modest computing resources.

Résumé

La décomposition atomique des signaux audio décrit le son comme une combinaison linéaire de formes d'onde élémentaires (atomes) à partir de laquelle un codage, une séparation de source ou une transformation de signal de haute qualité peuvent être obtenus. Lorsque la décomposition souhaitée ne comprend qu'un petit nombre d'atomes, elle est qualifiée de parcimonieuse. Les algorithmes de décomposition atomique existants reposent sur le calcul de nombreuses corrélations entre un signal temporel et un dictionnaire d'atomes redondant, ce qui les rend d'autant plus gourmands en mémoire et en calcul que la longueur du signal augmente ou que les atomes deviennent plus complexes.

Dans cette thèse, nous abordons la décomposition atomique par le prisme de *l'architecture de fonctions vectorielles* (VFA), un paradigme de l'informatique non traditionnel qui produit une représentation à taille fixe, contournant ainsi le fardeau des signaux de longue durée. Utilisant la VFA, nous montrons comment coder systématiquement les atomes, et construire un système de décomposition atomique entièrement nouveau à partir de ces codages, que nous appelons *décomposition atomique hyper-dimensionnelle* (HD-AD). Cette approche évite complètement le calcul de corrélations dans le domaine temporel. Nous montrons que notre système se dimensionne plus en fonction de la parcimonie du signal à coder, que de sa durée. En conséquence, HD-AD effectue des décompositions atomiques beaucoup plus rapidement qu'en temps réel - une vitesse inconnue des méthodes de décomposition atomiques existantes. Cependant, l'encodage avec une VFA nécessite une étape de décodage non triviale pour laquelle nous présentons une solution fondée sur la *méthode de Newton*.

Nous montrons que la méthode HD-AD peut également être utilisée comme étape de pré-traitement pour les algorithmes de décomposition atomique traditionnels tels que le *Matching Pursuit* (MP) et ses variantes. Nous appelons ce système *Matching Pursuit* accéléré par

réseau de neurones (NN-MP). NN-MP accroît considérablement la propension de la poursuite à s'adapter à des atomes asymétriques, ce qui lui permet de tourner sur des ordinateurs dotés de capacités de calcul modestes.

Acknowledgements

I have had the good fortune of coming across a set of high-quality educators to advise me throughout my higher education. These are Matt Deady, Matt Sargent, and most recently Philippe Depalle. Across the past 8 years they have given me countless hours of their time: an experience that has taught me the value of time well spent. The other set of people who have always helped steer the ship are my parents. They looked out for me when my sight was too short. I am grateful for the people I have around me.

Many of the resources which made this project possible were generously provided by Compute Canada.

Contents

List of Figures

List of Tables

List of Acronyms

Notation	Description
ANN	artificial neural network.
CNN	convolutional neural network.
CPU	central processing unit.
dB	decibel.
DFT	discrete Fourier transform.
DL	deep learning.
DNN	deep neural network.
DS	damped sinusoid.
DSP	digital signal processing.
ELU	exponential linear unit.
EMG	electromyography.
FC	fully connected.
FFT	fast Fourier transform.
FOF	formant-wave-function.
FPE	fractional power encoding.
GLU	gated linear unit.

Notation	Description
GPU	graphical processing unit.
GT	gammatone.
HD	hyperdimensional.
HD-AD	Hyperdimensional Atomic Decomposition.
HDC	hyperdimensional computing.
HRR	holographic reduced representation.
KLPE	kernel LPE.
LPE	locality preserving encoding.
LS	least squares.
MAP	multiply-add-permute.
MFCCs	Mel-frequency cepstral coefficients.
ML	machine learning.
MP	matching pursuit.
MSE	mean squared error.
NM	Newton's method.
NN-MP	neural network accelerated matching pursuit.
NN-OMP	neural network accelerated orthogonal matching pursuit.
OMP	orthogonal matching pursuit.
RAM	random access memory.
REDS	ramped exponentially damped sinusoid.
ReLU	rectified linear unit.
RGB	red, green, blue.
SNR	signal-to-noise ratio.

Notation	Description
SRR	signal-to-residual ratio.
STFT	short-time Fourier transform.
TPPS	tensor product production system.
VFA	vector function architecture.
vREDS	vibrato REDS.
VSA	vector symbolic architecture.

Notation

Notation	Description
A	asymmetric atom attack envelope.
$A_{\odot}$	set of unitary HD vectors under binding operation $\odot$.
B_w	bandwidth.
C	number of hypervectors in superposition.
E	envelope.
H	Hessian matrix.
I, J	layer channels where I is input channels and J is output channels.
K	similarity kernel.
L	loss function.
M	dictionary size.
$M_{\mathbf{HD}}$	HD dictionary size.
$N_{\mathbf{FFT}}$	FFT size.
$N_{\mathbf{HD}}$	hypervector dimensionality.
Q	number atom prototype parameters.
$\%$	standard modulo operation.
α	envelope damping parameter.

Notation	Description
β	envelope attack parameter.
$\mathbf{\Phi_{HD}}$	HD matrix dictionary.
$\mathbf{\Pi}$	sub-dictionary of atoms taken from $\mathbf{\Phi}$.
$\boldsymbol{\lambda}$	set of atom parameters.
$\boldsymbol{\theta}$	hypervector of angles.
$\cdot_{\mathbf{HD}}$	HD version of operation/variable $\cdot$.
$\circ$	generic binding.
$\circledast$	circular convolution binding.
δ	envelope difference measure.
$\dot{\lambda}$	encoding of atom parameter λ.
$\dot{g}$	parameter encoding map.
ϵ	noise/error.
$\gamma_\circ(x)$	function which maps x to HD space for binding operation $\circ$.
λ	atom parameter.
$\mathbb{C}$	complex numbers.
$\mathbb{R}$	real numbers.
$\mathbb{Z}$	integers.
$\mathbf{\Phi}$	matrix dictionary.
$\mathbf{r}^{(k)}$	residual signal at iteration k.
$\mathbf{s}(\boldsymbol{\lambda})$	HD encoding of atom $\phi_{\boldsymbol{\lambda}}$.
$\mathbf{x}$	weight vector.
$\mathbf{y}$	audio signal.
$\mathbf{z}$	HD encoding of audio signal $\mathbf{y}$.

Notation	Description
$\mathcal{D}$	set dictionary.
$\mathcal{F}$	Fourier transform.
$\mathcal{F}^{-1}$	inverse Fourier transform.
μ_λ	maximum value for range of parameter λ, alternatively $\max(\lambda)$.
$\odot$	Hadamard binding.
ω	angular frequency.
$\overline{\mathbf{v}}$	complex conjugate of $\mathbf{v}$.
ϕ	atom.
σ	sigmoid activation function.
τ	window time shift parameter.
$\mathbf{ds}_k$	down-sampling at layer k.
$\mathbf{sr}$	sampling rate.
$\tilde{\mathbf{u}}$	approximate inverse of $\mathbf{u}$.
b	base for exponential HD parameter mapping. $b = 2000$.
f	normalized frequency.
$f_{\mathbf{FPE}}$	fractional power encoding map.
f_m	vibrato rate or frequency modulation.
i	imaginary unit, $\sqrt{-1}$.
n	discrete time.
n_I	influence time.
n_m	envelope maximum.
p	attack envelope exponent in GT and REDS atoms.
s	window scale parameter.

Notation	**Description**
u	unit step function.
$\not\varnothing$	generic unbinding.
$\langle \cdot, \cdot \rangle$	inner product.
$\mathbf{max}(\lambda)$	maximum value for range of parameter λ, alternatively μ_λ.

Chapter 1

Introduction

The most common way we think to encode symbols is with *points*. A simple example of this would be a written letter of the alphabet. Representing symbols by points results in an encoding that is *minimally redundant*. However, such encodings are susceptible to noise – a horizontal line accidentally written across a handwritten "l" changes the meaning to "t". Not only are point encodings easily corruptible, but encoding multiple symbols requires multiple points, all separate from each other, resulting in data structures which are unbounded in size. If one wishes to write a long story, they will need a lot of paper.

An alternative way to encode symbols is across a *vector*, where the symbol's meaning is distributed equally along the entire length. Miscellaneous errors at certain coordinates do very little to obscure the encoded symbol, since it is still intact at all the other dimensions of the vector. Encoding a symbol with a vector of *high dimension* is *highly redundant* and difficult to corrupt. An example of such encoding structures are those in the human brain. It is not the case that individual neurons of the brain correspond to specific ideas or

understandings. Rather, it is through the constellation of billions of neurons that the brain takes in the world, and produces some high dimensional representation of it. The distributed nature of our brain's neural representation allows for uninterrupted cognition as cells die and are replaced, ultimately producing a conscious experience robust to certain realities of biology.

Such are the types of observations which inspired hyperdimensional computing (HDC), a brain inspired computing paradigm that encodes symbols across vectors of high dimension, rather than at points [2]. The large amount of redundancy introduced by high dimensional vector encodings allows for multiple symbols to be encoded "on top" of each other. Thus the shape of the encoding does not change as the number of symbols to be encoded grows. One could call this the *high dimensional advantage*. The steep initial investment of high redundancy is quickly repaid in the form of a very powerful, *fixed* dimensional representation which is – ironically – very flexible. Having an encoding which does not change size is valuable for problems that typically have data structures with a tendency to grow too large to compute with. This is a classic problem with traditional algorithms which compute atomic decompositions of audio.

Atomic decomposition of audio aims to represent sound as a linear combination of waveforms, called *atoms*. From a set of atoms which describe a sound, high-quality audio coding, source separation [3], and signal transformation [4] can be achieved. The more the structure of the atom agrees with the sound producing structures in the audio, the fewer atoms are needed to describe the sound [5]. When a decomposition of a signal has only a few atoms, the decomposition is *sparse*. Sparsity is valuable because it allows for more

compressed audio coding, and better control over sound synthesis.

The typical approach to retrieving this small set of atoms involves computing many time-domain correlations between a query signal and a larger set of atoms called a *dictionary*. The most general and well-known algorithm which does this is matching pursuit (MP) [5]. As the query signal grows in length, so too does the length of the atoms in the dictionary. The size of the dictionary grows because more atoms are required to account for the new time. Furthermore, sparsity-promoting atoms generally have many parameters which also increases the size of the dictionary. The amount of memory required to store dictionaries of atoms with more than a few parameters to decompose signals of more than a few seconds can require terabytes of storage, and the number of time correlations to be computed with a matrix of that size is very high. This makes sparse atomic decomposition of audio using dictionaries of meaningful atoms virtually impossible on most personal laptops. These conditions make atomic decomposition a candidate for investigation through the lens of HDC.

In this thesis we show how to systematically encode various time-frequency atoms using a subset of HD computing called vector function architecture (VFA) [6]. Encoding with VFA vectors involves a non-trivial decoding step for which we present a solution based on Newton's method. From these atom encodings, we develop a new method to produce an atomic decomposition which avoids time-domain correlations altogether. In order to decompose a signal with HD encoded atoms, we first need an HD encoding of the signal itself. We choose a deep neural network (DNN) as our HD encoder. The fixed dimension VFA encoding also helps create a DNN model architecture that can generalize to signals of

any length. In the case that the user wishes to compute an atomic decomposition using time-domain correlations, we show that our method can serve as a pre-processing step which pre-selects dictionary atoms for use with MP or its variants such as orthogonal matching pursuit (OMP) [7], ultimately making these algorithms more tractable. To the best of our knowledge, our atomic decomposition system is the first which completely avoids the computation of time-domain correlations.

The contributions of this thesis are:

1. An atomic decompositions which completely avoids time-domain correlations.
2. A method to quickly estimate atom parameter anchor-points, making MP and its derivative algorithms with these atoms more tractable.
3. A method to decode VFA vectors which are the bound product of other vectors and/or in superposition.

1.1 Motivation

The prime motivation of this thesis is to design an atomic decomposition system that is fast. We believe that sparse atomic decomposition is a powerful creative tool, since it allows for flexible control over audio synthesis at the user's discretion. However, the computation difficulties mentioned earlier have almost entirely kept this tool out of the hands of the average creative audio person. Our method is a step in the direction of making an atomic decomposition for all, since it generates decompositions faster than real-time, and can run on a modest personal computer or laptop. Furthermore, our methodology is not specific to

any one atom prototype, but can be applied to any new atom prototypes that arise in the future, or existing atom prototypes which we did not explore in this thesis.

1.2 Overview of Thesis

In Chapter 2 we give an overview of some essential aspects of atomic decomposition of audio, as well as existing solutions to the sparse atomic decomposition problem. In Chapter 3 we start by giving an overview of HD computing, then present our solution to the non-trivial VFA decoding problem, and end the chapter with our reformulation of atomic decomposition. In Chapter 4 we give a review of the areas of deep learning which are relevant to our neural network HD encoder. In Chapter 5 we present the results of our HD atomic decomposition, as well as show that our method can be used in a traditional MP algorithm or its variants. We conclude the thesis in Chapter 6 by sharing our thoughts after having gone through with the project and discuss future directions we see this research going.

Chapter 2

Atomic Decomposition

Atomic decomposition describes a signal as a linear combination of elementary functions (atoms) which belong to a dictionary [8]. For audio, we think of these functions as waveforms. The linear relationship between a signals and a dictionary of atoms means that the atoms must share enough energy with the signal in order to decompose it. We loosely define this as the *synthesis* criteria of atoms. Typically dictionaries are designed by "paving" the time-frequency plane with the atom's time-frequency shape so that sound happening at any time and any frequency has a nearby atom to represent it [8]. Adequate paving can be ensured if the atom has a parameterized *prototype* that controls is time-frequency shape.

There also exist data-driven methods that learn atoms based on a given class of signals, such as speech [9]. These atoms are not parameterized and try to decompose a signal with as few atoms as possible, which is called *sparse* atomic decomposition. The advantage that prototype atoms have over non-parameterized atoms is that their parameters give some

insight into the sound generating structures within the signal. We call this the *analysis* criteria of atoms. The ideal atom has a prototype whose parameters create shapes that match many time-frequency structures in a wide range of audio. Atoms which satisfy both synthesis and analysis criteria allow for flexible and controlled resynthesis of signals, making atomic decomposition a power tool with creative audio applications.

Formally the linear relationship between an audio signal $\mathbf{y} \in \mathbb{R}^N$ and a set of M atoms $\mathcal{D}$ in matrix form $\mathbf{\Phi} \in \mathbb{R}^{N \times M}$ is expressed as

$$\mathbf{y} = \mathbf{\Phi}\mathbf{x} + \epsilon \tag{2.1}$$

$\mathbf{x} \in \mathbb{R}^M$ is a vector of weights describing the contribution of each atom to $\mathbf{y}$ and ϵ is the noise/error term. In this project we focus on the case when $M \gg N$, which means the dictionary is *redundant*. In order to retrieve a sparse $\mathbf{x}$, equation (2.2) is solved.

$$\begin{aligned} &\underset{\mathbf{x}}{\text{minimize}} \ \|\mathbf{x}\|_0 \\ &\text{subject to } \|\mathbf{y} - \mathbf{\Phi}\mathbf{x}\|_2^2 \leq \epsilon \end{aligned} \tag{2.2}$$

where $\|\mathbf{x}\|_0$ is the l_0 "norm" of $\mathbf{x}$, and counts the number of non-zero components. Quotations marks are included around the l_0 "norm" because $\|a\mathbf{x}\|_0 \neq a\|\mathbf{x}\|_0$ for $a \in \mathbb{R}$, and so is not technically a norm. However, the l_0 "norm" directly models the sparsity of $\mathbf{x}$. Minimizing the l_0 "norm" of $\mathbf{x}$ results in a sparse but non-unique solution to equation (2.1).

Time-shift frequency-shift invariant parametric atoms $\phi_m \in \mathcal{D}$ where $1 \leq m \leq M$ have the general form

$$\phi[n] = E[n]e^{i\omega_c n} \tag{2.3}$$

where E is the envelope (windowing) function, $\omega_c = 2\pi f_c$ is the normalized angular frequency of oscillation ($0 \leq f_c \leq \frac{1}{2}$), and n is discrete time. The shape of $E[n]$ can be broadly divided into two categories – *symmetric* and *asymmetric*.

2.1 Symmetric Atoms

Symmetric atoms are characterized by symmetry about a time instant. One of the most common symmetric atoms, the Gabor atom, was introduced in 1946 by Dennis Gabor in [10]. Since no signal can be arbitrarily localized in both time and frequency, Gabor thought the most compact time-frequency structure possible, the Gaussian, should be used as the building block of signals [8]. Gabor decided to modulate infinite duration complex sinusoids with a Gaussian function

$$E[n] = \exp\left(-\pi\left(\frac{n-\tau}{s}\right)^2 \right) \tag{2.4}$$

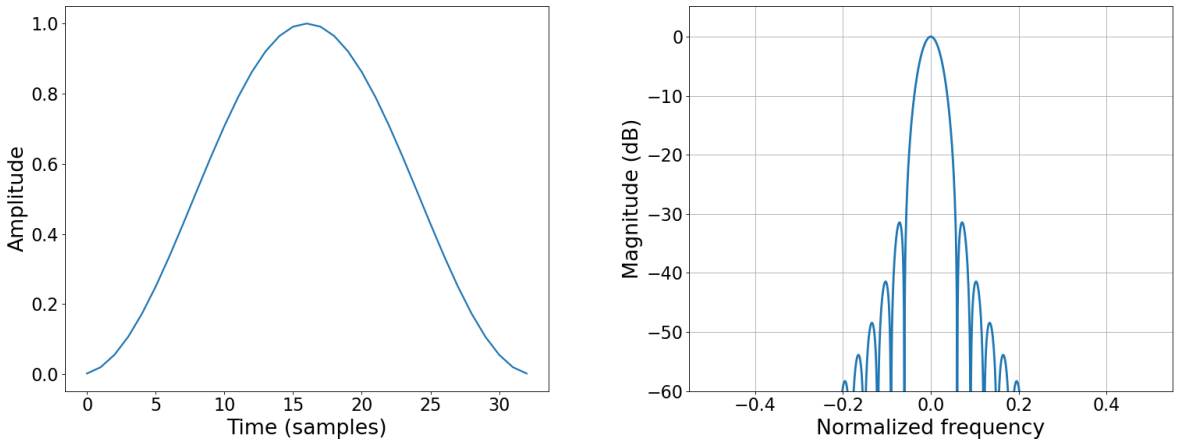
where τ is the time center of the windowing function, and $s \in \mathbb{R}_{\geq 1}$ is the scale parameter controlling the time support and frequency bandwidth of the atom. The small time-frequency footprint of Gabor atoms make them excellent atoms for decompositions which compute correlations via the fast Fourier transform (FFT). Perhaps the most commonplace time-frequency representation, the short-time Fourier transform (STFT), is an atomic decomposition of symmetric atoms [11].

In practice, a pure Gaussian function – which is of infinite duration – is substituted for a discrete time window like the Hanning window function, or Hann window [12]. The N point

Hann window is

$$E_{\text{Hann}}[n] = 0.5 + 0.5\cos\left(\frac{2\pi(n-\tau)}{N-1}\right) \tag{2.5}$$

for $0 \leq n \leq N-1$. The use of a Hann window when computing an STFT is a common choice, and our experiments with Gabor atoms use this window in order to draw a connection between our method and existing popular atomic decompositions with symmetric atoms. The Hann window and its discrete Fourier transform (DFT) is shown in Figure 2.1.



(a) **(b)**

Figure 2.1: Hann window for $\tau = 16$ in **(a)** time and **(b)** frequency.

Gabor atoms are typically short – between 2 and 250 ms – and their small time-frequency footprint means they are good at localizing regions of energy in the time-frequency plane. However, their short duration means that long time-frequency structures, such as partials, must be decomposed with many Gabor atoms, and thus are not sparse. The task of collecting short atoms which together describe a long time-frequency structure is called partial tracking [13]. The reason that long duration Gabor atoms cannot be used to sparsely decompose partials is that the window symmetry is an unrealistic feature of audio. More common in

audio is a window $E[n]$ which is asymmetric.

2.2 Asymmetric Atoms

Asymmetric atoms are characterized by an asymmetric envelope E, where, in general, the portion before the envelope's maximum (the *attack*) is shorter than the portion after the envelope maximum (the *decay*). Asymmetry is an essential characteristic for parametric atoms that sparsely represent a wide range of time-frequency shapes common in audio. The most basic asymmetric envelope parameter is the damping factor.

2.2.1 Damped Sinusoid

The damped sinusoid (DS) is an essential asymmetric atom in atomic modelling of audio given its relationship to a vibrating mode of a resonant structure. Early examples of its use in signal modelling include the well-known Prony's method [14]. An envelope damping parameter is introduced into the atom prototype which allows control over its decay characteristics. The amplitude envelope of a damped sinewave is

$$E_{\mathrm{DS}}[n] = e^{-\alpha n} u[n] \tag{2.6}$$

where $\alpha \in \mathbb{R}_{\geq 0}$ is the damping factor and u is the unit step function. The DS has a well known discrete time Fourier transform

$$\mathcal{F}[\phi_{\mathrm{DS}}](\omega) = \frac{1}{1 - e^{-\alpha + i(\omega - \omega_c)}} \tag{2.7}$$

and a well known N point DFT.

$$\mathcal{F}[\phi_{\mathrm{DS}}](k) = \frac{1 - e^{N(-\alpha + i(\omega_c - 2\pi k/N))}}{1 - e^{-\alpha + i(\omega_c - 2\pi k/N)}} \tag{2.8}$$

From viewing $\mathcal{F}[E_{\mathrm{DS}}](k)$ in Figure 2.2 we see that the spectrum of $E_{\mathrm{DS}}[n]$, and consequently ϕ_{DS}, is unimodal, and thus ensures an optimal shape for parameter estimation in the time-frequency plane. However, the initial discontinuity at the envelope's

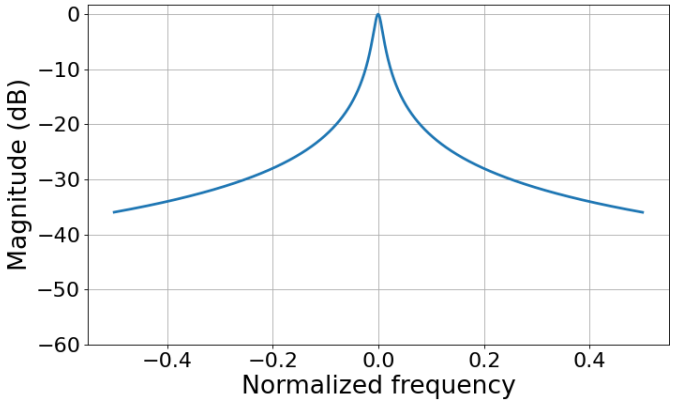


Figure 2.2: DS normalized magnitude spectrum for $\alpha = 0.05$.

attack is an unrealistic feature of real-world audio and requires smoothing in order to better agree with musical sounds. This motivates the investigation into other atom prototypes that introduce an attack parameter in order to provide flexible control over the envelope while maintaining a uni-modal spectrum.

Asymmetric atom prototypes are built from the damped sinusoid window by introducing an attack portion of the envelope denoted $A_{\dagger}[n]$ where $\dagger$ specifies the atom attack shape.

$$E_{\dagger}[n] = A_{\dagger}[n]e^{-\alpha n}u[n] \tag{2.9}$$

It is clear that for damped sinusoids $A_{\mathrm{DS}}[n] = 1$.

Defining the asymmetric atom prototype as a modulation of a damped sinusoid introduces two important quantities: n_m, the *envelope maximum*, and n_I, the *influence time*. In sound synthesis, n_m is typically used as a measure of the attack time [15].

$$n_m = \arg\max_n(E[n]) \tag{2.10}$$

and n_I is

$$n_I = \arg\max_n\Big(e^{-\alpha n}(1 - A[n-1]) > \delta\Big) \tag{2.11}$$

where typically $\delta = 0.001$ i.e. -60 decibels (dBs). Being able to control n_m and n_I through atom parameters are important for designing different sounds from atoms.

2.2.2 Ramped Exponentially Damped Sinusoids

The ramped exponentially damped sinusoid (REDS) is an asymmetric atom designed specifically for sparse atomic decomposition of audio and follows the form of equation (2.9) [1]. It has a compact and unimodal spectrum, and can control the attack and decay characteristics with a minimum number of parameters: 2.

REDS atoms are inspired by two other asymmetric atom prototypes: the gammatone

(GT) [16] and the formant-wave-function (FOF) [17].

$$A_{\mathrm{GT}}[n] = n^p \tag{2.12}$$

$$A_{\mathrm{FOF}}[n] = \begin{cases} \frac{1}{2}(1 - \cos(n\beta)) & \text{for } 0 \leq n \leq \frac{\pi}{\beta} \\ 1 & \text{for } \frac{\pi}{\beta} < n \end{cases} \tag{2.13}$$

The GT and FOF both introduce a smoothing of the DS discontinuity but fall short in terms of desirable audio processing properties in one way or another. On the one hand, the GT provides a smooth attack shape which results in a unimodal spectrum; however, the polynomial attack parameter p does not provide flexible enough control for representation of a wide range of sounds. On the other hand, a FOF introduces a β parameter which allows precise control of the envelope attack; however, the piece-wise construction results in a multi-modal spectrum, making parameter estimation sub-optimal.

The REDS attack envelope uses elements of both a GT and a FOF while avoiding the shortcomings of each atom. $A_{\mathrm{REDS}}[n]$ is a polynomial like a GT and is parameterized by an attack parameter β like a FOF.

$$A_{\mathrm{REDS}}[n] = (1 - e^{-\beta n})^p \tag{2.14}$$

Unlike the piece-wise construction of A_{FOF}, $A_{\mathrm{REDS}}[n]$ is smooth and therefore has a unimodal spectrum. Finally, $p = 1$ reveals the DS-like construction (one minus the DS envelope by letting $\alpha = \beta$). For $p > 1$ the attack onset is smoothed and results in a more narrow bandwidth. REDS envelope maximum is $n_m = \frac{1}{\beta}\log(1 + \frac{p\beta}{\alpha})$ and its influence time is approximately $n_I \approx -\frac{1}{\beta}\log(1 - (1 - \delta)^{1/p})$.

A binomial expansion of $\phi_{\text{REDS}}[n]$ shows its sum of exponentials form

$$\begin{aligned}\phi_{\text{REDS}}[n] &= (1-e^{-\beta n})^p e^{n(-\alpha+i\omega_c)}u[n] \\ &= \sum_{r=0}^{p}(-1)^r\binom{p}{r}e^{n(-\alpha-r\beta+i\omega_c)}u[n]\end{aligned} \tag{2.15}$$

where $\binom{p}{r} = \frac{p!}{(p-r)!r!}$. Since the spectrum of an exponential is well known, the discrete time Fourier transform of ϕ_{REDS} follows easily.

$$\mathcal{F}[\phi_{\text{REDS}}](\omega) = \sum_{r=0}^{p}(-1)^r\binom{p}{r}\frac{1}{1-e^{-\alpha-r\beta+i(\omega-\omega_c)}} \tag{2.16}$$

Similarly its N point DFT is

$$\mathcal{F}[\phi_{\text{REDS}}](k) = \sum_{r=0}^{p}(-1)^r\binom{p}{r}\frac{1-e^{N(-\alpha-r\beta+i(\omega_c-2\pi k/N))}}{1-e^{-\alpha-r\beta+i(\omega_c-2\pi k/N)}} \tag{2.17}$$

REDS concentrated and unimodal spectrum is shown in Figure 2.3.

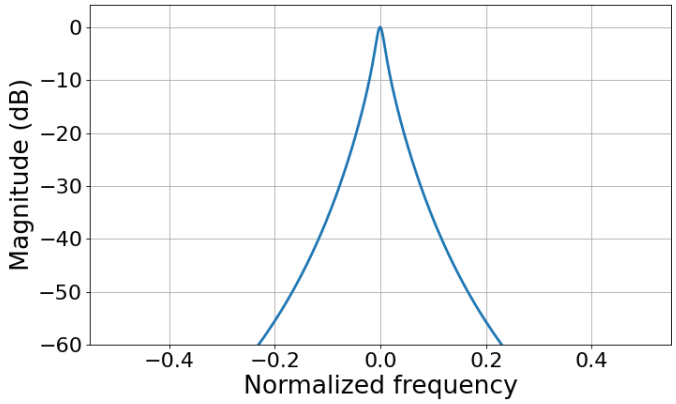
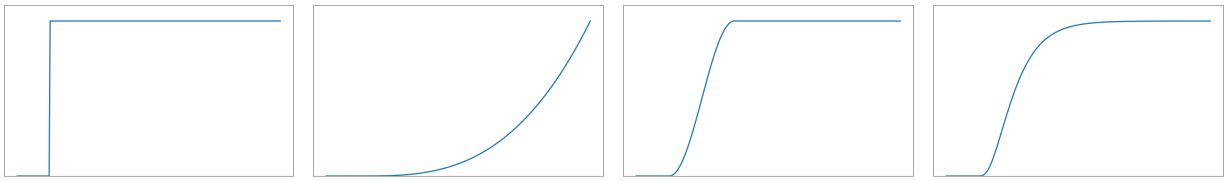


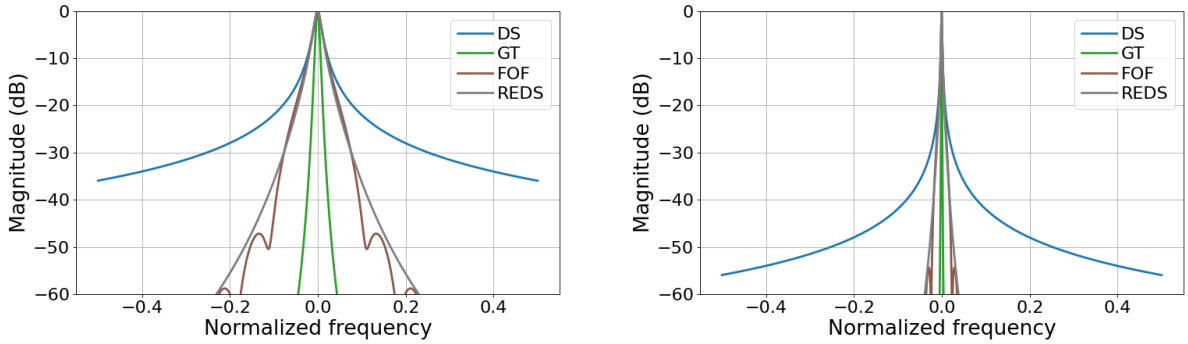
Figure 2.3: REDS normalized magnitude spectrum for $\alpha = 0.05$, $\beta = 0.23$, and $p = 3$.

Figure 2.4 visually compares the asymmetric $A_\dagger[n]$, Figure 2.5 compares the DFT of $A_\dagger[n]$, and Table 2.1 compares each $A_\dagger[n]$ in terms of valuable audio processing qualities. All comparisons point to REDS atoms being a good choice for this project.



(a) DS. **(b)** GT. **(c)** FOF. **(d)** REDS.

Figure 2.4: Comparison of asymmetric $A_\dagger[n]$ for $\beta = 0.05$, and $p = 3$.



(a) $\alpha = 0.05$, $\beta = 0.23$, and $p = 3$. **(b)** $\alpha = 0.005$, $\beta = 0.05$, and $p = 3$.

Figure 2.5: Spectra comparison of asymmetric ϕ for short transient-like atoms (*left*) and long tonal-like atoms (*right*).

When an atom prototype matches a particular sound generating structure, certain parameters of the atom might become analogous to physical properties of the structure itself. For example, consider the asymmetry of sinusoidal components of a plucked guitar string. Decomposing this signal with a dictionary of asymmetric atoms causes α to be seen as a control for the amount of energy reflected at the nut and bridge of the guitar. Tuning

Criteria	DS	GT	FOF	REDS
Concentrated Spectrum	–	✓	✓	✓
Unimodal Spectrum	✓	✓	–	✓
Influence Time Control	–	–	✓	✓
Time-domain Simplicity	✓	✓	–	✓
Causal Filter Simplicity	✓	✓	–	✓
Inner Product Simplicity	✓	–	–	✓

Table 2.1: Asymmetric envelope overview [1].

α has a similar effect as making the nut and bridge at the edge of the string harder or softer, causing the components to ring for longer or shorter.

By using parameters of atoms in a sparse decomposition, new sounds can be generated which are related to, but not an exact reconstruction of the input audio. For example, tuning α so that more energy than is physically possible is reflected along the string in the guitar example. This demonstrates how powerful of a creative tool atomic decomposition can be, where physically unrealistic acoustic sources can be realized by the tuning of a few, simple parameters.

2.3 Models of Audio Synthesis

Asymmetric atoms are typically thought to be used in the *additive* model of audio synthesis. The additive model controls periodic elements in audio, such as partials, by the frequency parameter of the atom. The necessity of post-processing of Gabor atoms with partial tracking in order to reveal these structures shows a natural consequence of decomposition by asymmetric atoms with the additive model: the identification of partials. The partial-identifying nature of asymmetric atoms makes decompositions with them

naturally more meaningful since it reveals structures in the sounds without any further processing.

An alternative model of audio synthesis, the *source-filter* model, controls the periodic elements in audio by the time spacing of atoms, rather than their frequency parameter. This is done by generating audio as the output of a set of filters in parallel, excited by a periodic train of impulses. When the response of the filter is the time waveform of an atom, we can see this model as an atomic decomposition. Here, the frequency parameter controls the resonance of the filter. A sound-producing structure which fits the source-filter model is the human voice, where the excitation of the glottis has an asymmetric shape and is filtered by the vocal tract. Our experiments perform decompositions from both the additive model and the source-filter point of view using the atoms discussed in this chapter.

2.4 Matching Pursuit

In practice there are two major obstacles that must be overcome when solving (2.1): the creation of a set of waveforms on which to decompose, and a method for selecting a sparse subset of these waveforms. The first is accomplished largely through user design, although there exist data-driven methods which aim to learn waveforms for a given set of signals [18]. Selecting a subset of the waveforms is a challenge that yet offers no clear-cut solution, with many approaches focusing on quality [7], others on speed [19]. The most well-known approach to selecting a subset of atoms to decompose a wide-range of signals with is Matching Pursuit (MP) [5].

Matching Pursuit is an iterative, greedy algorithm used to minimize the l_0 "norm" of $\mathbf{x}$ by approximately solving equation (2.2) [5]. At each iteration matching pursuit chooses the atom $\phi_m \in \mathcal{D}$ which best fits a signal $\mathbf{y}$ and computes its contribution x. This operation is expressed as

$$\arg\min_{x,m} \|\mathbf{y} - \phi_m x\|_2^2 \tag{2.18}$$

We can expand $\|\mathbf{y} - \phi_m x\|_2^2$ to see that

$$\begin{aligned} \|\mathbf{y} - \phi_m x\|_2^2 &= (\mathbf{y} - \phi_m x)^\intercal (\mathbf{y} - \phi_m x) \\ &= \mathbf{y}^\intercal \mathbf{y} - \mathbf{y}^\intercal \phi_m x - x\phi_m^\intercal \mathbf{y} + x\phi_m^\intercal \phi_m x \\ &= \|\mathbf{y}\|^2 - 2x\phi_m^\intercal \mathbf{y} + x^2 \|\phi_m\|_2^2 \end{aligned} \tag{2.19}$$

For any m, $\|\mathbf{y} - \phi_m x\|_2^2$ is minimized when its gradient with respect to x is 0, in other words

$$\begin{aligned} 0 = \nabla_x \|\mathbf{y} - \phi_m x\|_2^2 &= \frac{\partial}{\partial x}\left(\|\mathbf{y}\|^2 - 2x\phi_m^\intercal \mathbf{y} + x^2 \|\phi_m\|_2^2 \right) \\ &= -2\phi_m^\intercal \mathbf{y} + 2x\|\phi_m\|_2^2 \\ \rightarrow x &= \frac{\phi_m^\intercal \mathbf{y}}{\|\phi_m\|_2^2} \end{aligned} \tag{2.20}$$

where ϕ_m is often normalized such that $\|\phi_m\|_2^2 = 1$. By substituting x from equation (2.20) in the expansion from equation (2.19), it follows that

$$\begin{aligned} \|\mathbf{y} - \phi_m x\|_2^2 &= \|\mathbf{y}\|^2 - 2\phi_m \mathbf{y}^\intercal \phi_m^\intercal \mathbf{y} + (\phi_m^\intercal \mathbf{y})^2 \|\phi_m\|_2^2 \\ &= \|\mathbf{y}\|^2 - 2(\phi_m^\intercal \mathbf{y})^2 + (\phi_m^\intercal \mathbf{y})^2 \\ &= \|\mathbf{y}\|^2 - (\phi_m^\intercal \mathbf{y})^2 \end{aligned} \tag{2.21}$$

Thus minimizing equation (2.18) reduces to a search over m

$$\arg\min_m \left(\|\mathbf{y}\|^2 - (\phi_m^\intercal \mathbf{y})^2\right) \tag{2.22}$$

which can be equivalently expressed as

$$\arg\max_m |\phi_m^\intercal \mathbf{y}| \tag{2.23}$$

At each iteration k, equation (2.23) is computed on a residual signal $\mathbf{r}^{(k)}$ and the selected atom $\phi_{\tilde{m}}$ where $\tilde{m} = \arg\max_m |\phi_m^\intercal \mathbf{r}^{(k)}|$ is subtracted from $\mathbf{r}^{(k)}$ to get $\mathbf{r}^{(k+1)}$

$$\mathbf{r}^{(k+1)} = \mathbf{r}^{(k)} - x^{(k)}\phi_{\tilde{m}} \tag{2.24}$$

where $x^{(k)} = \phi_{\tilde{m}}^\intercal \mathbf{r}^{(k)}$. When $\phi_m \in \mathbb{R}$, the phase $\angle\phi_m$ must be estimated from a discrete set as for the other parameters. However, when $\phi_m \in \mathbb{C}$ then $x \in \mathbb{C}$ as well and thus contains magnitude and phase information. In this case, the update equation (2.24) becomes

$$\mathbf{r}^{(k+1)} = \mathbf{r}^{(k)} - 2\Re\{x^{(k)}\phi_{\tilde{m}}\} \tag{2.25}$$

This process is outlined in Algorithm 1. Theoretically, given a dictionary that constitutes a frame, MP can decompose any signal such that $r^{(k)} = \mathbf{0}$ for some, possibly infinite, k [5].

At each iteration k, MP selects an atom $\phi_{\tilde{m}}$ which is orthogonal to $r^{(k+1)}$. However, given a redundant dictionary of non-orthogonal atoms, $\phi_{\tilde{m}}$ is not necessarily orthogonal to the residual at later iterations, and consequently can be chosen again. This fact is what allows a dictionary of finite size to need an infinite number of MP iterations to achieve a

null residual as mentioned earlier. A modification to MP, orthogonal matching pursuit (OMP), avoids this by ensuring that the residual at each iteration is orthogonal to *all* atoms selected so far [7].

2.4.1 Orthogonal Matching Pursuit

OMP differs from MP by iteratively building a sub-dictionary of atoms. At each iteration, the most correlated atom from the dictionary is appended to the sub-dictionary, and the correlation with the residual of *all* atoms in the sub-dictionary is computed and used to update the coefficient vector. This results in a residual which is orthogonal to all atoms chosen so far. Because of this, at any iteration k, OMP gives the best k term approximation using the k atoms selected so far – this is not the case with MP. Consequently, for a dictionary of size M, OMP converges to the projection of $\mathbf{y}$ onto $\mathcal{D}$ in no more than M iterations. OMP is outlined in Algorithm 2. In the interest of brevity a least squares (LS) solution is shown for the coefficient update as in [19]. However, the original algorithm is less compact. Interested readers are directed to [7] for details.

Algorithm 1 Matching Pursuit

init: $k = 0,\ \mathbf{x}^{(k)} = \mathbf{0},\ \mathbf{r}^{(k)} = \mathbf{y}$

1: **repeat**
2: $\quad \tilde{m}^{(k)} = \arg\max_m |\phi_m^\intercal \mathbf{r}^{(k)}|$ ▷ best atom selection
3: $\quad x_{\tilde{m}}^{(k)} = \phi_{\tilde{m}}^\intercal \mathbf{r}^{(k)}$ ▷ coefficient computation
4: $\quad x^{(k)} = x^{(k-1)} + x_{\tilde{m}}^{(k)}$ ▷ coefficient update
5: $\quad \mathbf{r}^{(k)} = \mathbf{r}^{(k-1)} - 2\Re\{x^{(k)}\phi_{\tilde{m}}\}$ ▷ residual update
6: $\quad k = k + 1$
7: **until** stopping condition ▷ iteration limit or signal-to-residual ratio (SRR) threshold

In our experiments, we use the implementation proposed in [20] which utilizes efficient inverse Cholesky factorization to vastly reduce the computation complexity. The interested reader is directed to their paper for an in-depth discussion of their method.

2.5 Parameter Refinement

In practice, the parameters of atoms $\phi \in \mathcal{D}$, denoted $\boldsymbol{\lambda}$, are chosen from a discrete set of values. However, the best fitting atom to the signal likely has some set of parameters $\tilde{\boldsymbol{\lambda}}$ in the neighborhood of the initial estimate for $\boldsymbol{\lambda}$. The most general method which retrieves $\tilde{\boldsymbol{\lambda}}$ is Newton's method.

2.5.1 Newton's Method

For a general parameter $\lambda \in \boldsymbol{\lambda}$ we seek a value which minimizes the residual energy function

$$J(\lambda) = \|\mathbf{y} - \phi_\lambda x\|_2^2 \tag{2.26}$$

Algorithm 2 Orthogonal Matching Pursuit

init: $k = 0,\ \mathbf{x}^{(k)} = \mathbf{0},\ \mathbf{r}^{(k)} = \mathbf{y}, \Phi^{(0)} = \emptyset$

1: **repeat**
2: $\tilde{m}^{(k)} = \arg\max_m |\phi_m^\intercal \mathbf{r}^{(k)}|$ ▷ best atom selection
3: $\Phi^{(k)} = [\Phi^{(i-1)}, \phi_{\tilde{m}}]$ ▷ sub-dictionary update
4: $\chi^{(k)} = \left(\Phi^{(k)} * \Phi^{(k)}\right)^{-1} \Phi^{(k)} * r^{(k-1)}$ ▷ coefficient computation via LS solution
5: $x^{(k)} = x^{(k-1)} + \chi^{(k)}$ ▷ coefficient update
6: $\mathbf{r}^{(k)} = \mathbf{r}^{(k-1)} - \Phi^{(k)} \chi^{(k)}$ ▷ residual update
7: $k = k + 1$
8: **until** stopping condition ▷ SRR threshold

Newton's method (NM) searches for the zeros of a function $g(x)$ by iteratively computing

$$x^{(k)} = x^{(k-1)} + \frac{g(x^{(k-1)})}{g'(x^{(k-1)})} \tag{2.27}$$

Equation (2.27) can be modified to find the minimum of a function (where the first derivative equals zero) by letting $g(x) = f'(x)$. This construction finds the minimum of equation (2.26) using the first and second partial derivative of $J(\lambda)$. For a complex-valued atom ϕ_λ the first derivative of $J(\lambda)$ is

$$\frac{\partial}{\partial\lambda}\|\mathbf{y} - \phi_\lambda x\|_2^2 = 2\Re\Big\{\bar{x}\frac{\partial}{\partial\lambda}\phi_\lambda^H\mathbf{y} + |x|^2\frac{\partial\phi_\lambda^H}{\partial\lambda}\phi_\lambda\Big\} \tag{2.28}$$

and the second derivative is

$$\frac{\partial^2}{\partial\lambda^2}\|\mathbf{y} - \phi_\lambda x\|_2^2 = -2\Re\Big\{\bar{x}\frac{\partial^2\phi_\lambda^H}{\partial\lambda^2}\mathbf{y} + |x|^2\Big(\frac{\partial^2\phi_\lambda^H}{\partial\lambda^2}\phi_\lambda + \frac{\partial\phi_\lambda^H}{\partial\lambda}\frac{\partial\phi_\lambda}{\partial\lambda}\Big)\Big\} \tag{2.29}$$

At each iteration k Newton's method updates parameter λ with equation (2.30)

$$\lambda^{(k)} = \lambda^{(k-1)} - \frac{\frac{\partial J}{\partial\lambda}(\lambda^{(k-1)})}{\frac{\partial^2 J}{\partial\lambda^2}(\lambda^{(k-1)})} \tag{2.30}$$

until some maximum iteration is reached or $\mathrm{SRR}^{(k)} \not> \mathrm{SRR}^{(k-1)}$. Furthermore, this algorithm can be generalized to the multi-dimensional case in order to estimate multiple parameters simultaneously. For the parameter vector $\boldsymbol{\lambda} \in \mathbb{R}^Q$ which contains Q parameters of ϕ, we can update their values using a multi-dimensional Newton's step in equation (2.31)

$$\boldsymbol{\lambda}^{(k)} = \boldsymbol{\lambda}^{(k-1)} - \Big(\boldsymbol{H}_{\boldsymbol{\lambda}} J(\boldsymbol{\lambda}^{(k-1)})\Big)^{-1}\nabla_{\boldsymbol{\lambda}} J(\boldsymbol{\lambda}^{(k-1)}) \tag{2.31}$$

Where $\nabla_{\boldsymbol{\lambda}} J(\boldsymbol{\lambda}^{(k-1)})$ is the gradient vector

$$\nabla_{\boldsymbol{\lambda}} J(\boldsymbol{\lambda}^{(k-1)}) = \begin{bmatrix} \frac{\partial J}{\partial \lambda_1}(\lambda_1^{(k-1)}) \\ \vdots \\ \frac{\partial J}{\partial \lambda_Q}(\lambda_Q^{(k-1)}) \end{bmatrix}$$

and $\boldsymbol{H}_{\boldsymbol{\lambda}} J(\boldsymbol{\lambda})$ is the Hessian matrix of $J(\boldsymbol{\lambda})$

$$\Big(\boldsymbol{H}_{\boldsymbol{\lambda}} J(\boldsymbol{\lambda})\Big)_{i,j} = \frac{\partial^2 J}{\partial \lambda_i \partial \lambda_j}(\boldsymbol{\lambda}) = -2\Re\Big\{\bar{x}\frac{\partial^2 \phi_\lambda^H}{\partial \lambda_i \partial \lambda_j}\mathbf{y} + |x|^2\Big(\frac{\partial^2 \phi_\lambda^H}{\partial \lambda_i \partial \lambda_j}\phi_\lambda + \frac{\partial \phi_\lambda^H}{\partial \lambda_i}\frac{\partial \phi_\lambda}{\partial \lambda_j}\Big)\Big\} \tag{2.32}$$

The power of Newton's method lies in its ability to refine any parameter of an atom as long as the atomic expression is twice differentiable with respect to that parameter. For a detailed discussion and exploration of Newton's method for parameter refinement of asymmetric atoms the reader is referred to [1].

2.6 Problem Reformulation

When decomposing a signal, matching pursuit is limited by two factors. First, the reliance on computing many time-domain correlations means that as the signal length N grows, so does the computational cost of computing the decomposition. Second, adding parameters to the atom prototype results in an exponential growth of the dictionary size M. Desiring a sparser decomposition with more meaningful parameters, like those from asymmetric atoms, comes at a much higher computational cost.

Inspired by this observation, we develop an encoding method where atoms are represented by vectors of high but fixed dimension $N_{\mathrm{HD}} \ll N$, regardless of their length in

the time-domain. Furthermore, we utilize the time-frequency shift invariance of the atoms discussed so far, to "reuse" atom encodings throughout the time-frequency plane, resulting in a dictionary of size $M_{\text{HD}} \ll M$. This greatly lessens the trade-off between atom prototype selection and computation time. Given such an encoding method, the focus of atomic decomposition switches from designing a dictionary of waveforms, to generating an encoding of the query signal $\mathbf{y}$ from which the atom parameters can be extracted.

The core of reframing atomic decomposition in this way comes from the field of hyperdimensional computing (HDC), a brain inspired computing paradigm. In Chapter 3 we give an overview of HDC as it pertains to our approach to atomic decomposition.

Chapter 3

Hyperdimensional Computing

Hyperdimensional computing (HDC) is a computing paradigm inspired by models of a biological brain. Like a brain, which has a very high number of neurons ($\approx$ 86 billion for humans), HDC operates in a very high dimensional space of size N_{HD}. Unlike traditional computing methods, where quantities are represented with a single 8-64 bit binary unit, HDC representations encode symbols throughout the entire high dimensional space. This makes it a *distributed* representation. When one or many of the dimensions is corrupted, the encoded symbol remains intact since it is spread equally across all other dimensions. This feature of HDC imbues it with an extreme robustness to noise. The elements of HDC are either random or built from randomness, meaning that the structures encoded in the high dimensional space are not specific to a particular state of the space, but rather to the relationship *between* different states of the space – another observation taken from biological neural modelling [2]. The symbols encoded using HD vectors can, in principal, be anything. In the past, HDC has been used in speech recognition [21], language identification [22], image classification [23], robotic decision making [24], and

encryption [25], to name a few.

In order to motivate the utility of HD computing for our problem, we first start by discussing the traditional form a parameter vector takes. We typically think of the parameter vector for an atom of Q parameters to be a $Q \times 1$ dimensional structure in $\mathbb{R}^{Q\times 1}$ like the parameter vectors discussed in section 2.5.

$$\begin{bmatrix} \lambda_1 \\ ... \\ \lambda_Q \end{bmatrix} \tag{3.1}$$

Equation (3.1) somewhat resembles a key-value pair, where the "key", or parameter position, returns the parameter value. Because the values are stored in a *maximally* sparse, or alternatively *minimally* redundant, manner, they are susceptible to corruption by noise. In addition, atoms with more parameters require more vector positions, and storing multiple parameter sets from M different atoms requires adding another dimension to the structure, making it of size $Q \times M \times 1$

$$\begin{bmatrix} \lambda_{1,1}, ..., \lambda_{M,1} \\ \vdots \quad \vdots \quad \vdots \\ \lambda_{1,Q}, ..., \lambda_{M,Q} \end{bmatrix} \tag{3.2}$$

One of the primary insights from HD computing is that by expanding the "value" dimension from size 1 to a size that is highly redundant, N_{HD}, we can collapse the "key" dimensions (Q and M) entirely. The resulting structure is of fixed dimension N_{HD}, for any

number of encoded parameters and atoms. In this chapter we will show how to systematically build such an encoding, starting with the basic units of HDC.

The basic building blocks of HDC are vectors of high dimension N_{HD} usually between 10^3 and 10^4, sometimes referred to as *hypervectors* [2]. In order to compute with these vectors, certain algebraic operations must be defined which describes how data structures are built and decoded. These operations roughly correspond to multiplication (*binding*) and addition (*superposition*). Together, the set of hypervectors and algebraic operations form a ring-like structure called a Vector Symbolic Architecture.

3.1 Vector Symbolic Architecture

Vector symbolic architecture (VSA) is a term coined by Robert Gayler in [26] where he aimed to solve certain linguistic challenges posed to models of symbolic processing and brain function. In particular, his VSA model, known as the multiply-add-permute (MAP) VSA model, has the useful property of preserving vector dimensionality throughout computation. This differs from other cognitive models such as the tensor product production system (TPPS), whose dimensionality expands throughout subsequent computations [27]. Many properties of a VSA model, such as the dimensionality preserving nature, or lack thereof, are dictated by the specific binding operation chosen. In many ways, binding is the heart and soul of any VSA model.

3.1.1 Binding

The binding operation, generically denoted as $\circ$, is used to "associate" two symbols to each other using their corresponding hypervectors. Binding two non-identity hypervectors to each other generates a new vector which is dissimilar to either vector used in the binding. For example, if we encode symbols $\dot{u}, \dot{v}$ with hypervectors $\mathbf{u}$ and $\mathbf{v}$, and wish to associate them with each other to generate a new symbol $\dot{s}$, the corresponding hypervector $\mathbf{s}$ is built by

$$\mathbf{u} \circ \mathbf{v} = \mathbf{s} \tag{3.3}$$

The similarity between two hypervectors is measured by their inner product

$$\langle \mathbf{u}, \mathbf{v} \rangle = \sum_i u_i v_i \tag{3.4}$$

or more commonly, their cosine similarity

$$\frac{\langle \mathbf{u}, \mathbf{v} \rangle}{\|\mathbf{u}\|_2 \|\mathbf{v}\|_2} = \frac{\sum_i u_i v_i}{\sqrt{\sum_i u_i^2}\sqrt{\sum_i v_i^2}} \tag{3.5}$$

where u_i is the i^{th} coordinate of $\mathbf{u}$. The similarity of vectors used in equation (3.3) can be summarized by

$$\langle \mathbf{u}, \mathbf{s} \rangle \approx 0$$

$$\langle \mathbf{v}, \mathbf{s} \rangle \approx 0$$

In fact, binding a vector with itself results in a vector which is dissimilar to the original vector. In other words

$$\langle \mathbf{u}, \mathbf{u} \circ \mathbf{u} \rangle \approx 0 \tag{3.6}$$

We will see more consequences of this later on in section 3.2. The desired properties of binding are:

1. associative – $(\mathbf{a} \circ \mathbf{b}) \circ \mathbf{c} = \mathbf{a} \circ (\mathbf{b} \circ \mathbf{c}) = (\mathbf{a} \circ \mathbf{c}) \circ \mathbf{b}$
2. distributive over addition – $\sum_i^{D_1} \mathbf{a}_i \circ \sum_{j=1}^{D_2} \mathbf{b}_j = \sum_i^{D_1} \sum_j^{D_2} \mathbf{a}_i \circ \mathbf{b}_j$
3. has an inverse operation to perform unbinding, denoted $\oslash$.

The inverse operation to binding is used to retrieve constituent hypervectors from a bound vector, in other words

$$\mathbf{u} \approx \mathbf{v} \oslash \mathbf{s} = \tilde{\mathbf{u}} \circ \mathbf{s} \tag{3.7}$$

Where $\tilde{\mathbf{u}}$ is the *approximate* inverse of $\mathbf{u}$. $\tilde{\mathbf{u}}$ is an approximation in that it has the same inner product with the other hypervectors as the original vector used in the binding [28], however it can have different amplitude than $\mathbf{u}$. Binding allows encodings to be built which are unique, and not similar to other encodings even if they each have many of the same hypervectors bound inside.

Hadamard Product

The Hadamard binding is perhaps the simplest binding operation we will discuss. Denoted $\odot$, the Hadamard product of two hypervectors is simply the element-wise product of each vector. In other words if $\mathbf{u} \odot \mathbf{v} = \mathbf{s}$, then

$$s_i = u_i \cdot v_i \tag{3.8}$$

This was the binding operation of choice for Gayler in [26]. He also chose his hypervectors to be *bipolar*, where each element is ± 1, chosen randomly. Bipolar hypervectors have the

nice property that each vector unbinds itself, in other words, $\tilde{\mathbf{u}} = \mathbf{u}$. If $\mathbf{u} \odot \mathbf{v} = \mathbf{s}$, then

$$\mathbf{v} = \mathbf{u} \odot \mathbf{s} \quad \text{and } \mathbf{u} \odot \mathbf{u} = \mathbf{1}^{N_{\text{HD}}} \tag{3.9}$$

where $\mathbf{1}^{N_{\text{HD}}}$ is a vector of size N_{HD} where each element is 1. The fact that each hypervector unbinds itself in Gayler's MAP is one of the core ideas behind resonator networks, a highly efficient and accurate algorithm for retrieving bound vectors [29].

Circular Convolution

Another popular binding operations is circular convolution as used by Tony Plate in his holographic reduced representation (HRR) [30]. Denoted as $\circledast$, if the circular convolution of two hypervectors $\mathbf{u} \circledast \mathbf{v} = \mathbf{s}$ then

$$s_k = \sum_{i=1}^{N_{\text{HD}}} u_{(i-k)\%N_{\text{HD}}} v_i \tag{3.10}$$

where % is the standard modulo operator. The relationship between convolution and multiplication via the Fourier transform reveals the simpler form of circular convolution. $\mathbf{s} = \mathbf{u} \circledast \mathbf{v}$ can alternatively computed as

$$\mathbf{s} = \mathcal{F}^{-1}(\mathcal{F}(\mathbf{u}) \odot \mathcal{F}(\mathbf{v})) \tag{3.11}$$

As can be seen from equation (3.11), unbinding circular convolution is done by conjugation of the Fourier transform, in other words

$$\mathbf{u} \oslash \mathbf{v} = \mathcal{F}^{-1}(\overline{\mathcal{F}(\mathbf{u})} \odot \mathcal{F}(\mathbf{v})) \tag{3.12}$$

The ability to perform $\circledast$ via the FFT greatly increases its scalability for an otherwise expensive computation.

3.1.2 Superposition

The other primary operation in a VSA is superposition. Superposition, denoted $+$, collects hypervectors into a new vector. Collecting $\mathbf{u}$ and $\mathbf{v}$ to generate $\mathbf{s}$ via superposition is written

$$\mathbf{u} + \mathbf{v} = \mathbf{s} \tag{3.13}$$

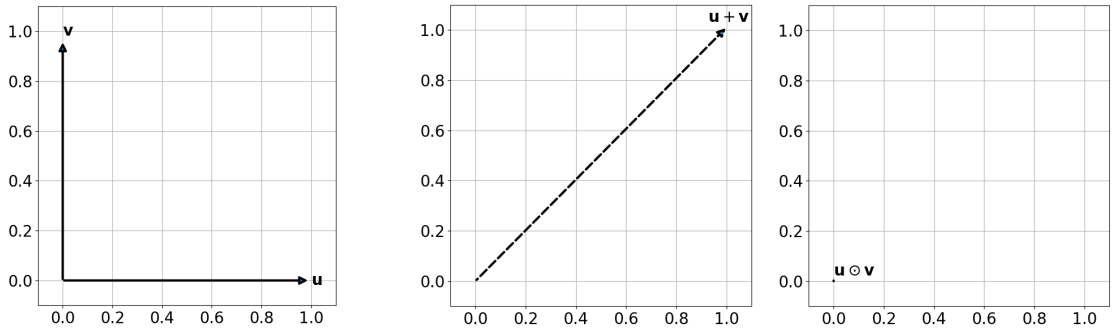
where $s_i = u_i + v_i$. Superposition differs from binding in that it preserves the similarity of the input vectors to the output vector, in other words

$$\begin{aligned} \langle \mathbf{u}, \mathbf{s} \rangle &\gg 0 \\ \langle \mathbf{v}, \mathbf{s} \rangle &\gg 0 \end{aligned} \tag{3.14}$$

In practice, binding is generally used to associate symbols belonging to a single entity, such as the color and texture of an object, where it is the *combination* of symbols which prevails, not any one symbol. Alternatively, superposition is used to collect the encodings of several entities into a single hypervector, where each entity prevails as equally as the others.

In order to illustrate the similarity preserving or destroying nature of superposition and binding, we show their effects on vectors in $\mathbb{R}^2$ in Figure 3.1.

In many early VSA works, the problems being solved originated in linguistics [28]. Cognitive behavior was observed through language, and then attempted to be reproduced



(a) $\mathbf{u} \in \mathbb{R}^2$, $\mathbf{v} \in \mathbb{R}^2$ **(b)** $\langle \mathbf{u}, \mathbf{u}+\mathbf{v} \rangle \gg 0$, $\langle \mathbf{v}, \mathbf{u}+\mathbf{v} \rangle \gg 0$ **(c)** $\langle \mathbf{u}, \mathbf{u} \odot \mathbf{v} \rangle = 0$, $\langle \mathbf{v}, \mathbf{u} \odot \mathbf{v} \rangle = 0$

Figure 3.1: Simple $N_{\text{HD}} = 2$ example to show the similarity consequences of superposition and binding.

in symbolic models of reasoning. As such, many of the original symbols which were being encoded using VSA were words. Since individual words by and large have no obvious numerical relationship between each other, their corresponding VSA hypervectors were generated randomly – either random binary [31], random bi-polar [26], or random Gaussian [28] – and consequently are quasi-orthogonal to each other [29]. However, for encoding symbols which have a numerical relationship to each other, such as size or distance, it is desirable to have this relationship persist in the encoding space.

Much of the initial experimentation in this thesis was done purely with the VSA technologies mentioned so far. However, we found that encoding symbols which have numerical relationships between each other in a way which does not preserve this relationship to be sub-optimal. We were encouraged by experimentation which preserves this relationship, however existing methods of preserving numerical quantites are restricted to one encoding symbol [32]. For the time-frequency atoms we discussed in chapter 2 there

are either $2, 3$, or 4 numerical symbols (parameters).

In [6], an HDC framework is presented which collects previous VSA explorations and formalizes an architecture that generalizes symbol encoding from a discrete set to a continuous range of real numbers for any number of parameters. Additional capabilities such as computing with functions are addressed as well, all in a mathematically rigorous manner. This new framework gave our project the tools it needed to truly build the atomic decomposition system we envisioned.

3.2 Vector Function Architecture

Vector Function Architecture (VFA) is an extension of Vector Symbolic Architecture which permits the encoding of a continuous range of values, as well as the manipulation of functions such as function convolution and function shifting, all in the high dimensional space. VFA collects contributions from previous VSA projects and unifies them in a well-defined mathematical framework. The curious reader is directed to [6] for a complete description of Vector Function Architecture, while here we will give an overview of certain elements of VFA which will be used in our approach to atomic decomposition.

In general, the coordinates of VFA hypervectors are complex numbers. The expression for the inner product between encodings is rewritten to the general vector form

$$\langle \mathbf{u}, \mathbf{v} \rangle = \mathbf{u}^{\intercal} \overline{\mathbf{v}} \tag{3.15}$$

With these complex-valued high dimensional vectors, real numbers can be encoded through

fractional power encoding (FPE) by using the binding operation. FPE starts with self-binding, which describes binding a vector with itself k times

$$\mathbf{u}(k) = (\mathbf{u})^{(\circ k)} = \mathbf{u} \underbrace{\circ ... \circ}_{k-1 \text{ times}} \mathbf{u} \tag{3.16}$$

as a way to encode integers. This is then generalized to real numbers r

$$f_{\text{FPE}} : r \in \mathbb{R} \rightarrow \mathbf{u}(r) = (\mathbf{u})^{(\circ r)} \in \mathbb{C} \tag{3.17}$$

For Hadamard binding, self-binding corresponds to

$$\mathbf{u}(k) = (\mathbf{u})^{(\odot k)} = \mathbf{u} \underbrace{\odot ... \odot}_{k-1 \text{ times}} \mathbf{u} \tag{3.18}$$

which results in the exponentiation of each coordinate of $\mathbf{u}$ by r, or

$$\mathbf{u}(r)_n = (u_n)^r \tag{3.19}$$

For circular convolution, self-binding corresponds to

$$\mathbf{u}(k) = (\mathbf{u})^{(\circledast k)} = \mathbf{u} \underbrace{\circledast ... \circledast}_{k-1 \text{ times}} \mathbf{u} = \mathcal{F}^{-1}\Big(\mathcal{F}(\mathbf{u}) \underbrace{\odot ... \odot}_{k-1 \text{ times}} \mathcal{F}(\mathbf{u})\Big) \tag{3.20}$$

which results in the exponentiation of each element of the Fourier transform of $\mathbf{u}$ by r

$$\mathbf{u}(r)_n = \mathcal{F}^{-1}(\mathcal{F}(\mathbf{u})^r)_n \tag{3.21}$$

The particular FPE encoding of a continuous value r depends on the base vector $\mathbf{u}$. A

notable subset of hypervectors with complex coordinates is the set of unitary vectors

$$A_\circ = \{\mathbf{u} : \|\mathbf{u} \circ \mathbf{v}\|_2 = \|\mathbf{v}\|_2 \ \forall \mathbf{u}\} \tag{3.22}$$

These vectors are special because binding two unitary vectors generates another unitary vector. For Hadamard product binding, vectors with entries on the unit circle are an obvious set of unitary vectors

$$A_\odot = \{\mathbf{u} : u_n = e^{i\theta_n} | \forall \theta_n \in \mathbb{R}\} \tag{3.23}$$

For circular convolution, unitary vectors correspond to the inverse Fourier transform of points on the unit circle

$$A_\circledast = \{\mathbf{u} : \mathcal{F}(\mathbf{u})_n = e^{i\theta_n} | \forall \theta_n \in \mathbb{R}\} \tag{3.24}$$

Because of this, there is the nice consequence of being able to generate *real* unitary vectors using circular convolution. It is well known that the inverse Fourier transform of a complex vector with Hermitian symmetry is real. Consequently, taking the inverse Fourier transform of a vector of points on the unit circle which is Hermitian symmetric returns a real unitary vector under circular convolution.

For either binding operation, a vector of points on the unit circle must be generated. In general we denote the function

$$f_{A_\circ} : \boldsymbol{\theta} \in \mathbb{C} \to \mathbf{u} \in A_\circ \tag{3.25}$$

which takes a vector of angles and maps them to a corresponding unitary vector for the given binding operation $\circ$.

The property that similarity between symbols is preserved in the encoding space make the Hadamard product and circular convolution a locality preserving encoding (LPE). In fact, if the base vector $\mathbf{u}$ is sampled from certain distributions, there exists a similarity kernel K which precisely defines the similarity between the encodings for two real numbers r_1 and r_2. Encodings that have a similarity kernel are known as a kernel LPE (KLPE). An LPE of the form in equation (3.17) is a KLPE if for sufficiently large N_{HD} the inner product of encoding points defines a translation-invariant similarity kernel

$$\mathbf{u}(r_1)^\intercal \overline{\mathbf{u}(r_2)} \to K(r_1 - r_2) \tag{3.26}$$

Furthermore, this similarity kernel is compatible with a VSA binding operation if encoding $r_1 + r_2$ corresponds to binding the encoding for each value, in other words

$$\mathbf{u}(r_1 + r_2) = \mathbf{u}(r_1) \circ \mathbf{u}(r_2) \tag{3.27}$$

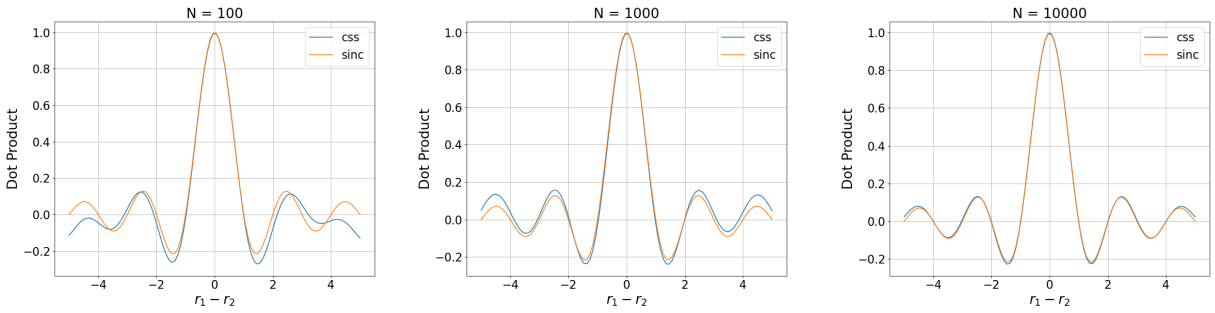
For unitary vectors whose $\boldsymbol{\theta}$ is sampled from a uniform distribution, the resulting similarity kernel is the sinc function.

$$\text{sinc}(r) = \frac{\sin(\pi r)}{\pi r} \tag{3.28}$$

The similarity between two encoding vectors approaches the analytic similarity kernel as N_{HD} increases, as shown in figure 3.2.

This similarity behavior can be generalized to binding $Q > 1$ vectors.

$$(\mathbf{u}_1(r_1) \circ ... \circ \mathbf{u}_Q(r_Q))^\intercal \overline{(\mathbf{u}_1(s_1) \circ ... \circ \mathbf{u}_Q(s_Q)} = K(r_1 - s_1) \times ... \times K(r_Q - s_Q) \tag{3.29}$$



(a) $N_{\mathrm{HD}} = 100$ **(b)** $N_{\mathrm{HD}} = 1000$ **(c)** $N_{\mathrm{HD}} = 10,000$

Figure 3.2: VFA Sinc Similarity Kernel for uniform sampling. $Q = 1$.

This is shown in Figure 3.3 for $Q = 2$.

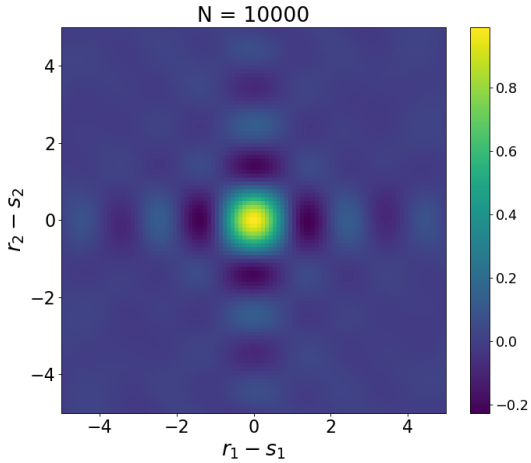


Figure 3.3: Similarity Kernel for uniform sampling when $N_{\mathrm{HD}} = 10,000$ and $Q = 2$.

Looking at the similarity kernel displays how encoding real numbers is a natural extension of encoding integers since the zero crossing remain at integer values. When thinking of encoding integers i as self binding a base vector with itself i times as written in (3.16), the true similarity-destroying nature of binding is on full display: even when two HD vectors are identical, binding them together creates a hypervector almost completely orthogonal to the

original vector. This is the same behavior we pointed out in equation (3.6) but now we see it displayed as a sinc function and generalized by the VFA framework.

3.3 VFA Atomic Encoding

Our approach to atomic decomposition is based on encoding atoms in HD space using the VFA framework laid out in the previous section. For an atom $\phi_{\boldsymbol{\lambda}}$ of Q parameters, we first create a base vector $\mathbf{u}_{\lambda_q}$ for each $\lambda_q \in \boldsymbol{\lambda}$. These base vectors are stored for the decoding process which we detail in the next section. Then we encode each particular $\dot{\lambda}_q$ using its corresponding base vector, $\mathbf{u}_{\lambda_q}$ via the FPE described in equation (3.17). Finally, the HD vector which encodes $\phi_{\boldsymbol{\lambda}}$ is built by binding together all $\mathbf{u}_{\lambda_q}(\dot{\lambda}_q)$ using $\circ$:

$$\mathbf{s}_{\circ}(\dot{\boldsymbol{\lambda}}) = \mathbf{u}_{\lambda_1}(\dot{\lambda}_1) \circ \mathbf{u}_{\lambda_2}(\dot{\lambda}_2) \circ ... \circ \mathbf{u}_{\lambda_Q}(\dot{\lambda}_Q) \tag{3.30}$$

We choose unitary vectors for our base vectors $\mathbf{u}_{\lambda_q}$ because of their success in previous HDC works as well as their nice mathematical properties. For each parameter λ_q we generate a Hermitian symmetric phasor $\boldsymbol{\theta}_q$ and store the corresponding unitary vector

$$\mathbf{u}_{\lambda_q} = f_{A_{\circ}}(\boldsymbol{\theta}_q) \tag{3.31}$$

In order to scale the atoms by their complex gain coefficient x, we bind the atom encoding $\mathbf{s}(\dot{\boldsymbol{\lambda}})$ with a vector encoding of x, $\vec{\mathbf{x}}_{\text{HD}}$. For Hadamard binding, $\vec{\mathbf{x}}_{\text{HD}}$ is simply

$$(\vec{\mathbf{x}}_{\text{HD}})_n = x \tag{3.32}$$

for $1 \leq n \leq N_{\text{HD}}$. However, because circular convolution operates using the Fourier transform of vectors, the coefficient encoding vector for $\circledast$ is

$$\mathcal{F}(\vec{\mathbf{x}}_{\text{HD}})_n = x \tag{3.33}$$

for $1 \leq n \leq \frac{N_{\text{HD}}}{2}+1$, and then conjugate mirrored for the following $\frac{N_{\text{HD}}}{2}-1$ vector coordinates. We define the function

$$\gamma_\circ(x) : x \in \mathbb{C} \mapsto \vec{\mathbf{x}}_{\text{HD}} \tag{3.34}$$

which maps an atom's coefficient x to the corresponding HD vector $\vec{\mathbf{x}}_{\text{HD}}$ depending on the binding operation $\circ$.

Equations (3.30) - (3.34) are combined to encode a time-signal which is the linear combinations of time-frequency atoms

$$\mathbf{y} = \sum_k x_k \phi_{\boldsymbol{\lambda}_k}$$

by the HD vector

$$\mathbf{z} = \sum_k \gamma_\circ(x_k) \circ \mathbf{s}_\circ(\boldsymbol{\lambda}_k) \tag{3.35}$$

for a binding operation $\circ$. For this project we choose circular convolution as the binding operation ($\circ = \circledast$) because of the ability to generate real-valued vectors, as discussed in section 3.2. Because circular convolution binding (and Hadamard binding) produce an output vector the same size as its inputs, which is of fixed dimension N_{HD}, the size of encoded atoms does not change with its length in the time-domain, N. The burden of long duration signals is of no concern given this encoding method.

Because binding destroys the similarity between the input and output vectors, retrieving $\boldsymbol{\lambda}_k$ is not trivial for any VSA model, and is further complicated by the continuous nature of VFA encodings. In the next section we show that extending Newton's method as described in section 2.5.1 to unitary HD vectors works quite well for retrieving $\boldsymbol{\lambda}_k$.

3.4 Decoding VFA Vectors

Once bound, retrieving the vectors which went into the binding is difficult because the similarity of the vectors to the bound product is destroyed. When the set of symbols being encoded is discrete, retrieving the individual vectors used in the binding is a hard combinatorial search problem. Recently, efficient algorithms for retrieving the input vectors have emerged which only search a fraction of the total search space [29]. However, for continuous encodings, there exists no finite set of vectors which we can search through in a brute-force manner.

In [6] an optimization approach was presented to decode VFA vectors which uses the structure of unitary vectors to approximate an encoded real number. First, an initial estimate is obtained through a matrix dot product of discrete points in the encoding space, referred to as anchor-points. Then gradient descent refines the course estimate to better approximate the true value. Here, we present an alternative method by extending Newton's method, as described in section 2.5.1 for time-domain signals, to the HD space. We call this "HD Newton's method" in order to distinguish it from the time-domain version already discussed, as well as draw comparisons between the two. For simplicity we

detail HD Newton's method for $\circ = \odot$, but note that the since $\odot$ and $\circledast$ are related by the Fourier transform, the same steps we show for $\odot$ apply to the Fourier transform of vectors bound with $\circledast$.

For unitary VFA base vectors, HD Newton's method aims to find the minimum of

$$J_{\text{HD}}(\boldsymbol{\lambda}) = \|\mathbf{z} - \gamma_{\odot}(x) \odot \mathbf{s}_{\odot}(\boldsymbol{\lambda})\|_2^2 \tag{3.36}$$

The structure of unitary vectors (exponentials) give easy access to their first and second partial derivative which allows for a seamless extension of equation (2.31). The first partial derivative of HD encodings for atoms in the form of equation (3.30) is

$$\frac{\partial \mathbf{s}}{\partial \boldsymbol{\theta}_j} = i\dot{\lambda}_j \cdot e^{i\dot{\lambda}_1 \boldsymbol{\theta}_1} \cdot ... \cdot e^{i\dot{\lambda}_Q \boldsymbol{\theta}_Q} \tag{3.37}$$

and the second partial derivative is

$$\frac{\partial^2 \mathbf{s}}{\partial \boldsymbol{\theta}_j \partial \boldsymbol{\theta}_k} = -1 \cdot \dot{\lambda}_j \dot{\lambda}_k \cdot e^{i\dot{\lambda}_1 \boldsymbol{\theta}_1} \cdot ... \cdot e^{i\dot{\lambda}_Q \boldsymbol{\theta}_Q} \tag{3.38}$$

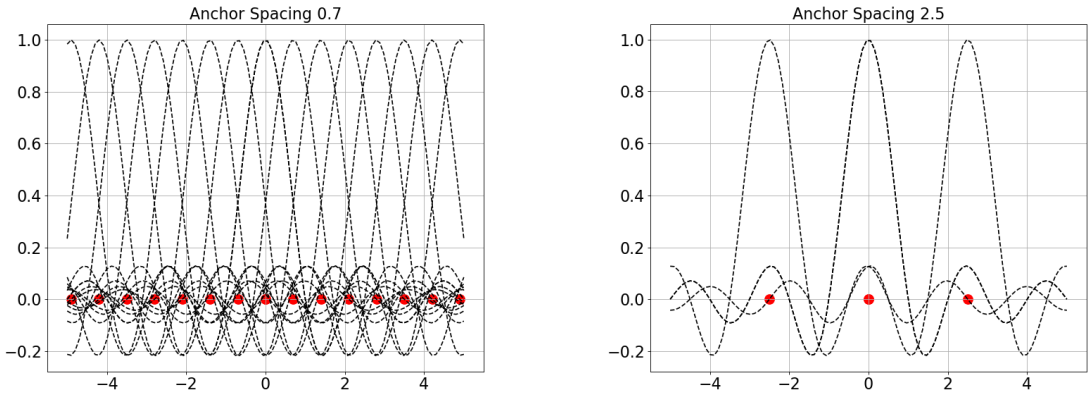
for $j, k \in [1, ..., Q]$. Once again, multiple atomic parameters can be refined simultaneously, except this time it is being done in the HD space. At each iteration k, we update the estimation of $\boldsymbol{\lambda}$

$$\boldsymbol{\lambda}^{(k)} = \boldsymbol{\lambda}^{(k-1)} - \left(\boldsymbol{H}_{\boldsymbol{\lambda}} J_{\text{HD}}(\boldsymbol{\lambda}^{(k-1)})\right)^{-1} \nabla_{\boldsymbol{\lambda}} J_{\text{HD}}(\boldsymbol{\lambda}^{(k-1)}) \tag{3.39}$$

The initial estimate $\boldsymbol{\lambda}^{(0)}$ is obtained from a matrix dot product of HD vectors.

$$\boldsymbol{\lambda}^{(0)} = \text{argmax}_m |\boldsymbol{\Phi}_{\text{HD}}^{\intercal} \bar{\mathbf{z}}| \tag{3.40}$$

The anchor-point matrix can be seen to be analogous to the matrix dictionary from Chapter 2, and so we call it the HD matrix dictionary, denoted $\mathbf{\Phi}_{\text{HD}}$. The anchor spacing, which is used to generate an initial estimate $\boldsymbol{\lambda}^{(0)}$, can be visualized by placing the VFA similarity kernel along the encoding range at each anchor-point. In Figure 3.4 we see the anchor-points for spacing of 0.7 and 2.5. Having anchor-points further apart from each other leaves large regions of the encoding range with no nearby anchor, therefore making those values difficult to decode. For example, as we see in Figure 3.4b, an anchor spacing of 2.5 would make an encoded value of $\lambda = 1$ non-decodable because it does not have a nearby anchor which can be used to generate a good initial estimate $\lambda^{(0)}$. Anchor-points should be spaced often enough so that every point in the entire encoding range has a nearby anchor from which its value can be decoded from.



(a) **(b)**

Figure 3.4: Similarity (dotted line) between encoded values in the range $[-4, 4]$ and a discrete set of anchor-points (red dots).

To test the ability of Newton's method to retrieve $\boldsymbol{\lambda}$ given $\boldsymbol{\lambda}^{(0)}$ we encode random values

and run a grid search through various N_{HD}, Q, and anchor spacing. C is the number of vectors in superposition, which we choose to be 1. Figure 3.5 shows box plots comparing the anchor spacing with the cosine similarity between the refined estimate vector and the ground truth vector for $N_{\text{HD}} = 500$ and $Q = 1, 2, 3, 4$.

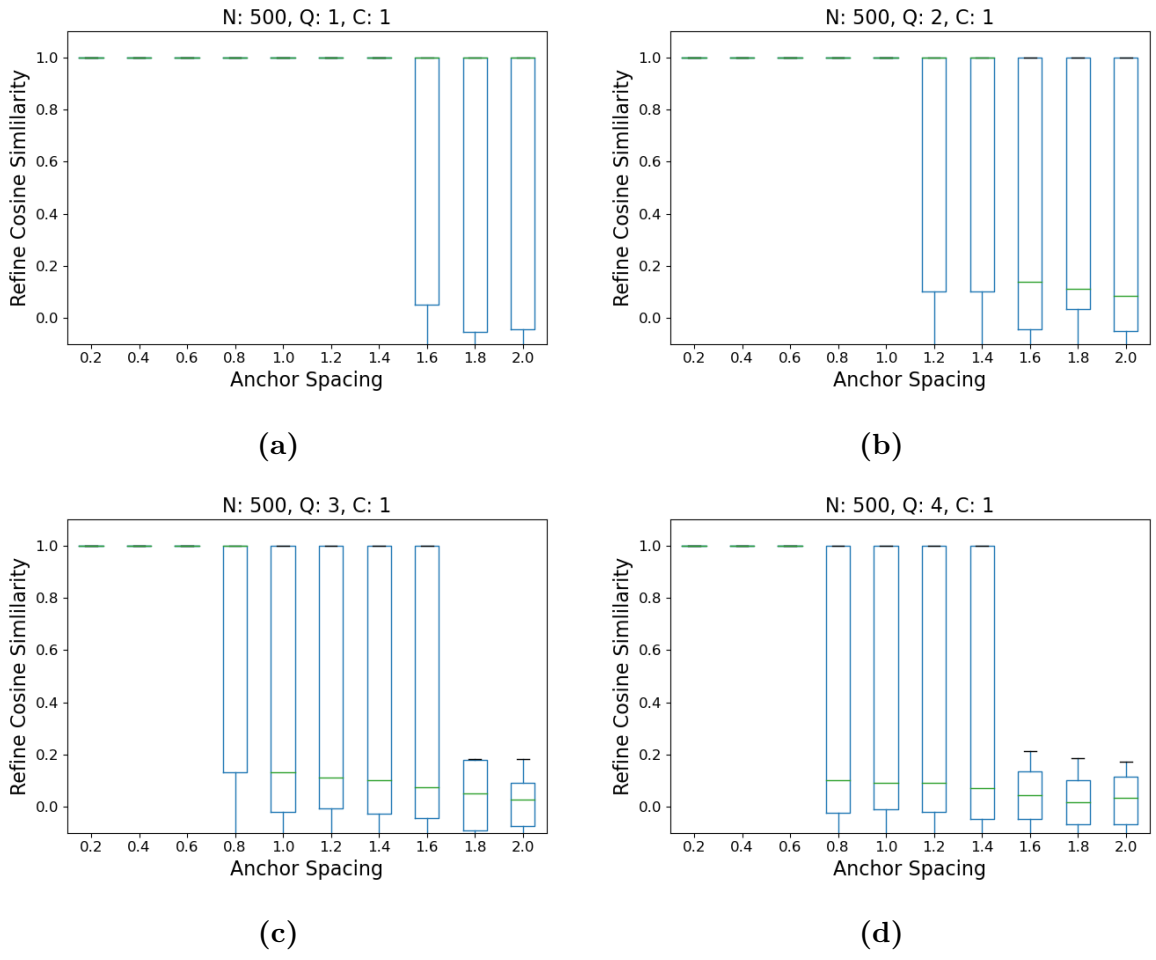


Figure 3.5: Evaluation of Newton's method for VFA vectors for different Q.

We see three cases arise. First, when the anchor spacing is tight HD Newton's method always provides a very accurate parameter estimate. This is shown by a box plot which is

condensed into a line at cosine similarity = 1. Second, less dense anchor spacing results in a box plot whose 25^{th}-percentile is down near 0 but its 75^{th}-percentile is near 1, with the median being at either extreme. Third, when the anchor spacing is very sparse the box plot range is concentrated near 0, meaning that HD Newton's method cannot retrieve an accurate parameter estimates. In general, as Q gets bigger the anchors need to be closer to each other in order to retrieve an accurate estimate of $\boldsymbol{\lambda}$. This is likely due to the nature of multi-dimensional kernels, as shown in equation (3.29), where the peaks around anchor-points get sharper and more selective due to multiplying together many numbers which are less than 1.

From the HD perspective, noisy approximations of $\mathbf{s}_{\odot}(\boldsymbol{\lambda})$ should not be very detrimental to retrieving the true parameters due to HD vector's natural robustness to noise and the similarity preserving nature of superposition. A noisy approximation of $\mathbf{s}_{\odot}(\boldsymbol{\lambda})$, $\hat{\mathbf{s}}_{\odot}(\boldsymbol{\lambda})$ can be seen as $\mathbf{s}_{\odot}(\boldsymbol{\lambda}) + \epsilon$ where ϵ is some amount of noise. As we saw in Figure 3.1, the similarity between $\mathbf{s}_{\odot}(\boldsymbol{\lambda})$ and $\hat{\mathbf{s}}_{\odot}(\boldsymbol{\lambda})$ still has a relatively high cosine similarity. Thus the initial estimate $\boldsymbol{\lambda}^{(0)}$ shouldn't be that inaccurate and Newton's method should be able to correctly denoise $\hat{\mathbf{s}}_{\odot}(\boldsymbol{\lambda})$ to retrieve $\mathbf{s}_{\odot}(\boldsymbol{\lambda})$, and consequently $\boldsymbol{\lambda}$. We test Newton's method ability to retrieve $\mathbf{s}_{\odot}(\boldsymbol{\lambda})$ from $\hat{\mathbf{s}}_{\odot}(\boldsymbol{\lambda})$ by adding Gaussian white noise to the vector $\mathbf{s}_{\odot}(\boldsymbol{\lambda})$ for various signal-to-noise ratio (SNR) levels as shown in Figure 3.6 for $N_{\text{HD}} = 500$ and $Q = 2, 3, 4$.

We see that Newton's method performs very well at retrieving good parameter estimates as long as the noise isn't more powerful than the signal. HD Newton's method also works to decode the superposition of multiple superposed VFA vectors, $C > 1$. When

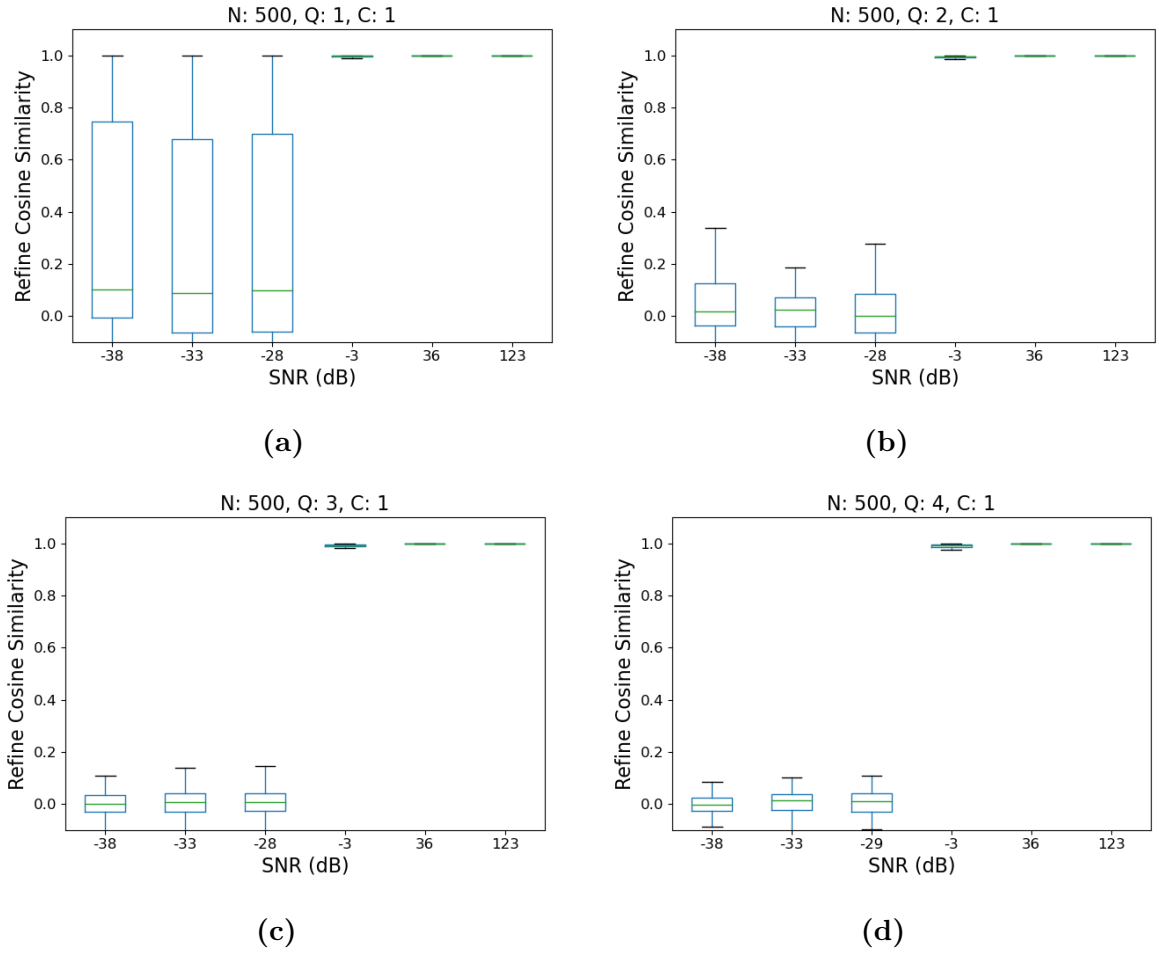


Figure 3.6: HD Newton's method performs well as long as the noise is not more powerful than the signal.

decoding the superposition of multiple HD vectors, HD Newton's Method refines one encoding at a time, and subtracts it from the remainder until the squared norm of the remainder no longer decreases. In practice we observe that HD Newton's method performs worse as Q and C increase. This can be accounted for by increasing N_{HD}, which allows for the encoded symbols to be distributed across more dimensions, making them more robust to different scenarios.

Finally, the coefficient x is estimated in the same way as in signal processing: a dot product.

$$x = \langle \mathbf{s}_{\odot}(\boldsymbol{\lambda}), \mathbf{z} \rangle = \mathbf{s}_{\odot}(\boldsymbol{\lambda})^{\intercal} \overline{\mathbf{z}} \tag{3.41}$$

We summarize the steps to decoding VFA vectors in algorithm 3.

Algorithm 3 Decoding bound VFA hypervectors in superposition

given: $\boldsymbol{\Phi}_{\mathrm{HD}}, \mathbf{z}$ **init:** $k = 0,\ \boldsymbol{\Pi}^{(k)} = \emptyset,\ \mathbf{X}^{(k)} = \emptyset,\ \mathbf{r}_{\mathrm{HD}}{}^{(k)} = \mathbf{z},\ x^{(k)} = 1$

repeat
  $\boldsymbol{\lambda}^{(k)} = \mathrm{argmax}_m |\Phi_{\mathrm{HD}}^{\intercal} \bar{\mathbf{r}}_{\mathrm{HD}}^{(k)}|$ ▷ best atom selection
  init: $j = 0,\ \boldsymbol{\lambda}^{(j)} = \boldsymbol{\lambda}^{(k)}$ ▷ Newton's method
  repeat
    $\boldsymbol{\lambda}^{(j+1)} = \boldsymbol{\lambda}^{(j)} - \left(\boldsymbol{H}_{\boldsymbol{\lambda}} J_{\mathrm{HD}}(\boldsymbol{\lambda}^{(j)}) \right)^{-1} \nabla_{\boldsymbol{\lambda}} J_{\mathrm{HD}}(\boldsymbol{\lambda}^{(j)})$
    $x^{(j+1)} = \mathbf{s}_{\odot}(\boldsymbol{\lambda}^{(j+1)})^{\intercal} \bar{\mathbf{r}}_{\mathrm{HD}}^{(k)}$ ▷ update coefficient approximation
    $j = j + 1$
  until stopping condition ▷ iteration limit
  $\boldsymbol{\Pi}^{(k+1)} = [\boldsymbol{\Pi}^{(k)}, \boldsymbol{\lambda}^{(j)}]$ ▷ store refined parameters
  $\mathbf{X}^{(k+1)} = [\mathbf{X}^{(k)}, x^{(j)}]$ ▷ store coefficient
  $\mathbf{r}_{\mathrm{HD}}^{(k+1)} = \mathbf{r}_{\mathrm{HD}}^{(k)} - \gamma_{\odot}(x^{(j)}) \odot \mathbf{s}_{\odot}(\boldsymbol{\lambda}^{(j)})$ ▷ residual update
  $k = k + 1$
until stopping condition ▷ square norm of HD residual increases

3.5 Hyperdimensional Atomic Decomposition

We call our atomic decomposition system which uses the atom encodings from section 3.3 and HD Newton's method decoding from section 3.4, Hyperdimensional Atomic Decomposition (HD-AD). To the best of our knowledge, ours is the first system which computes atomic decompositions of audio and avoids time-domain correlations altogether. Having said that, a number of similarities exist between a traditional matching pursuit

atomic decomposition and our reformulated HD approach. We outline these in Table 3.1.

Step	Time-domain MP	HD Atomic Decomposition
initialization	$k = 0,\ \mathbf{x}^{(k)} = \mathbf{0},\ \mathbf{r}^{(k)} = \mathbf{y}$	$k = 0,\ \mathbf{x}^{(k)} = \mathbf{0},\ \mathbf{r}_{\text{HD}}^{(k)} = \mathbf{z}$
best atom selection	$\tilde{m}^{(k)} = \arg\max_m \lvert\mathbf{\Phi}^{\intercal}\overline{\mathbf{r}}^{(k)}\rvert$	$\tilde{m}^{(k)} = \arg\max_m \lvert\mathbf{\Phi}_{\text{HD}}^{\intercal}\overline{\mathbf{r}}_{\text{HD}}^{(k)}\rvert$
parameter - refinement	to minimize: $J(\boldsymbol{\lambda}) = \lVert\mathbf{r}^{(k)} - \phi_{\lambda}x\rVert_2^2$ run: $\boldsymbol{\lambda}^{(j+1)} = \boldsymbol{\lambda}^{(j)} - \frac{\nabla_{\boldsymbol{\lambda}} J(\boldsymbol{\lambda}^{(j)})}{\boldsymbol{H}_{\boldsymbol{\lambda}} J(\boldsymbol{\lambda}^{(j)})}$	to minimize: $J_{\text{HD}}(\boldsymbol{\lambda}) = \lVert\mathbf{r}_{\text{HD}}^{(k)} - \gamma_{\odot}(x) \odot \mathbf{s}_{\odot}(\boldsymbol{\lambda})\rVert_2^2$ run: $\boldsymbol{\lambda}^{(j+1)} = \boldsymbol{\lambda}^{(j)} - \frac{\nabla_{\boldsymbol{\lambda}} J_{\text{HD}}(\boldsymbol{\lambda}^{(j)})}{\boldsymbol{H}_{\boldsymbol{\lambda}} J_{\text{HD}}(\boldsymbol{\lambda}^{(j)})}$
coefficient computation	$x_{\tilde{m}}^{(k)} = \phi_{\tilde{m}}^{\intercal}\overline{\mathbf{r}}^{(k)}$	$x_{\tilde{m}}^{(k)} = \mathbf{s}_{\odot}(\boldsymbol{\lambda}^{(j)})^{\intercal}\overline{\mathbf{r}}_{\text{HD}}^{(k)}$
residual update	$\mathbf{r}^{(k+1)} = \mathbf{r}^{(k)} - 2\Re\{x_{\tilde{m}}^{(k)}\phi_{\tilde{m}}\}$	$\mathbf{r}_{\text{HD}}^{(k+1)} = \mathbf{r}_{\text{HD}}^{(k)} - \gamma_{\odot}(x_{\tilde{m}}^{(k)}) \odot \mathbf{s}_{\odot}(\boldsymbol{\lambda}^{(j)})$

Table 3.1: Comparison of a MP atomic decomposition steps and our HD-AD.

We conclude this chapter the way we started it: with a discussion of atom parameter vectors. The typical parameter encoding vector we stated in equation (3.2) is a $Q \times M \times 1$ dimensional data structure which encodes M atoms of Q parameters in a minimally redundant manner. Expanding the "value" dimension allows for the M and Q dimensions to be collapsed entirely, which we now understand is possible because of the properties of binding and superposition. Here we remind the reader of another high dimensional encoding which also allows the M and Q dimensions to be collapsed, the time-signal $\mathbf{y}$ itself! The time-domain waveform of an atom encodes the parameter values via the expression of the atom prototype $\phi_{\dagger}(\boldsymbol{\lambda})$. Moreover, multiple atoms parameters are encoded

by summing the atom waveforms (superposition).

While high dimensional time-signal representations expand in size in accordance with the atom parameters, VFA hypervectors do not. Another advantage of HDC which we have emphasized less is the *distributed* nature of VFA encodings. As we can see by looking at the window of an atom, the parameter value encoding is *concentrated* under the non-zero portion of the envelope E. For example, we cannot estimate the damping factor α of a DS by looking at the waveform before its time shift τ. VFA vectors distribute this meaning equally at all points in the vector.

One place the utility of this distributed nature is displayed is within Newton's method. If we think back to parameter refinement with Newton's method using time-signals, the residual energy function $J(\boldsymbol{\lambda}) = \|\mathbf{y} - \phi_\lambda x\|_2^2$ requires a coefficient $x > 0$. However, if $\mathbf{y}$ is a short Gabor atom with a time window of 32 samples and the τ estimate in $\phi_{\boldsymbol{\lambda}}$ is inaccurate by 16 or more samples, then $\phi_{\boldsymbol{\lambda}}^\intercal \overline{\mathbf{y}} = 0$ and $\phi_{\boldsymbol{\lambda}}$'s parameter cannot be refined using Newton's method. The concentrated nature of time-signal encodings is not robust enough to misaligned windows. Encoding $\boldsymbol{\lambda}$ in a distributed manner fixes this problem, and if a high-quality HD encoder exists to generate $\mathbf{z}$, then $\mathbf{s}_\circ(\boldsymbol{\lambda})^\intercal \overline{\mathbf{z}} \gg 0$ will be selected by the argmax and the parameters $\boldsymbol{\lambda}$ can be refined with HD Newton's method.

Furthermore, the "tangling" of parameters in the time-domain, such as the attack parameter β and time shift τ of a REDS, causes refinement to sometimes be difficult with Newton's method [1], where the estimation (or mis-estimation) of one parameter directly affects another. VFA parameter encoding overcomes this problem by "flattening"

parameter encodings in the HD space: all parameters are encoded in the same manner (exponentials raised to a power). No two parameters can become entangled since the VFA encoding base vectors $\mathbf{u}_{\lambda_q}$ are all chosen to be quasi-orthogonal to each other for all q and the particular $\dot{\lambda}_q$ is encoded equally at all positions of the hypervector.

Finally, we point out that VFA vector and time-signals encode symbols in orthogonal directions. Encoding in a time-signal is done *between* vector coordinates – for example, the difference between two samples of a sine tone gives its frequency. VFA vectors encode *within* vector coordinates, by rotating the base vector's value at each position by an amount related to the encoding value $\dot{\lambda}_q$. It is through the ensemble of all of these nuances that make Hyperdimensional Atomic Decomposition possible.

We now turn to the generation of an HD encoding of $\mathbf{y}$ from which $\boldsymbol{\lambda}$ can be extracted. HD vectors have been produced through averaging the mapped signal values in HD space for classification prototypes [32], with spiking neural networks [33] [23], and deep neural networks [34] [29]. We choose deep neural networks as an encoder given their success in other HDC problems, as well as their ability to perform well in noise – an important feature for general audio processing. In Chapter 4 we will give an overview of deep learning techniques as it relates to our problem.

Chapter 4

Deep Learning

Machine learning (ML) consists of algorithms which improve through the use of large amounts of data, a subset of which is deep learning (DL). For audio, DL has been applied with success to problems traditionally solved with digital signal processing (DSP) techniques such as audio synthesis, enhancement, classification, speech recognition, and music information retrieval, to name a few [35]. In particular, it is common to adapt deep learning techniques developed for images to audio given the image-like shape of audio time-frequency representations like the STFT. However, inherent differences between time-frequency representations of signals and 2D spatial images suggest that simply grafting image-processing techniques to audio problems is sub-optimal. At the moment, while developing audio-specific deep learning techniques is an open area of research, many traditional deep learning methods still produce state of the art results for many audio processing tasks. This is one of the reasons why we choose a neural network as our HD encoder. We start by looking at the building block of deep learning models – the artificial neural network.

4.1 Artificial Neural Networks

Artificial neural networks (ANNs), like hyperdimensional computing, are inspired by brain-like computation. Neural networks approximate a function f of some input x to output y by defining a mapping $y = f(x; \boldsymbol{\theta})$ and optimizing its parameters $\boldsymbol{\theta}$ using a training dataset. The assumption is that the training dataset has essential characteristics common to all of its kind, allowing the network to generalize to unseen inputs.

The basic building blocks of ANNs is the artificial neuron, or node. Like biological neurons, ANN nodes are connected to each other. An artificial neuron is defined by

$$h(\mathbf{x}) = g(\mathbf{w}^\intercal \mathbf{x} + b) \tag{4.1}$$

where $\mathbf{x}$ is the input to the neuron, $\mathbf{w}$ is a weight vector, b is the neuron's scalar bias, and g is the neuron's activation function. The use of an activation function is inspired by the behavior of biological neurons, as well as allow networks to model more complex relationships within data. When many artificial neurons connect to each other, the network becomes deep, with the output of each layer becoming the input to the next.

4.1.1 Deep Neural Networks

Deep neural networks (DNNs) consist of multiple layers of artificial neurons. Each layer k consisting of D_k neurons and $h(\mathbf{x})$ can be defined recursively as

$$\begin{aligned} h_k(\mathbf{x}) &= g(\mathbf{W}_k h_{k-1}(\mathbf{x}) + \mathbf{b}_k) \\ h_0 &= \mathbf{x} \end{aligned} \tag{4.2}$$

where $\mathbf{W} \in \mathbb{R}^{D_k} \times \mathbb{R}^{D_{k-1}}$ is a matrix of layer k's weights and $\mathbf{b}_k$ is the layer k's bias vector. Neural networks of this structure are called feed-forward, since information flows from layer to layer, and is never fed back. Layer 0 is the input layer, while layer K is the output layer. Layers $1, ..., K-1$ are known as hidden layers since they do not represent the model input or output, but rather some high dimensional feature space from which patterns in the data can be extracted. The simplest ANN neuron structure, the fully connected (FC), or linear layer, has every neuron of the layer connecting to each neuron of the next layer. This is shown in Figure 4.1.

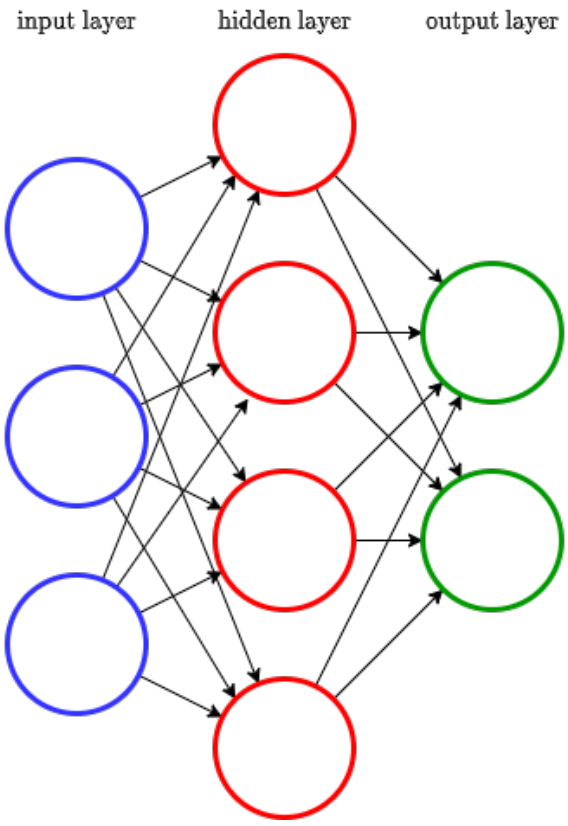


Figure 4.1: Fully connected neural network with 1 hidden layer.

The maximum connectivity of FC neurons causes the feature space to describe more

global characteristics, since behavior at any one neuron is sent to every other neuron, no matter how far away it is. Because VFA vectors encode information globally, we will use an FC layer at the output of our network to generate $\mathbf{s}_{\circledast}(\boldsymbol{\lambda})$. In contrast to the FC structure, a *convolutional* layer computes a hidden feature space not as a matrix multiplication but as a discrete convolution between a sliding kernel and the layer's input. The sliding kernel results in more local activity in the network. A network which contains convolutional layers is known as a convolutional neural network (CNN) [36]. CNNs are widely used in the field of image processing as well as in audio, in part because of the importance of location in the input data such as edges in space, or energy distribution in the time-frequency plane.

4.1.2 Convolutional Neural Networks

Convolutional layers compute the discrete convolution between a kernel h and an input x where the weights of h are learnable parameters, just like those in equation (4.2) [37]. While $1D$ convolutional kernels exist, we specify the more common $2D$ case, which is what we will use. For a $2D$ kernel w, the convolution with input x is

$$(x \star w)[m,n] = \sum_{p}\sum_{q} x[p,q]w[p-m,q-n] \tag{4.3}$$

Furthermore, since x often has more than 1 channel – such as red, green, blue (RGB) for images, or real-imaginary for audio STFT representations – equation (4.3) is performed to each input channel, for each output channel. Thus, to compute an output feature space of J channels from an input of I channels, $I \cdot J$ kernels are defined – one kernel for each input/output channel pair – and the output is summed [37]. The output feature space $\mathbf{c}_j$ is

computed via

$$\mathbf{c}_j = \sum_{i=1}^{I} \mathbf{w}_{i,j}\mathbf{x}_i \tag{4.4}$$

Then, like in equation (4.1), a bias is added to $\mathbf{c}_j$ and we pass the sum through an activation function to generate the input to the next layer $\mathbf{y}_j$,

$$\mathbf{y}_j = h(\mathbf{x}) = g(\mathbf{c}_j + \mathbf{b}_j) \tag{4.5}$$

The $2D$ convolutional kernels h are typically square between $3-7$ units tall and the filter $\mathbf{w}_{i,j}$ moves along the input feature space by some stride amount in $\mathbb{Z}^+ \times \mathbb{Z}^+$. This movement along the feature space is what captures local activity in data [38]. Striding determines the size of the output feature space and can be used to down-sample the layer's input. On the other hand, if the feature size is to remain the same the filter slides by one and the feature space is to be padded at the edges. The interested reader is directed to [39] for a complete discussion on CNNs.

4.1.3 Activation Functions

The activation function g allows the network to model very complex relationships between data by introducing non-linearities into the flow of information. Without these non-linearities, the network's ability to perform even simple tasks is severely limited. Furthermore, any deep neural network without non-linear activation functions can be rewritten as a single layer network [37].

The sigmoid function

$$g(x) = \frac{1}{1+e^{-x}} \tag{4.6}$$

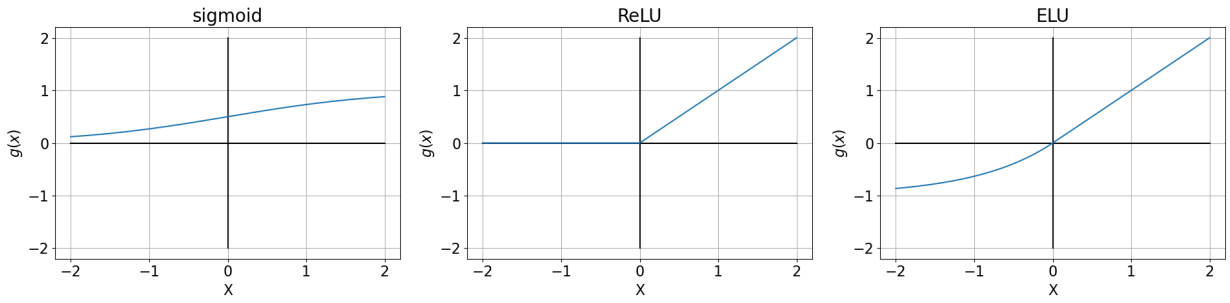
was a common activation function in early neural networks. However, it is often replaced with a rectified linear unit (ReLU). ReLU passes positive values but zeros negative ones

$$g(x) = \max(0, x) \tag{4.7}$$

Experimental results have shown that ReLU activations converge faster than sigmoid, as well as produce superior results in some cases [40]. An activation function related to ReLU, the exponential linear unit (ELU), also passes positive values but attenuates negative values rather than zeroing them. ELU is defined in equation (4.8)

$$g(x) = \begin{cases} x & x > 0 \\ \alpha(e^x - 1) & x \leq 0 \end{cases} \tag{4.8}$$

where it is typical for $\alpha = 1$. Notably, ELU's derivative is continuous at $x = 0$, while ReLU's derivative is not. ELU has been shown to suffer less from the dying neuron problem than ReLU, as well as generally converge faster [41]. These three activation functions are shown in Figure 4.2.



(a) **(b)** **(c)**

Figure 4.2: Different activation functions $g(x)$.

By and large, selecting the proper activation function is done more by trial and error than being an exact science.

4.2 Gradient-Based Optimization

Training a neural networks is the process of tuning the network parameters $\boldsymbol{\theta}$ so that a desired output is achieved. These parameters are adjusted according to a loss function which, for supervised learning problems, typically takes the ground truth and the model's output and compares them. The loss function should be differentiable as well as chosen such that the minimization of the loss results in the desired output from the network. However, because of the non-linearity introduced by the network activation functions, the loss function is non-convex and cannot be minimized with classic convex optimization techniques. Thus, model parameters are adjusted according to gradient descent.

For a loss function L, the derivative of each network parameter is computed with respect to L. A typical loss function, and the one used in this project, is mean squared error (MSE). For some model input $\mathbf{x}$, the MSE between the model output $\hat{\mathbf{y}} = f(\mathbf{x}; \boldsymbol{\theta})$ and the ground truth $\mathbf{y}$ is

$$L_{\text{MSE}} = \frac{1}{N} \sum_{i=1}^{N} \|\hat{\mathbf{y}}_i - \mathbf{y}_i\|_2^2 \tag{4.9}$$

Since we want to minimize the loss, we then adjust the network parameters in the direction of steepest descent by some amount ϵ, known as the learning rate.

$$\boldsymbol{\theta}_{n+1} = \boldsymbol{\theta}_n - \epsilon \nabla_{\boldsymbol{\theta}} L \tag{4.10}$$

A true gradient descent would compute the loss for the entire set of training data, and then perform equation (4.10). However, in practice this can be very memory and computationally expensive. Instead, the loss is computed for a smaller subset of the training data called a *batch*. The gradient computation and subsequent parameter updates are performed per batch. This not only allows networks to scale to very large datasets, but also has been shown to converge faster [42]. In order to further speed-up neural network training, the gradient is computed via back-propagation, an algorithm which uses the chain-rule in calculus to avoid computing individual derivatives multiple times by storing the derivative for each parameter at each layer of the network for reuse deeper in the network [43]. A complete discussion of back-propagation is given in [37].

4.3 Models

As mentioned earlier, deep learning for audio often involves applying techniques developed for image processing to $2D$ audio representations such as the STFT. In addition to the STFT, other $2D$ representations of sound are used for input to deep learning models such as the Log-mel spectrogram and the constant-Q spectrogram [35]. Outside of the audio-specific features fed to the neural network, there have been attempts which try to incorporate audio- and DSP- specific ideas into deep learning layers. These include SincNet, which learns cut-off frequencies of sinc functions used for the task of speech recognition [44], and harmonic convolutions, which incorporates the harmonic spacing of related partials into $2D$ convolutions [45]. We adopt traditional elements of CNNs which take as input complex-valued STFTs.

We define a general CNN structure which is flexible and performs well in our experiments, shown in Figure 4.3. It has elements borrowed from other neural network architectures such as residual connections from ResNet [46] and a fully-connected layer which generates the final HD vector [29]. Using convolutional layers at the head of the network and fully connected (FC) layers at the tail makes sense since we are trying to translate the local time-frequency characteristics of the signal into HD vectors which encode them "globally" by distribution along the entire output vector. Each layer performs down-sampling of the input features by some amount $\mathrm{ds}_k \in \mathbb{Z}^+ \times \mathbb{Z}^+$. This is done with a max-pooling operation which strides by ds_k. Max pooling searches a patch of feature space and returns only the maximum value in that area [47]. The fixed size N_{HD} of VFA atom encodings works well with neural networks because they too have layers of fixed size.

The filter_k modules in the model can either be chosen to be stacked $2D$ convolutional units, or convolutional gated linear unit (GLU), or convGLU. The stacked 2D convolutional units are built from two 2D filters, one right after the other separated by an activation function. The second filter's kernel has size 1×1, which effectively scales the input feature. A similar structure has been used in popular audio CNNs such as WaveNet [48], a state of the art model for speech generation.

The convolutional GLU is defined in equation (4.11). Output channel j of a convGLU layer is

$$(\mathbf{c}_j + \mathbf{b}_j) + \sigma((\mathbf{c}_j + \mathbf{b}_j)) \tag{4.11}$$

where $\mathbf{c}_j$ and $\mathbf{b}_j$ are defined from equation (4.4) and (4.5), and σ is the sigmoid activation function. The idea behind gating is that the flow of information is controlled to allow for more

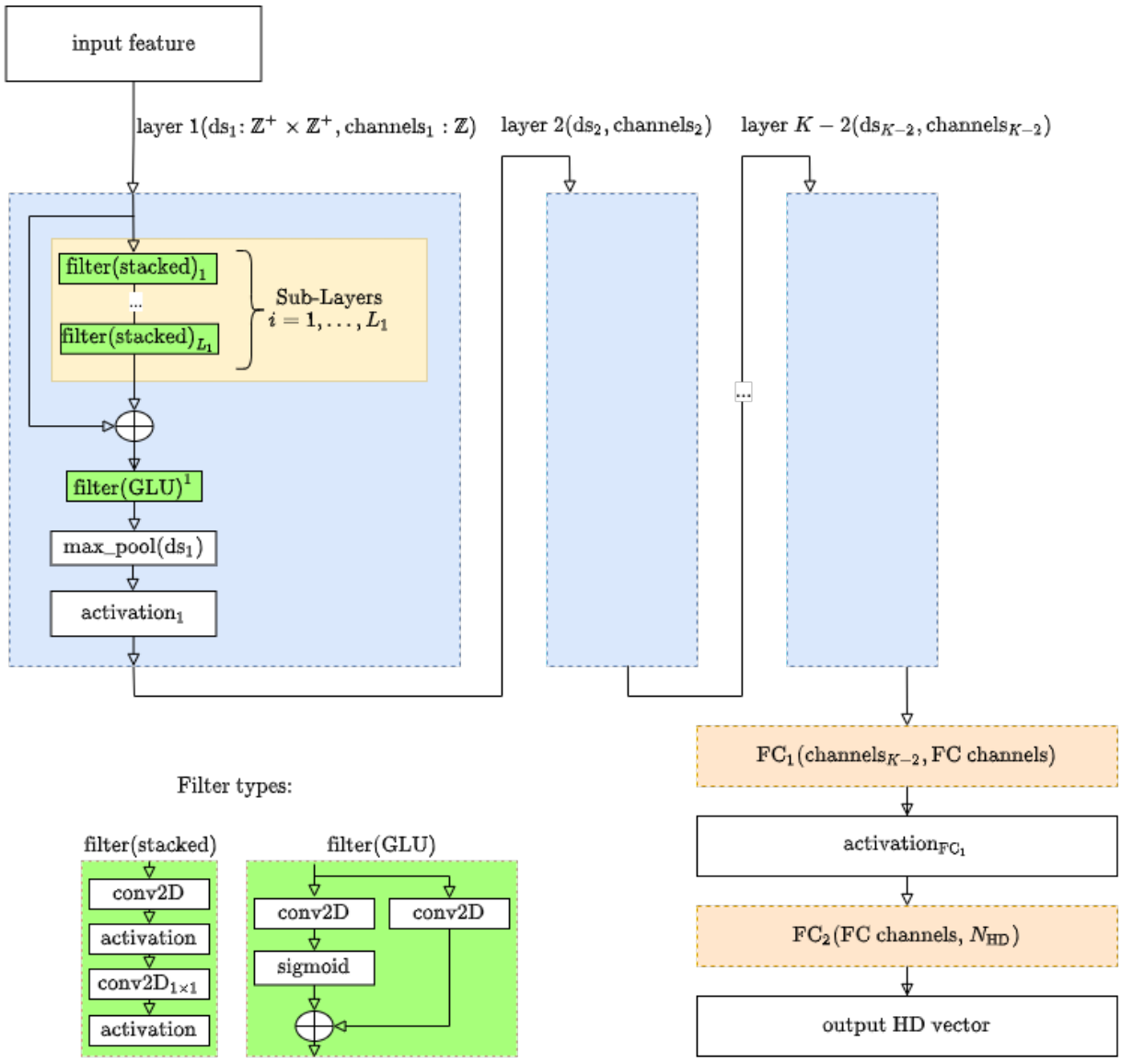


Figure 4.3: Generic Model Structure. ds_k is the down-sampling amount for the given layer, performed by striding by ds_k with a 2D max-pooling unit. $\text{conv2D}_{1\times1}$ denotes a conv2D unit with a 1×1 kernel. filter_k(GLU) is the convGLU unit defined in equation (4.11). For our experiments, the input feature is a complex-valued audio STFT.

complex interactions within the network [49]. ConvGLU layers have a history of success in audio deep neural networks [50]. The exact model architecture changes for each experiment

is detailed in the Appendix A for each experiment in the next chapter.

4.3.1 Training

Code for training and testing our neural networks is written in Python 3 using the PyTorch [51] deep learning library. The actual training took place on machines at Compute Canada[1], an organization that provides computing resources to researchers in Canada. These resources not only include hardware accelerators like graphical processing units (GPUs), but also the software infrastructure which allow us to train with multiple GPUs running in parallel. Our training jobs ran on the Cedar compute cluster[2] and were allocated 64 GB of random access memory (RAM), 4 GPUs (NVIDIA P100), and 6 CPU cores per GPU for a total of 24 CPU cores. With the availability of so many high power resources we were able to train our models within a week, allowing for downtime after the maximum 24 job time and having to wait in the queue when resuming training.

The testing and evaluation of our models is done locally on an Intel Core i5 2.9GHz central processing unit (CPU) with 16 GB of RAM. Consequently the reported speeds of our experiments are not recorded for hardware accelerated machines with GPUs.

We conclude this chapter with a discussion of existing atomic decomposition projects with neural networks, and ours. Existing atomic decomposition methods which incorporate neural networks do so in order to "learn" atoms in a data-driven manner, as we discussed at the beginning of Chapter 2. These waveforms are typically tailored to one class of signals, such as speech, and are not parameterized meaning flexible resynthesis cannot be achieved. These

[1] `https://www.computecanada.ca/`
[2] `https://docs.computecanada.ca/wiki/Cedar`

other investigations aim to increase sparsity by maximizing the synthesis criteria of atoms. However, the absence of parameters means little insight into the sound generating structures of the signal can be made. Furthermore these systems still rely on the computation of many time-domain correlations. Our approach is unique in that it uses atoms which satisfy both synthesis and analysis criteria of atoms and use a neural network to generate the coefficient x, rather than relying on an algorithm which computes time-domain correlations.

Chapter 5

Experiments

In this chapter we realize the atom encoding developed in Chapter 3, of atoms discussed in Chapter 2, using the neural networks like the ones outlined in Chapter 4. The complete HD-AD system is tested on signals which are a synthetic mixture of atoms plus noise, synthesizer sounds, and real-world recordings in order to demonstrate its ability to generate atomic decompositions. In addition, we introduce a hybrid algorithm – neural network accelerated matching pursuit (NN-MP) – which builds a dictionary of atoms based on the parameters of HD-AD, but computes the gain coefficient in the time domain like MP. We show this approach can be extended to MP variants such as OMP.

To recap, our goal is to generate an encoding of a time-signal $\mathbf{y}$ using the VFA encoding paradigm, from which atomic parameters can be extracted. To do this, we rewrite the traditional expression of $\mathbf{y}$ as a linear combination of waveforms

$$\mathbf{y} = \sum_k x_k \phi_{\boldsymbol{\lambda}^k}$$

to an expression $\mathbf{z}$ which is a linear combination of encoded atoms in HD space

$$\mathbf{z} = \sum_k \gamma_{\circledast}(x_k) \circledast \mathbf{s}_{\circledast}(\boldsymbol{\lambda}_k)$$

where $\circledast$ is the circular convolution binding operation discussed in section 3.2. In order to train a neural network to generate these encodings, we create datasets of signals $\mathbf{y}$ using the atoms discussed in Chapter 2, and feed their STFT representation as input to variations of the CNN in Figure 4.3, using $\mathbf{z}$ as the model's ground truth. When building $\mathbf{z}$, we generate base encoding vectors $\mathbf{u}_{\lambda_q}$ for each atom parameter λ_q, as discussed in section 3.3. These VFA base vectors are stored and used to decode the neural network output via HD Newton's method, as discussed in section 3.4.

Parameter Rescaling

In theory, a perfect encoder and decoder should be able to construct $\mathbf{z}$ so that the precise set $\boldsymbol{\lambda}^k$ for all k is retrieved. However, in practice the interaction between the time-domain synthesis parameters $\boldsymbol{\lambda}$ and the HD encoding space leads to a sub-optimal representation, and we make adjustments to equation (3.35) in order to correct for these and achieve better performance. This means choosing encoding values $\dot{\boldsymbol{\lambda}}$ (to use in $\mathbf{s}_{\circledast}(\dot{\boldsymbol{\lambda}})$) which correspond to the time-domain synthesis parameters $\boldsymbol{\lambda}$ (used in $\phi_{\boldsymbol{\lambda}}$).

In order to illustrate the necessity of encoding values $\dot{\lambda} \neq \lambda$, recall from Chapter 3 that the similarity of two real values r_1 and r_2 in HD space is given by the sinc of their difference.

$$\langle \mathbf{u}(r_1), \mathbf{u}(r_2) \rangle = \text{sinc}(r_1 - r_2)$$

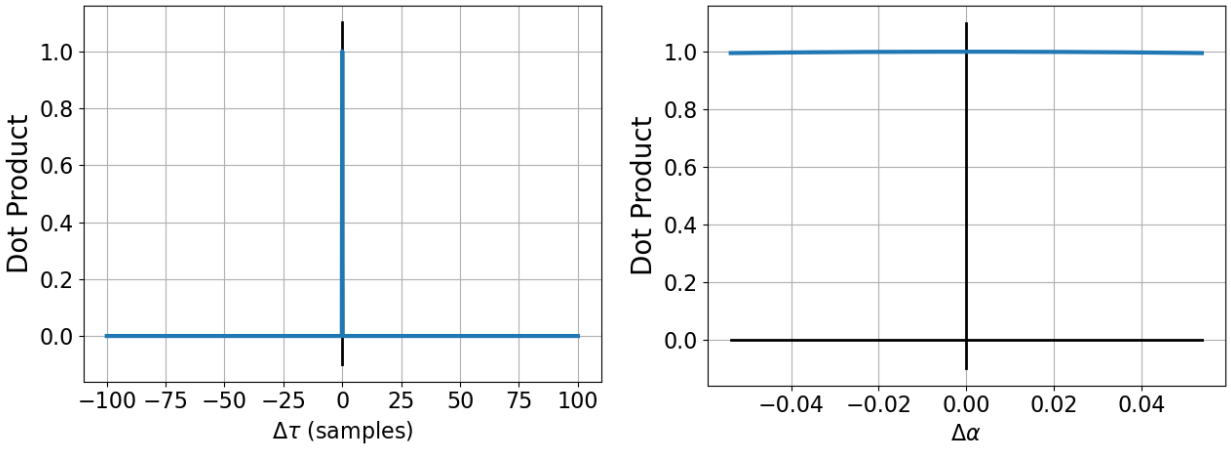
For certain parameters λ, encoding the synthesis values (used in $\phi_{\boldsymbol{\lambda}}$) leads to bad encoding similarity behavior for a neural network trying to learn the encoding. For example, consider the time shift parameter τ and damping factor α of a DS atom. Since τ is given in an integer number of samples then

$$\langle \mathbf{u}_\tau(\tau_1), \mathbf{u}_\tau(\tau_2) \rangle = \text{sinc}(\tau_1 - \tau_2) = 0 \quad \forall \tau_1, \tau_2 \text{ if } \tau_1 \neq \tau_2 \tag{5.1}$$

since $(\tau_1 - \tau_2) \in \mathbb{Z}$. Furthermore, for a wide range for α, $[\min(\alpha), \max(\alpha)]$, where $\min(\alpha) = 1.439e^{-4}$ and $\max(\alpha) = 5.397e^{-2}$, corresponding to DS window lengths of 3 seconds and 8 milliseconds, respectively, the encoding similarity between the endpoints of the range is

$$\langle \mathbf{u}_\alpha(\max(\alpha)), \mathbf{u}_\alpha(\min(\alpha)) \rangle = \text{sinc}\Big(\max(\alpha) - \min(\alpha)\Big) = \text{sinc}(0.054) = 0.995 \approx 1$$

Figure 5.1 illustrates this encoding similarity behavior for each parameter. For a neural



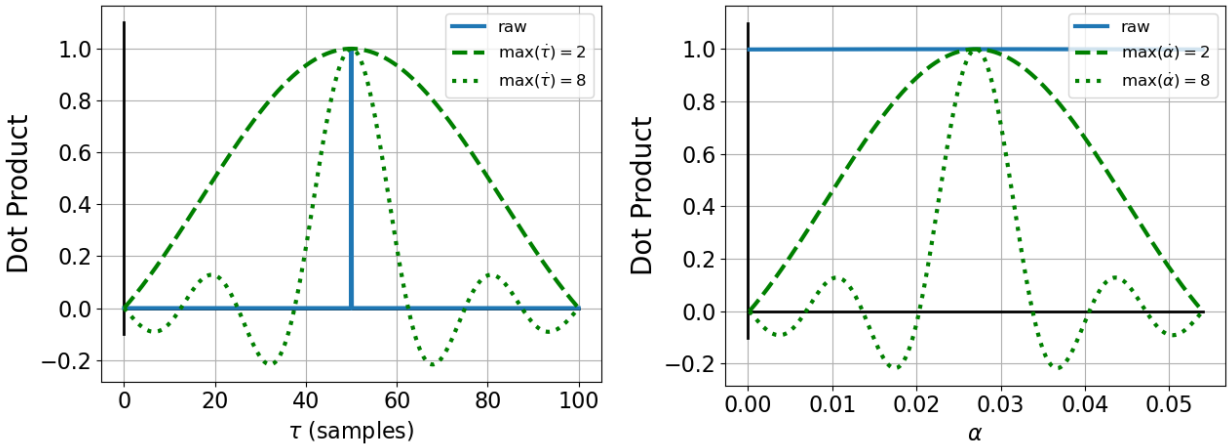
(a) **(b)**

Figure 5.1: Encoding similarity between raw synthesis values parameterized by their difference for $\Delta\tau$ **(a)** and $\Delta\alpha$ **(b)**.

network trying to learn the HD parameter encoding space, each situation is sub-optimal. On the one hand, any τ estimate by the network has an encoding similarity of 0 with the ground truth unless it is sample accurate. On the other, the worst the network can do for predicting α in any situation results in an encoding which is nearly identical to the one it is trying to learn. These parameters illustrate encoding similarity behavior at two extremes – nearly impossible to get right, and nearly impossible to get wrong – both of which lead to poor conditions for training neural networks. In order to correct for this, we map the raw synthesis parameters λ to corresponding encoding values $\dot{\lambda}$ in order to better tune the similarity shape of nearby parameter encodings in the HD space.

The range of synthesis parameters in the training datasets is $[\min(\lambda), \max(\lambda)]$. In order to generate the corresponding encoding parameter $\dot{\lambda}$ in $[\min(\dot{\lambda}), \max(\dot{\lambda})]$, we define a mapping function $\dot{g}_\lambda : \lambda \mapsto \dot{\lambda}$. In many cases, $\dot{g}_\lambda$ is a linear map, so choosing $\dot{g}_\lambda$ is synonymous with rescaling λ values from $[\min(\lambda), \max(\lambda)]$ to $\dot{\lambda}$ in $[\min(\dot{\lambda}), \max(\dot{\lambda})]$. However, for certain parameters the map is more complicated, as we will see. Figure 5.2 shows how choosing $[\min(\dot{\lambda}), \max(\dot{\lambda})]$ affects the similarity of nearby parameter encodings. Furthermore, we systematically set $\min(\dot{\lambda}) = 0$ for all λ, so the parameter HD encoding range is specified in terms of $\max(\dot{\lambda})$, which is alternatively notated as $\mu_{\dot{\lambda}}$.

We see that for both extremes of encoding similarity, rescaling the encoding parameter range reshapes the similarity behavior of nearby points which better reflects the similarity of their corresponding waveforms. The rescaling of these parameters for encoding allows us to tune the similarity behavior of nearby points to a desired level of precision.



(a) $\tau = 50$ **(b)** $\alpha = 2.706e^{-2}$

Figure 5.2: Encoding similarity of nearby parameter values to τ and α.

Changing the encoding range by expanding or contracting $\max(\dot{\lambda})$ results in encodings that are harder or easier for our neural network to learn. If we again consider Figure 5.2a, and the similarity between the encoding for $\tau_1 = 50$ and $\tau_2 = 62.5$, we see that for $\max(\dot{\lambda}) = 2$,

$$\mathbf{u}_\tau(\tau_1)^\intercal \overline{\mathbf{u}_\tau(\tau_2)} = 0.900$$

However, increasing $\max(\dot{\lambda})$ to 8 results in

$$\mathbf{u}_\tau(\tau_1)^\intercal \overline{\mathbf{u}_\tau(\tau_2)} = 0.000$$

For a neural network trying to learn these encodings, this means that it can achieve a lower loss value (MSE) for a bad parameter estimate the smaller the parameter HD encoding range is. However, if the parameter encoding range is too large, the space once again becomes too sparse and the proper encoding becomes hard to find unless you are right on top of it. Thus

$\max(\dot{\lambda})$ is a hyperparameter that must be tuned in order to balance the trade-off between encodings that are learnable by the network and accurate for good reconstruction in the time-domain.

Input Data

The encodings in equation (3.35) are also modified to accommodate the fixed size input feature used by CNNs. Because we want to be able to decompose signals of any length, we feed the neural network *tiles* of the time-frequency plane meaning that the time and frequency parameters are encoded *local* to that tile. The benefits of this are two-fold. First it allows us to compute high frequency resolution STFTs for long signals without worrying about the neural network becoming too large. Second, the encoding of time and frequency parameters locally means that we can create $\mathbf{\Phi}_{\text{HD}}$ for the *tile area only*, and reuse it throughout the time-frequency plane. This is possible because these atoms are time-frequency shift invariant, as discussed in Chapter 2. Being able to reuse $\mathbf{\Phi}_{\text{HD}}$ vastly reduces its size M_{HD} and consequently lowers the number of HD correlations that need to be computed. Figure 5.3 illustrates this sub-sampling of the time-frequency plane.

We choose a *dual*-resolution STFT to take tiles from and feed to the neural network. A similar idea is used in [52] for the task of automatic speech recognition with DNNs. Via the Heisenberg principle, time resolution must be sacrificed in order to achieve high frequency resolution, and vise-versa. We try and somewhat overcome this constraint by calculating two STFTs, one which is high in frequency resolution, and the other which is high in time resolution. The high frequency resolution STFT has an FFT size of N_{FFT} and a hop size of

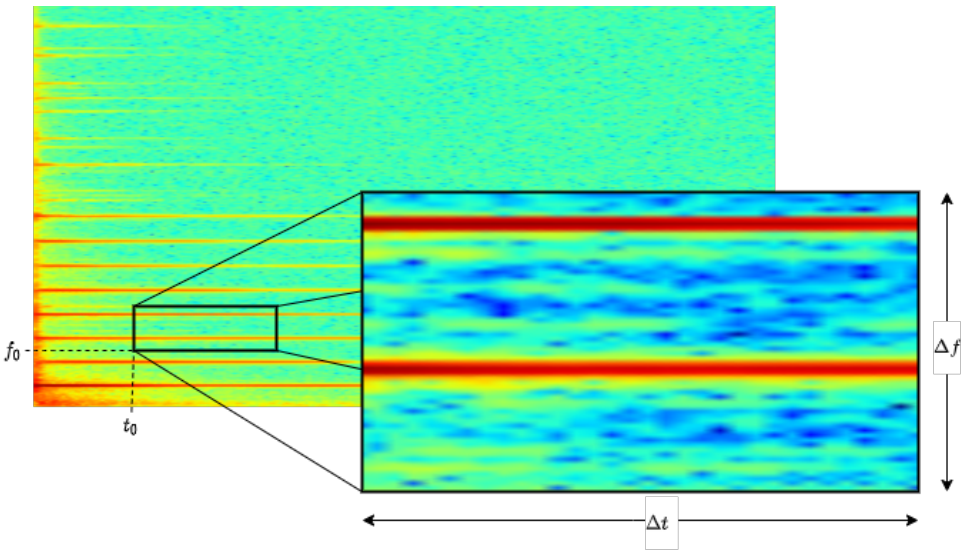
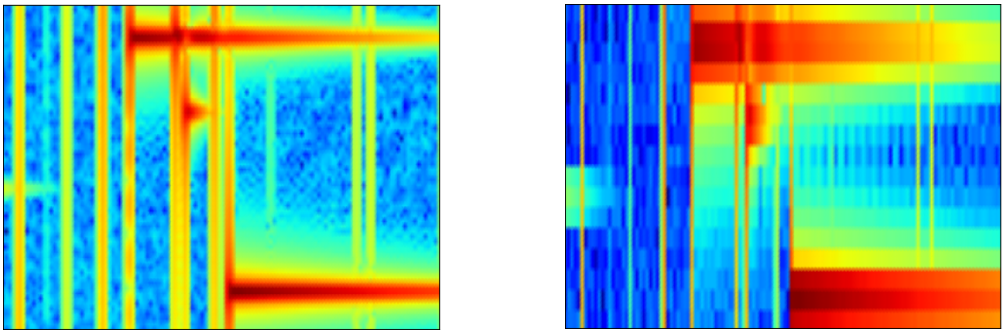


Figure 5.3: Sub-sampled STFT input tiles.

$\frac{N_{\text{FFT}}}{4}$. The high time resolution STFT has an FFT size of $\frac{N_{\text{FFT}}}{4}$ and a hop size of $\frac{N_{\text{FFT}}}{4}$, i.e. no time overlap between frames. In order to create time-frequency tiles from both STFTs whose time-frequency coordinates correctly correspond to each other, the frequency axis of the high time resolution STFT is upsampled by a factor of 4. Figure 5.4 shows tiles from these two STFTs of a mixture of DS atoms. Figure 5.4a better localizes horizontal structure (partials) and Figure 5.4b better localizes vertical structures (transients).

Neural-Network Accelerated Matching Pursuit

Finally we point out that HD-AD can be used as a pre-processing step to traditional atomic decomposition algorithms. By disregarding the coefficients x_k predicted by the network, the parameters $\boldsymbol{\lambda}_k$ can be used to populate a dictionary of waveforms $\boldsymbol{\Phi}$ for use with matching pursuit or its derivatives like orthogonal matching pursuit. This is a similar idea to methods of additive synthesis which use STFT peaks to "prune" a dictionary of



(a) $B_t = 64$ ms, $B_f = 15.625$ Hz **(b)** $B_t = 16$ ms, $B_f = 62.500$ Hz

Figure 5.4: Dual resolution STFT tiles and their corresponding time bandwidth B_t and frequency bandwidth B_f. **(a)** high frequency resolution. **(b)** high time resolution.

Gabor atoms. However, our neural network pre-processing goes beyond just time and frequency parameter estimation by being able to approximate parameter values for asymmetric envelopes. This greatly increases the tractability of MP with these waveforms. We call this system neural network accelerated matching pursuit (NN-MP). The same approach can be taken with variants of MP, such as OMP, for a system called neural network accelerated orthogonal matching pursuit (NN-OMP).

Aside from the inherent iterative structure from running MP, NN-MP introduces another level of iteration. Once MP has been run using a dictionary of atoms populated from the neural network output, the residual is fed back into the neural network and the process is repeated until some stopping criteria. We casually note that this somewhat resembles a clean-up memory, a common step in some HDC systems, which retrieves clean vectors from noisy ones. Interestingly, here the time-signal itself acts like some sort of clean-up memory by allowing previously misidentified or completely unidentified atoms to be extracted after any potential masking has been removed in the time-domain.

Algorithm

The atomic decomposition system, which includes both HD-AD and NN-MP, can be summarized in the following steps.

1. **Initialize**: *given* time-signal $\mathbf{y}$, HD dictionary $\mathbf{\Phi}_{\text{HD}}$
   (a) $k = 0$
   (b) residual $r^{(k)} = \mathbf{y}$.
   (c) approximation $\hat{\mathbf{y}} = \mathbf{0}$ is a signal of zeros.
   (d) global dictionary $\mathbf{\Pi} = \emptyset$ is empty.
   (e) SRR threshold = 30dB.
2. **Hyperdimensional Atomic Decomposition (HD-AD)**:
   (a) Compute dual-resolution STFT of $r^{(k)}$ and feed tiles i, j to the neural network to generate the encodings $\mathbf{z}_{i,j}$.
   (b) Run Algorithm 3 (decoding VFA vectors from section 3.4) for each $\mathbf{z}_{i,j}$
   (c) Map the HD parameter encodings to the synthesis parameter space via $\dot{g}_{\boldsymbol{\lambda}}^{-1}$
   (d) *If* NN-MP: go to step 3. *Else*: reconstruct the time-domain signal using $\gamma_{\circledast}(x)$, **done**.
3. **Neural Network accelerated Matching Pursuit (NN-MP)**:
   (a) Build the matrix dictionary $\mathbf{\Phi}$
   (b) Run MP or OMP to retrieve the coefficients for atoms in the dictionary.

(c) Reconstruct the time-signal, update the approximation $\hat{\mathbf{y}}$, update the residual $r^{(k+1)}$. $k = k + 1$.

(d) *If* SRR is above threshold: **done.**

Else: repeat steps 2a-2d.

and summarized graphically in Figure 5.5.

For HD-AD we measure its performance in terms of sparsity ($\|\mathbf{x}\|_0$) and speed ($\times$'s real-time $= \frac{\text{signal duration}}{\text{HD-AD time}}$). For NN-MP and NN-OMP we measure performance by sparsity and SRR.

5.1 HD-AD with Asymmetric Atoms

In this section we present the results of Hyperdimensional Atomic Decomposition. As discussed earlier, MP can take up to 100$\times$'s slower than real-time to generate a decomposition with simple symmetric atoms of two parameters and scales poorly with the signal length N. In contrast, we show that HD-AD scales with the sparsity of $\mathbf{y}$, $\|\mathbf{x}\|_0$, irrespective of N. For the additive model of synthesis this means atomic decompositions with asymmetric atoms are generated in at minimum real-time, and in certain situations much faster. For the source-filter model of audio synthesis HD-AD runs slower than real-time, however atomic decompositions are still generated in a reasonable amount of time and at a much lower computational cost than MP.

Our current neural network encoder only extracts the amplitude $|x|$ and not the phase. We found that training a network to estimate the phase ultimately made little difference in

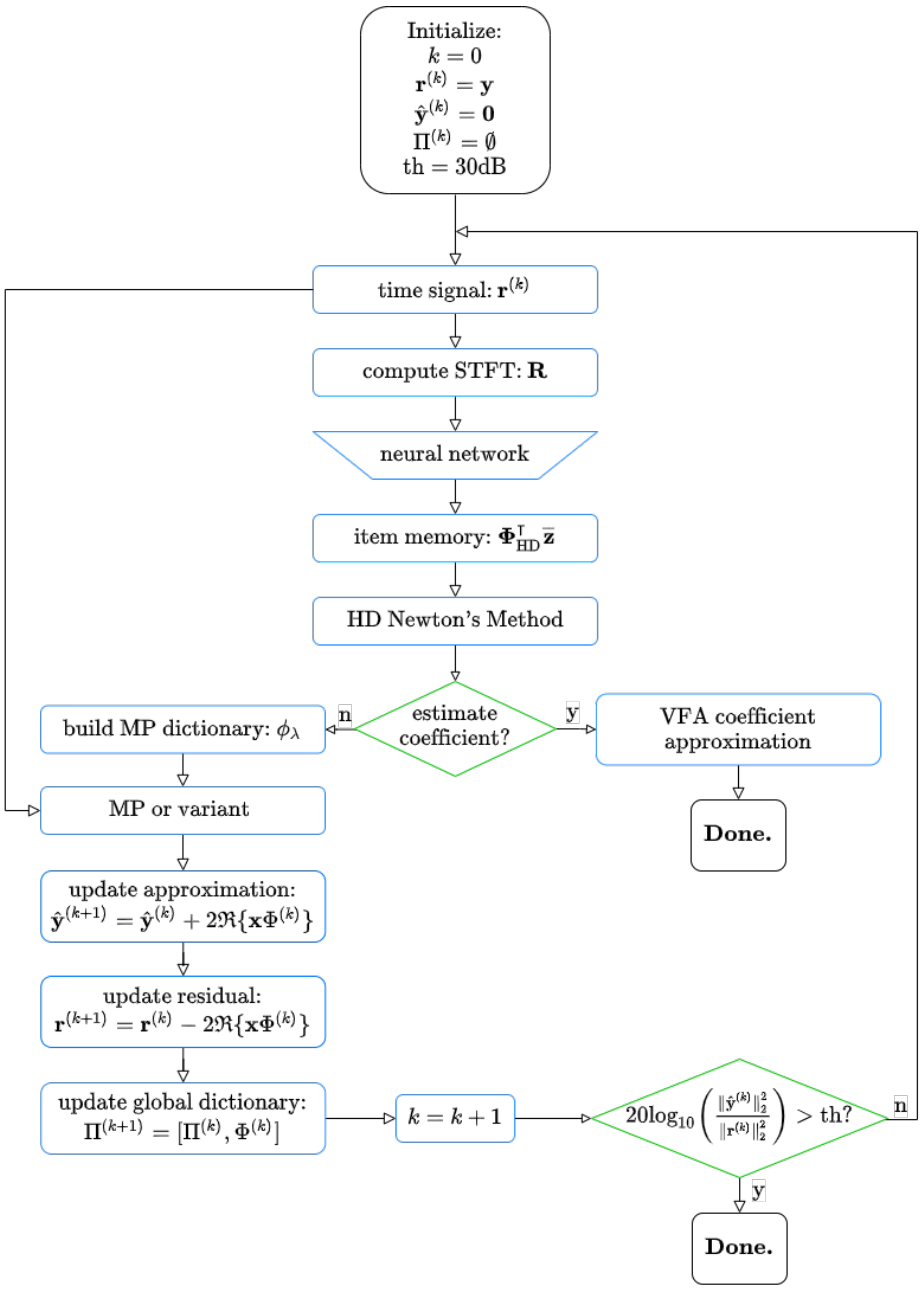


Figure 5.5: Atomic Decomposition System for signal $\mathbf{y}$.

terms of reconstructive quality unless the atom's time shift τ was extremely accurate. Our experiments showed that the network could learn the phase somewhat well, however the

decompositions were ultimately worse and so we decided the extra effort was not worth it. However, we believe it is remarkable that the models were able to learn the phase even to a moderately successful degree, since it involves not only learning the HD encodings of the atom parameter space, but their rotations as well. As a result, HD-AD reconstructs atoms with a constant phase value.

For each experiment, we detail the specific $\max(\dot{\lambda})$ for each atom parameter. The specific neural network architectures are detailed in Appendix A. The decompositions are shown as a waveform and STFT, and compared to the original signal in time and STFT. We report the total time our system takes to produce each decomposition, as well as divide this total into the amount taken by the neural network and decoding steps. Compared to the length of the original signal, this gives us the $\times$'s real-time performance of our HD atomic decomposition system.

5.1.1 Damped Sinusoid Atoms

For experiments with damped sinusoid atoms, we create encoding vectors for time, frequency, and damping parameters. Informal listening tests reveal that the perceptual characteristics of the damping factor are spaced somewhat exponentially rather than linearly. Consequently, α is mapped to $\dot{\alpha}$ via $\dot{g}_\alpha$ using a base value, b. First, α is normalized to α_n given the range of values in the training set via a linear map.

$$\alpha_n = \frac{\alpha - \min(\alpha)}{\max(\alpha) - \min(\alpha)} \tag{5.2}$$

Then we map α_n to $\dot{\alpha}$ via $\dot{g}_\alpha$

$$\dot{g}_\alpha : \alpha_n \in \mathbb{R} \mapsto \frac{b^{\alpha_n} - 1}{b - 1} \cdot \mu_{\dot{\alpha}} \tag{5.3}$$

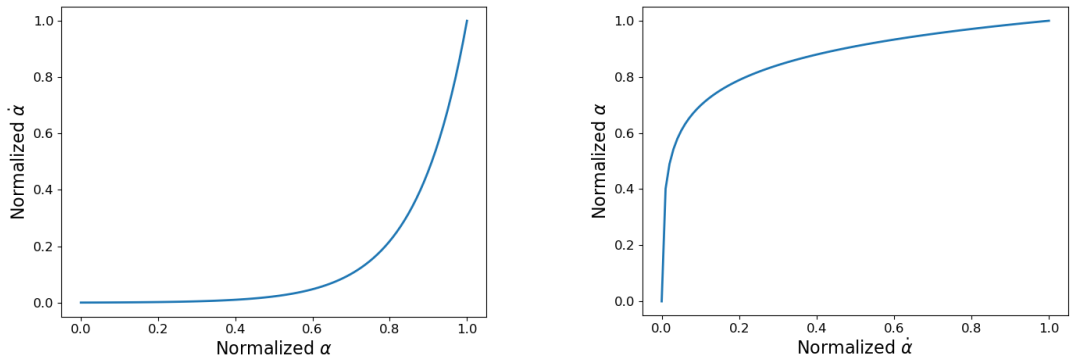
For decoding we retrieve α_n via $\dot{g}_\alpha^{-1}$

$$\dot{g}_\alpha^{-1} : \dot{\alpha} \in \mathbb{R} \mapsto \frac{\log\left(\frac{\dot{\alpha}(b-1)}{\mu_{\dot{\alpha}}} + 1\right)}{\log(b)} \tag{5.4}$$

Finally α is retrieved by rescaling the normalized α_n back to the synthesis range

$$\alpha = \alpha_n(\max(\alpha) - \min(\alpha)) + \min(\alpha) \tag{5.5}$$

Figure 5.6 shows $\dot{g}_\alpha$ and $\dot{g}_\alpha^{-1}$ for $b = 2000$.



(a) $\dot{g}_\alpha$. **(b)** $\dot{g}_\alpha^{-1}$.

Figure 5.6: Normalized α encoding map and decoding map for $b = 2000$. $\dot{\alpha}$ is normalized as well to show the general behavior for any $\max(\alpha)$.

Since these atoms have 3 parameters, the similarity between the encodings of two DS

atoms $\phi_{\mathrm{DS}}(\tau_1, f_1, \alpha_1)$ and $\phi_{\mathrm{DS}}(\tau_2, f_2, \alpha_2)$ is given by the product of 3 sinc functions.

$$\mathrm{sinc}(\dot{\tau}_1 - \dot{\tau}_2) \times \mathrm{sinc}(\dot{f}_1 - \dot{f}_2) \times \mathrm{sinc}(\dot{\alpha}_1 - \dot{\alpha}_2) \tag{5.6}$$

The encoding ranges for our experiments with DS atoms are

$$\begin{aligned} \max(\dot{\tau}) &= 8 \\ \max(\dot{f}) &= 8 \\ \max(\dot{\alpha}) &= 3 \end{aligned} \tag{5.7}$$

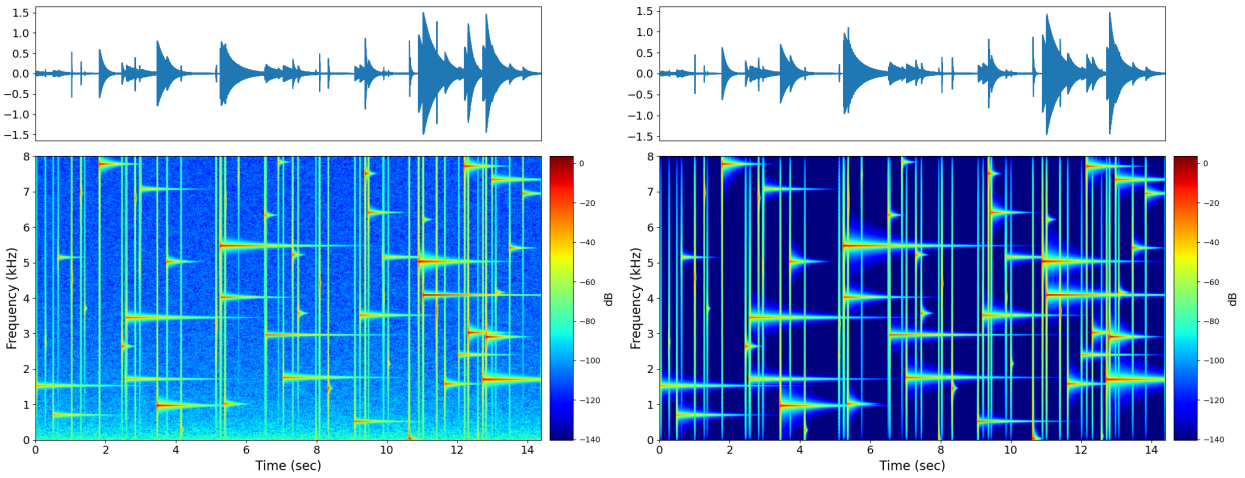
The input STFT tiles to the network are 128 FFT frames wide and 64 FFT bins tall, which have a time span of 2.048 seconds and a frequency range of 1000 Hz.

We train a model on synthetic signals which are the mixture of scaled damped sinusoid atoms and noise. The time and frequency parameters for atoms in the mixture are sampled randomly from the time and frequency support of the signal. Since the damping factor is encoded exponentially, we exponentially sampled a range of α values from

$$\alpha \in [1.439e^{-4}, 5.397e^{-2}] \ \frac{1}{\text{sample}} \tag{5.8}$$

which for a sample rate of 16 kHz corresponds to atoms of length 3 seconds and 8 milliseconds, respectively. In order to give the model enough time to estimate the damping factor, we ensure that long atoms have a time parameter such that at least $\frac{1}{3}$ of its length lies in the local time-frequency tile.

The atoms in the dataset are scaled to be between −30 dB and 0 dB with random phases. When encoded, the amplitude in decibels is scaled between 1 and 6, a range which seemed to work well in our experiments. A decrease in scalar by 1 corresponds to a change in amplitude by −6 dB. $\frac{1}{f}$ filtered white noise is randomly scaled between −60 dB and −30 dB and added to the signals. The model architecture, as well as dataset parameters are given in Table A.1. In Figure 5.7 we show the model decomposition on a mixture of scaled damped sinewave atoms like the ones it was trained on. As can be seen from the reconstruction, the atomic decomposition naturally denoises the input since only the sinusoidal components (atoms) are reconstructed.



(a) Input signal: 56 DS atoms.

(b) Net approx: 56 DS atoms.

Figure 5.7: Decomposing a mixture of DS atoms. Signal length: 14.384 s, HD-AD time: 2.651 s = 5.425×'s real-time.

In Figure 5.8 we show the model decomposition on a synthetic kalimba playing the notes C5 and C6, as well as signal transformations on the kalimba sound given the

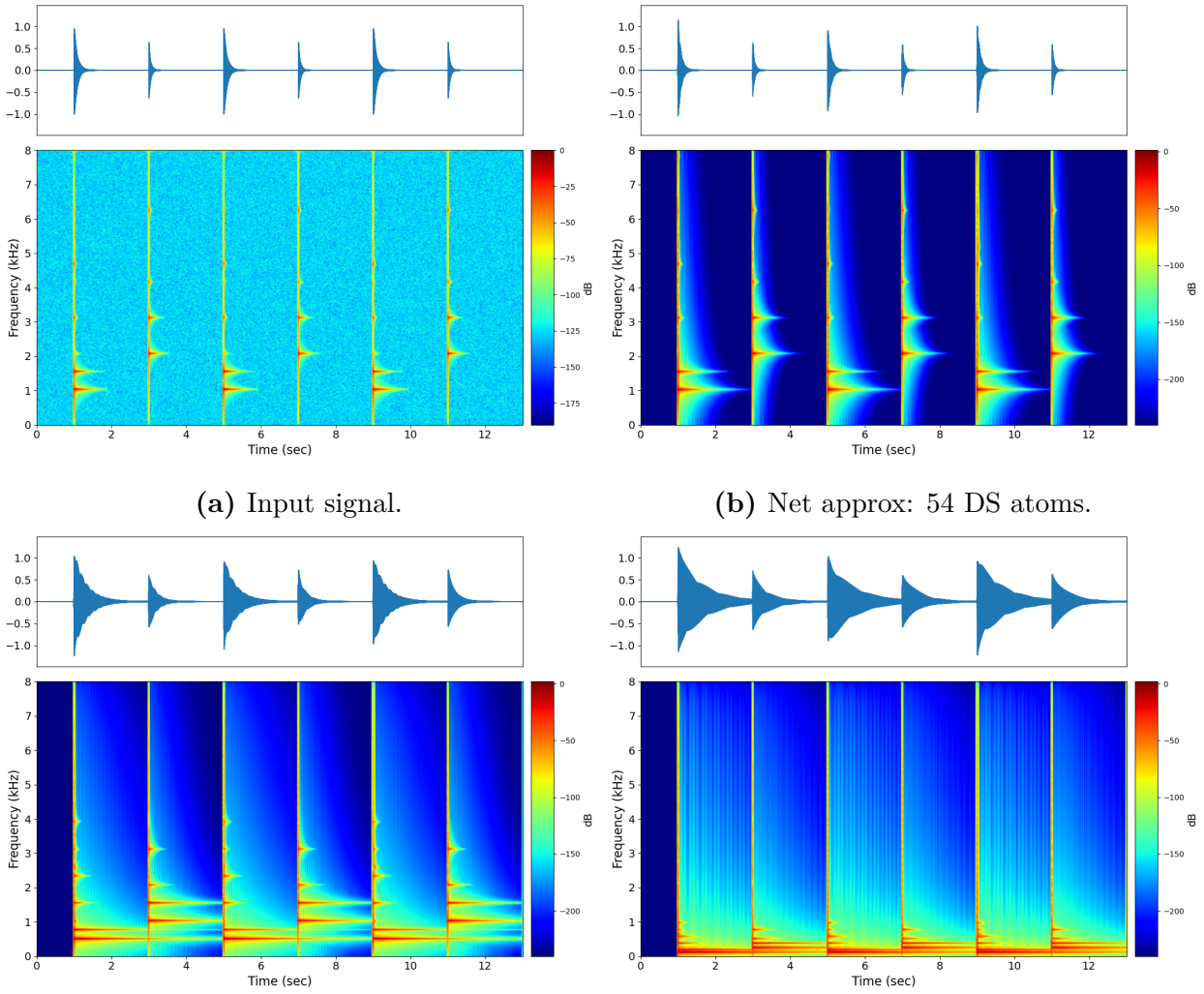
parameters produced by the model. This demonstrates the ability of the network to generate parameters that are meaningful to the sound generating structure of the instrument and allow for coherent manipulation of the audio. In particular, Figure 5.8c shows the signal transformation from shifting the frequency parameter of the atoms down by an octave and multiplying the decay factor α by $\frac{1}{4}$. Multiplying α by $\frac{1}{4}$ modifies the instrument's resonance, effectively making the notes ring for longer. Figure 5.8d shows the signal transformation from shifting the frequency parameter of the atoms down by three octaves and multiplying the decay factor α by $\frac{1}{8}$. The kalimba is a good choice for this experiment since its method of sound production – an impulse-like excitation followed by decaying partials – closely resemble the damped sinewave atom prototype.

Finally, Figure 5.9 shows a decomposition of a long duration kalimba recording from freesound.org[1], which demonstrates that HD-AD is able to scale to longer signals without sacrificing much performance speed. The total time of the method is still above 4 times real-time.

5.1.2 REDS Atoms

For experiments with REDS atoms, we create encoding vectors for time, frequency, damping, and attack parameters. Since a REDS is a modulated DS, all of the same procedures just mentioned for DS apply for REDS. Like with the damping factor, informal listening tests reveal that the perceptual characteristics of the attack parameter β are somewhat exponentially spaced, and thus are mapped with the same encoding map as that in equation (5.3) and are decoded with the decoding map in equation (5.4). For the REDS

[1] "Kalimba" by user "dermotte" (`https://freesound.org/s/244025/`) licensed under CCBYNC 3.0



(a) Input signal.

(b) Net approx: 54 DS atoms.

(c) $\frac{f}{2}$, $\frac{\alpha}{4}$.

(d) $\frac{f}{8}$, $\frac{\alpha}{8}$.

Figure 5.8: Sound processing examples. Signal length: 13.000 s, **(b)** HD-AD time: 3.988 s = 3.260×'s real-time. **(c)** Multiply f by $\frac{1}{2}$ and α by $\frac{1}{4}$. **(d)** Multiply f by $\frac{1}{8}$ and α by $\frac{1}{8}$.

atoms in our dataset, β is restricted such that $\beta \geq \alpha$ in order to maintain an asymmetric envelope shape. We use the same range for β as we did for α in equation (5.8).

$$\beta \in [1.439e^{-4}, 5.397e^{-2}] \ \frac{1}{\text{sample}} \tag{5.9}$$

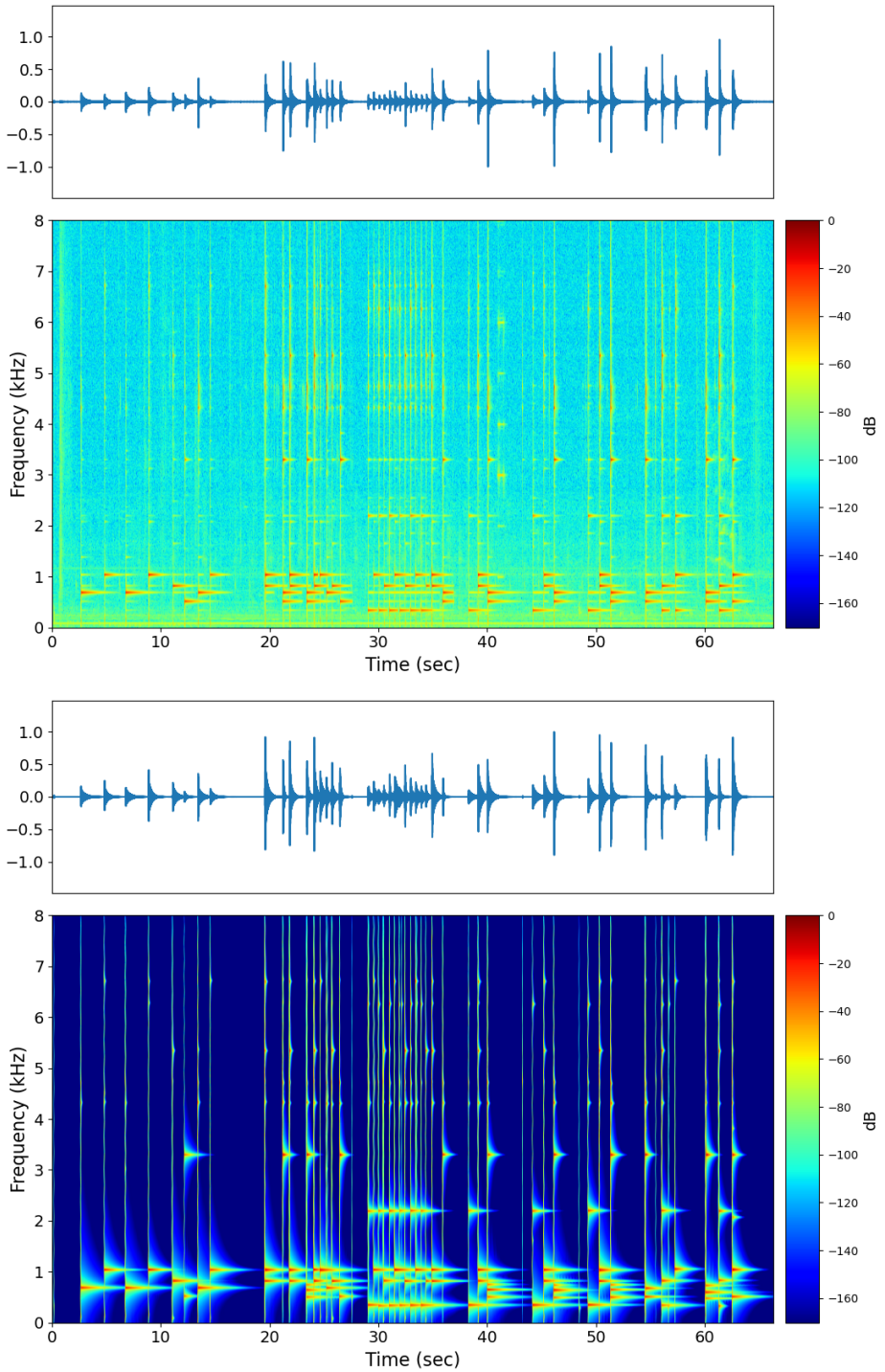


Figure 5.9: Decomposing a long recording with DS atoms. (*top*) Input signal: 66.316 s. (*bottom*) HD-AD time: 16.656 s = 4.236×'s real-time. $\|\mathbf{x}\|_0 = 288$.

which for a sample rate of 16 kHz corresponds to influence times of 3.447 seconds and 9ms, respectively. Since these atoms have 4 parameters, the similarity between the encodings of two REDS atoms $\phi_{\text{REDS}}(\tau_1, f_1, \alpha_1, \beta_1)$ and $\phi_{\text{REDS}}(\tau_2, f_2, \alpha_2, \beta_2)$ is given by the product of 4 sinc functions.

$$\text{sinc}(\dot{\tau}_1 - \dot{\tau}_2) \times \text{sinc}(\dot{f}_1 - \dot{f}_2) \times \text{sinc}(\dot{\alpha}_1 - \dot{\alpha}_2) \times \text{sinc}(\dot{\beta}_1 - \dot{\beta}_2) \tag{5.10}$$

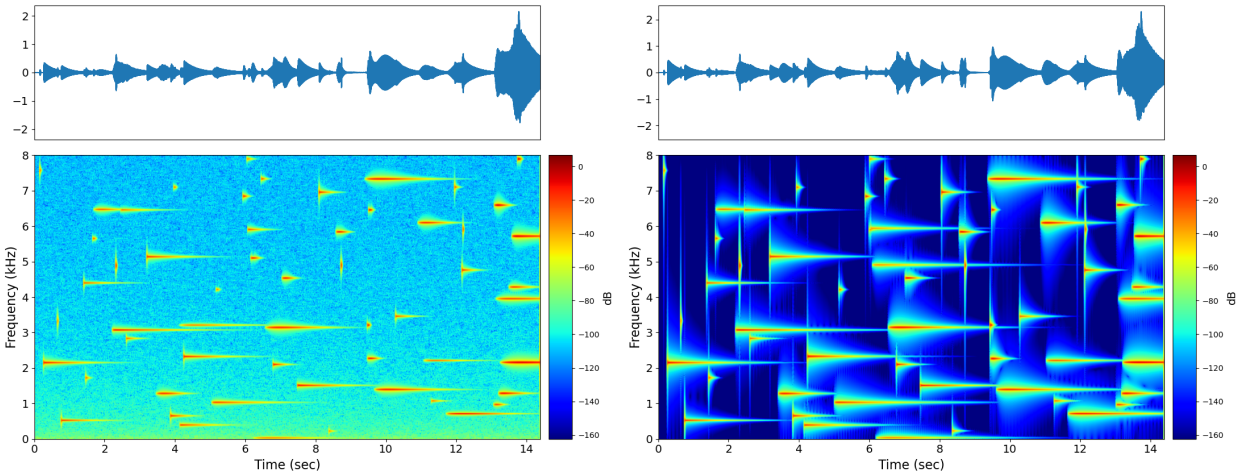
The encoding ranges for our experiments with REDS atoms are

$$\begin{aligned} \max(\dot{\tau}) &= 8 \\ \max(\dot{f}) &= 8 \\ \max(\dot{\alpha}) &= 3 \\ \max(\dot{\beta}) &= 3 \end{aligned} \tag{5.11}$$

The complete experiment settings are given in Table A.2.

Figure 5.10 shows the model decomposition on a mixture of scaled REDS atoms plus noise like the signals it was trained on.

Figure 5.11 shows the system's performance on a 6 second signal of a synthesizer toy piano, and then its performance on the same signal but zero-padded by 10 seconds on either end (total length of 26 seconds). Since the sparsity of the signal has not changed, our method performs each decomposition on the same order of time even though one signal is over four times longer than the other.



(a) Input signal: 56 REDS atoms. **(b)** Net approx: 60 REDS atoms.

Figure 5.10: Decomposing a mixture of REDS atoms. Signal length: 14.384 s, HD-AD time: 8.833 s = 1.628×'s real-time.

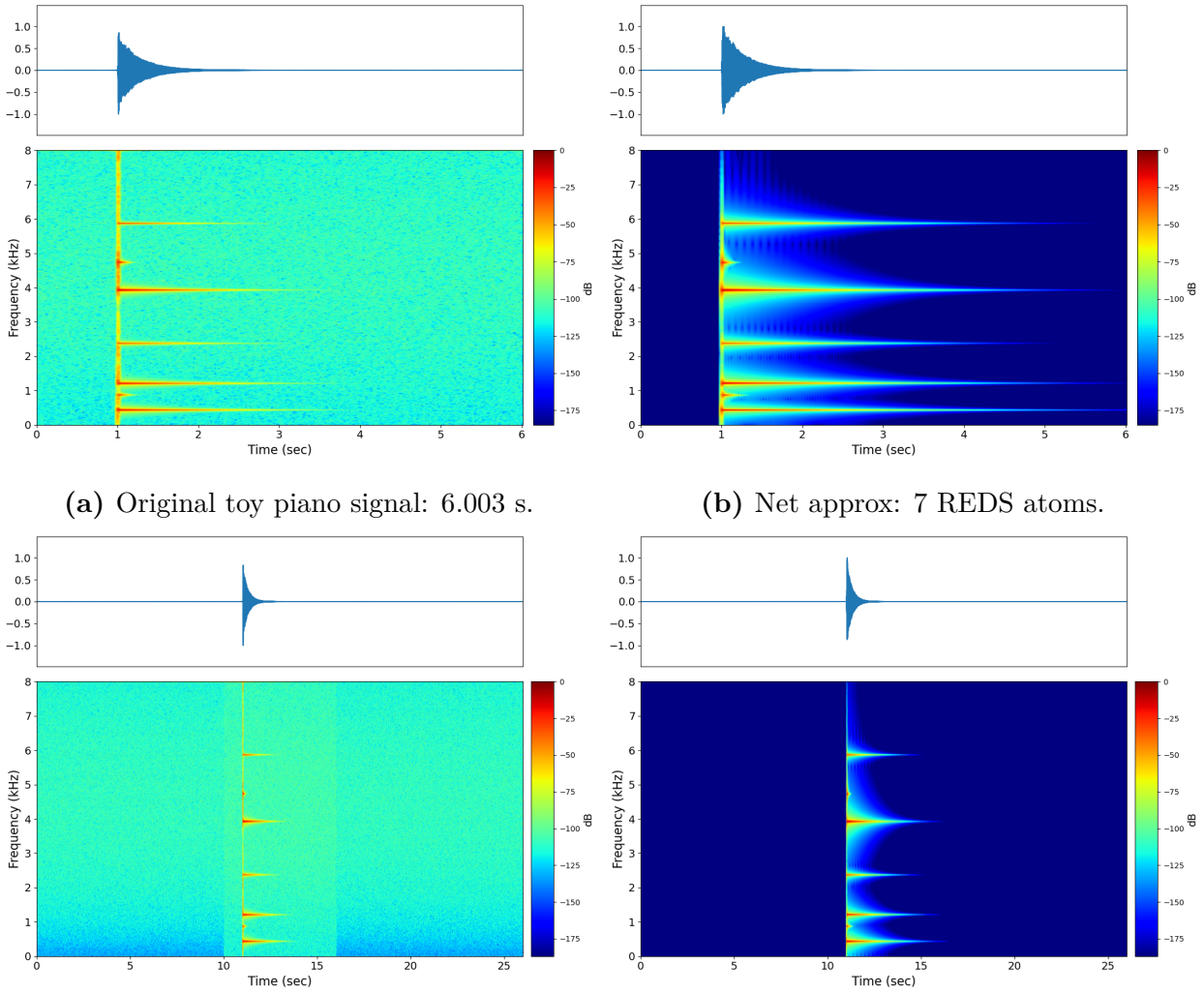
Source-Filter Model

We also train a model to generate decompositions from the source-filter point of view, which we discussed in section 2.3. Because of this, we specify the α and β parameters of a REDS in terms of their bandwidth $B_w = \frac{2\alpha}{\text{sr}}$ and influence time n_I. The range of B_w is

$$[200, 300] \text{ Hz} \tag{5.12}$$

and the range of influence time is

$$[1, 10] \text{ ms} \tag{5.13}$$



(a) Original toy piano signal: 6.003 s.

(b) Net approx: 7 REDS atoms.

(c) Zero-padded toy piano: 26.010 s.

(d) Net approx: 7 REDS atoms.

Figure 5.11: Decomposing **(a)** a signal, and **(c)** a zero-padded version. HD-AD scales with the signal's sparsity, not its length in time. **(b)** HD-AD time: 2.384 s = 2.518×'s real-time. **(d)** HD-AD time: 6.904 s = 3.767×'s real-time.

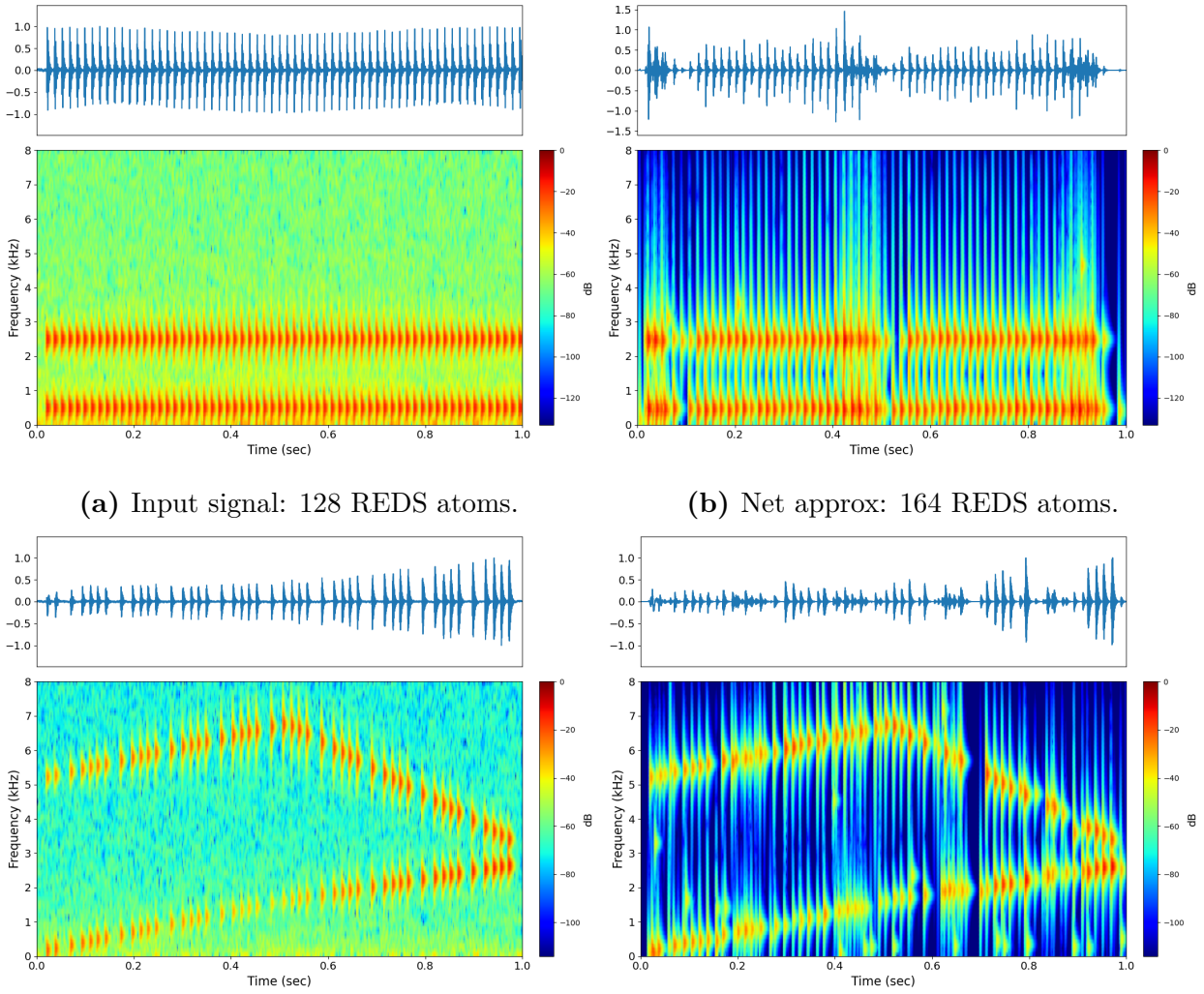
The encoding ranges for our source-filter experiments with REDS atoms are

$$\begin{aligned} \max(\dot{\tau}) &= 4 \\ \max(\dot{f}) &= 4 \\ \max(\dot{\alpha}) &= 1 \\ \max(\dot{\beta}) &= 1 \end{aligned} \tag{5.14}$$

Unlike the previous experiments, which used long time-duration STFT tiles input, the nature of the source-filter model requires much higher time resolution. Furthermore, the density of atoms in time results in a density of encodings, which as we have seen limits the ability to retrieve the parameters via HD Newton's method for a modest N_{HD}. Because of this we feed our network time-frequency tiles which are 8 FFT frames wide and 8 FFT bins tall for $N_{\text{FFT}} = 128$ samples. For a sampling rate of 16kHz, this corresponds to 22 milliseconds in time and 1000 Hz in frequency. The complete experiment settings are given in Table A.3.

Figure 5.12 shows the decomposition of two mixtures of short-duration REDS atoms plus noise like the ones it was trained on. Here a two resonant structure is created in two scenarios – one in which the resonances are stationary, and another where they move in time. In either case, the formant resonances are well localized in frequency. Due to the high density of atoms in time, these decompositions take much longer than real-time to generate. In both examples, the 1-second-long signal takes over 80 seconds to decompose, which corresponds to a real-time multiplier of $\times 0.012$.

Figure 5.13 shows the system decomposition a synthetic voice saying "near you". Much of the formant movement of the voice is captured and once again the time it takes HD-AD to generate a decomposition is on the same order as the previous example – $0.013\times$'s real-time.



(a) Input signal: 128 REDS atoms.

(b) Net approx: 164 REDS atoms.

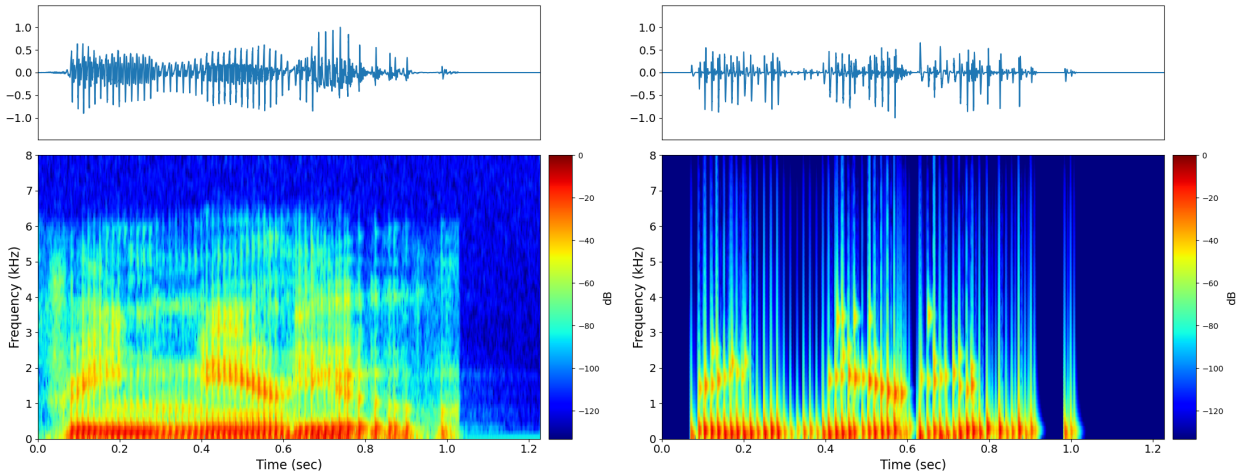
(c) Input signal: 94 REDS atoms.

(d) Net approx: 168 REDS atoms.

Figure 5.12: Decomposing a mixture of source-filter REDS atoms with **(a)** constant resonances and **(c)** moving resonances. **(b)**: HD-AD time: 86.139 s = 0.0012×'s real-time. **(d)**: HD-AD time: 81.827 s = 0.0012×'s real-time.

5.1.3 Vibrato Atoms

One of the advantages of HD-AD is its ability to control the size of the dictionary as the number of parameters in the atom prototype increases. In order to illustrate this, we add



(a) Synthetic voice saying "near you". **(b)** Net approx: 158 REDS atoms.

Figure 5.13: Decomposing a low synthetic voice with source-filter REDS atoms. Signal length: 1.228 s, HD-AD time: 96.456 s = 0.013×'s real-time.

vibrato parameters to the REDS atom prototype, and retrieve them in the same way as the previous experiments. The vibrato parameters are given in the argument to the complex exponential

$$\arg(n, \tau, \omega_c, s, f_m) = (n - \tau)\omega_c + \frac{s}{2\pi f_m} \sin\Big(2\pi f_m (n - \tau)\Big) \tag{5.15}$$

where s is the vibrato amplitude and f_m is the frequency modulation or vibrato rate. The full vibrato REDS atom prototype is

$$\phi(n, \tau, \omega_c, \alpha, \beta, s, f_m) = (1 - e^{-\beta(n-\tau)})^3 e^{-\alpha(n-\tau)} e^{i \cdot \arg(n, \tau, \omega_c, s, f_m)} \tag{5.16}$$

The training dataset is created by sampling s and r, in Hertz, randomly from

$$s \in [0, 250] \quad r \in [1, 8] \tag{5.17}$$

These vibrato REDS (vREDS) atoms generalize the REDS atoms we have been using by letting $s = 0$. α and β are sampled in the same manner as the other REDS experiments. Since these atoms have 6 parameters, the similarity between the encodings of two vibrato REDS atoms $\phi_{\text{vREDS}}(\tau_1, f_1, \alpha_1, \beta_1, s_1, r_1)$ and $\phi_{\text{vREDS}}(\tau_2, f_2, \alpha_2, \beta_2, s_2, r_2)$ is given by the product of 6 sinc functions.

$$\begin{aligned} &\text{sinc}(\dot{\tau}_1 - \dot{\tau}_2) \times \text{sinc}(\dot{f}_1 - \dot{f}_2) \times \text{sinc}(\dot{\alpha}_1 - \dot{\alpha}_2) \times \\ &\text{sinc}(\dot{\beta}_1 - \dot{\beta}_2) \times \text{sinc}(\dot{s}_1 - \dot{s}_2) \times \text{sinc}(\dot{r}_1 - \dot{r}_2) \end{aligned} \tag{5.18}$$

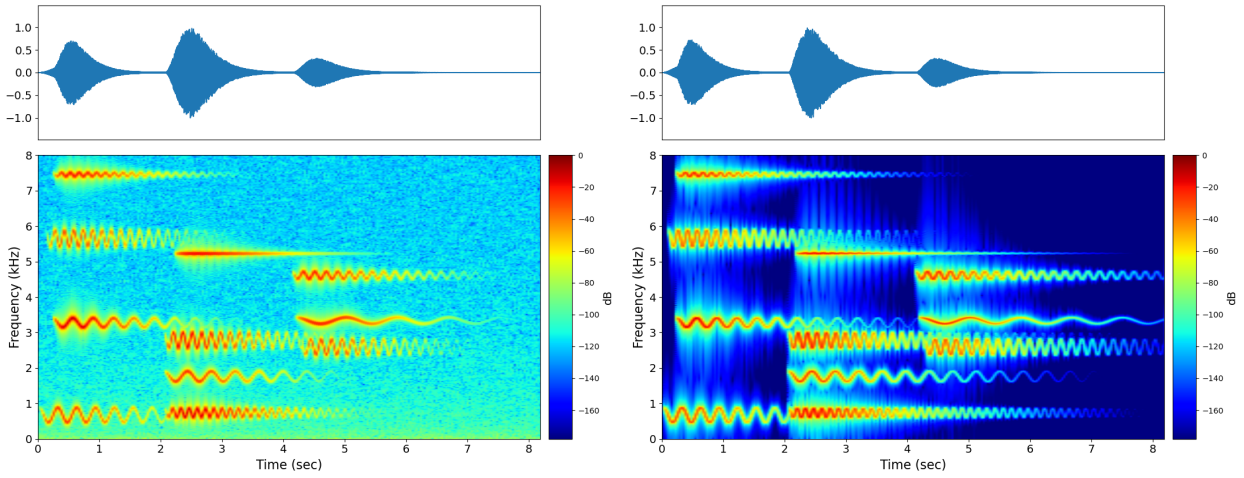
The encoding ranges for vibrato REDS atoms are

$$\begin{aligned} \max(\dot{\tau}) &= 4 \\ \max(\dot{f}) &= 4 \\ \max(\dot{\alpha}) &= 2 \\ \max(\dot{\beta}) &= 2 \\ \max(\dot{s}) &= 2 \\ \max(\dot{r}) &= 2 \end{aligned} \tag{5.19}$$

The STFT tile settings are the same as they were for the DS and REDS experiments. The complete experiment settings are given in Table A.4.

We simplify many aspects of this experiment, for example, there is at most 1 atom born per time-frequency tile, and the parameters are restricted so that atoms don't overlap if they are born in the same time-band. The purpose of this experiment is not to generate a model which can decompose real signals, but rather to demonstrate that creating more complex atom prototypes, which naturally introduce more parameters, does not prohibit

HD-AD from generating a decomposition in a timely manner. This is not the case with MP or many of its derivatives. Figure 5.14 shows the decomposition on a signal which is a mixture of vibrato REDS atoms plus noise like the ones the model was trained on. The 8.192-second-long signal is decomposed in 4.404 seconds, which is 1.860×'s real-time.



(a) Input signal: 11 vibrato REDS atoms. **(b)** Net approx: 11 vibrato REDS atoms.

Figure 5.14: Decomposing a mixture of vibrato REDS atoms. Signal length: 8.192 s, HD-AD time: 4.404 s = 1.860×'s real-time.

We conclude this section with a comparison of the memory footprint of the dictionaries for HD atomic decomposition and a traditional matching pursuit, shown in Table 5.2. As can be seen from the right column, for a given atom prototype, the HD dictionary size does not change as a function of signal length in the time-domain. Finally, the timing of all HD-AD experiments is summarized in Table 5.1 which also divides the total time taken into the neural network time and the decoding time.

experiment (Figure)	**length** (s)	**network** (s)	**decoding** (s)	**total** (s)	**× real-time**	$\|\mathbf{x}\|_0$ (#)
synthetic DS (5.7)	14.384	0.823	1.828	2.651	5.425×	56
kalimba note (5.8)	13.000	0.884	3.104	3.988	3.260×	54
kalimba song (5.9)	66.316	3.662	11.994	15.656	4.236×	288
synthetic REDS (5.10)	14.384	1.179	7.654	8.833	1.628×	60
toy piano (5.11a)	6.003	0.482	1.903	2.384	2.518×	7
toy piano zero-pad (5.11c)	26.010	1.793	5.165	6.904	3.767×	7
synthetic vibrato (5.14)	8.192	0.212	4.192	4.404	1.860×	11
source-filter REDS (5.12a)	1.000	0.352	85.787	86.139	0.012×	164
source-filter REDS (5.12c)	1.000	0.309	81.518	81.827	0.012×	168
synthetic "near you" (5.13)	1.228	0.386	96.071	96.456	0.013×	158

Table 5.1: Timing results for experiments shown in this section.

experiment (Figure)	**atom** (type)	**MP dictionary size** ($M \times N$)	**HD dictionary size** ($M_{\text{HD}} \times N_{\text{HD}}$)
kalimba note (5.8)	DS	$102731 \times 208000 \approx 159$ GB	$2023 \times 1000 \approx 0.016$ GB
kalimba song (5.9)	DS	$521555 \times 1.06\text{M} \approx 4.03$ TB	$2023 \times 1000 \approx 0.016$ GB
toy piano (5.11)	REDS	$332065 \times 96128 \approx 238$ GB	$14161 \times 1000 \approx 0.113$ GB
toy piano zero-pad (5.11)	REDS	$1.16\text{M} \times 416160 \approx 3.52$ TB	$14161 \times 1000 \approx 0.113$ GB
synthetic vibrato (5.14)	vREDS	$3.24\text{M} \times 131072 \approx 3.09$ TB	$50625 \times 1000 \approx 0.405$ GB

Table 5.2: Memory footprint comparison for a dictionary of 64-bit floats for each method, given in gigabytes (GB) or terabytes (TB).

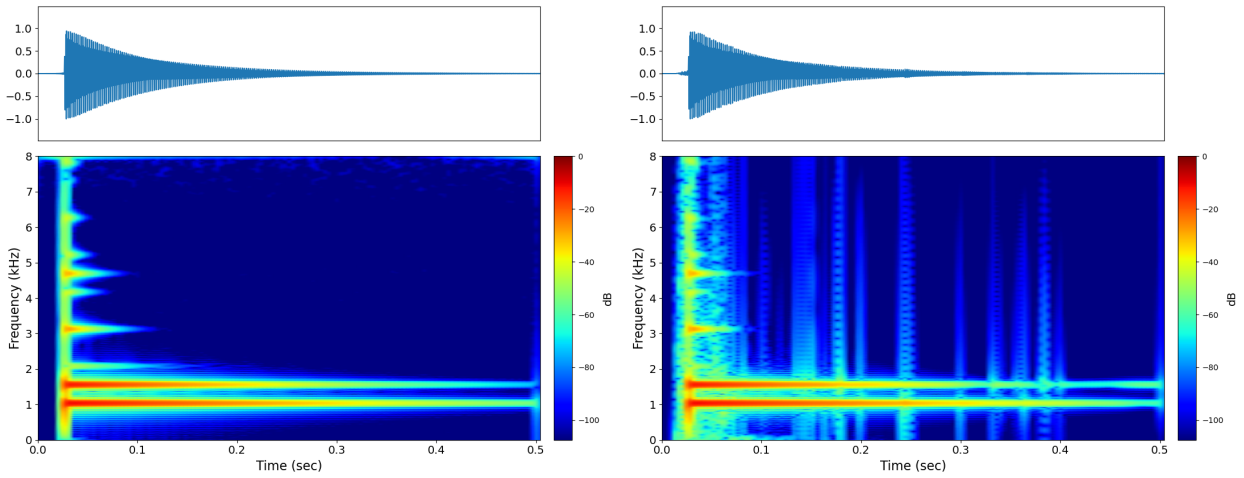
5.2 Neural Network Accelerated MP

In this section we present the results of NN-MP, a system which disregards the coefficient approximation of the neural network and instead builds a dictionary of atoms from the model output to use in a matching pursuit algorithm. The expansion of M for a dictionary of long asymmetric atoms is greatly mitigated by our neural network pre-processing. Furthermore, we show that the same idea can be applied to OMP for what we call NN-OMP. Here we show the time-domain waveform and STFT of the approximations from NN-MP and NN-OMP.

Appendix B has more detailed displays of how the SRR progresses as NN-MP and NN-OMP iterates.

5.2.1 Damped Sinusoid Atoms

We use the parameter estimates from the neural network in section 5.1.1 to populate a dictionary with DS atoms to generate a MP atomic decomposition. In addition, we include atoms whose parameters are around the ones predicted by the model, in order to account for possible inaccuracies from the neural network encoder. Figure 5.15 shows a 30 dB SRR decomposition of one of the kalimba notes from the signal decomposed with our HD atomic decomposition in Figure 5.8.



(a) Input signal. **(b)** NN-MP approximation.

Figure 5.15: NN-MP decomposition of a kalimba note. 30 dB SRR reconstruction with 192 DS atoms.

5.2.2 NN-OMP with Gabor Atoms

Not only does our system serve to matching pursuit more tractable, but it also increases the tractability of MP variants such as OMP, which we discussed in section 2.4.1. While usually generating sparser decompositions, OMP is much slower than traditional MP. In the same way as we did for MP in the previous section, we put a neural network on the front end of an OMP algorithm to pre-select a dictionary of waveforms, which once again mitigates the exponential grown of dictionary size M. We run NN-OMP with symmetric Gabor atoms like those discussed in section 2.1.

For experiments with Gabor atoms, we create encoding vectors for the time, frequency, and window length parameters. For window length, we choose two sizes: one long and one short. These correspond to tonal-like and transient-like structures in sound, respectively.

$$w_{\text{short}} = 32 \text{ samples}, \, w_{\text{long}} = 512 \text{ samples} \tag{5.20}$$

The particular symmetric window used for our experiments is the Hann window E_{Hann} defined in equation (2.5).

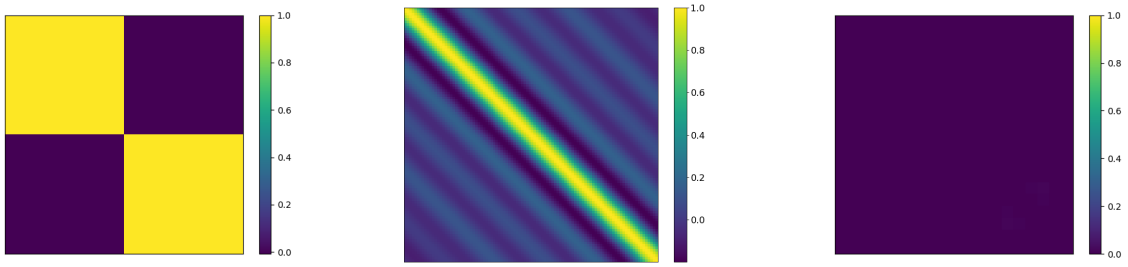
The encoding structure for time and frequency is the same as it has been for all previous experiments.

$$\begin{aligned} \max(\dot{\tau}) &= 4 \\ \max(\dot{f}) &= 8 \end{aligned} \tag{5.21}$$

However, the window length is specified in an integer number of samples, which as we saw in the beginning of this chapter results in two encodings which are almost orthogonal to

each other. Unlike our previous discussion of this phenomena with the time shift τ, where we changed the similarity behavior by choosing a different encoding range, for window length we choose to keep the encodings for tonal-like and transient-like windows quasi-orthogonal to each other.

For window length we seek to preserve the inherent *dis-similarity* between w_{short} and w_{long} when encoding them. In some sense a transient and tonal sound can be thought of as "orthogonal" to each other, where one is a vertical line in frequency, and the other a horizontal line in time. Binding the smooth time-frequency encoding space with either a transient or tonal window encoding results in two HD encoding sub-spaces – one for tonal structures, and one for transient structures. Figure 5.16 shows the effect of binding a smooth encoding space, such as time or frequency, with a quasi-orthogonal one like for window length.



(a) Orthogonal. **(b)** Continuous. **(c)** Bound.

Figure 5.16: **(a)** Similarity between window length encodings. **(b)** Similarity between time and frequency encodings. **(c)** Similarity between long and short window Gabor atoms. $N_{\text{HD}} = 1000$.

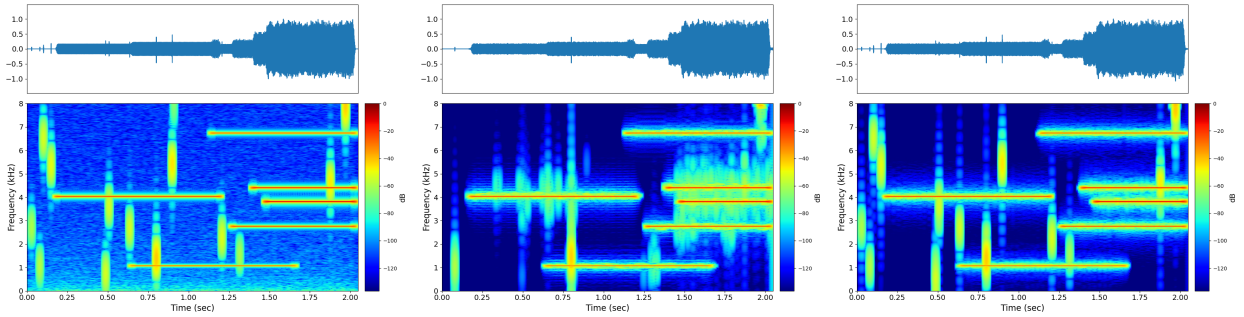
The binding of smooth with orthogonal vectors to divide the HD space into distinct sub-

spaces has been done for multi-channel electromyography (EMG) signals [32] or different quefrency bins in MFCCs [53]. In either case, as with ours, the goal is to preserve some aspect of the symbols to be encoded in the HD space. Since these atoms have 2 continuous parameters, the similarity between the encodings of two Gabor atoms $\phi_{\text{Hann}}(\tau_1, f_1, w_1)$ and $\phi_{\text{Hann}}(\tau_2, f_2, w_2)$ is given by the product of 2 sinc functions.

$$\begin{cases} \text{sinc}(\dot{\tau}_1 - \dot{\tau}_2) \times \text{sinc}(\dot{f}_1 - \dot{f}_2) & w_1 = w_2 \\ \approx 0 & w_1 \neq w_2 \end{cases} \tag{5.22}$$

We train our model on synthetic signals which are the mixture of transient and tonal-like structures, built from Gabor atoms randomly scaled between -30 dB and 0 dB with random phase. The phase of atoms which represent the same tonal structure are set to ensure a smooth sinusoidal oscillation. Similar to the source-filter experiment with REDS atoms in section 5.1.2, the STFT tiles fed to the neural network for Gabor atoms are relatively small: 8 FFT frames wide and 64 FFT bins tall for $N_{\text{FFT}} = 1024$ samples. For a sampling rate of 16kHz, this corresponds to 176 milliseconds in time and 1000 Hz in frequency. When generating data, each tile can have up to two tonal structures within it and up to three transient structures. The model architecture as well as dataset parameters are given in Table A.5.

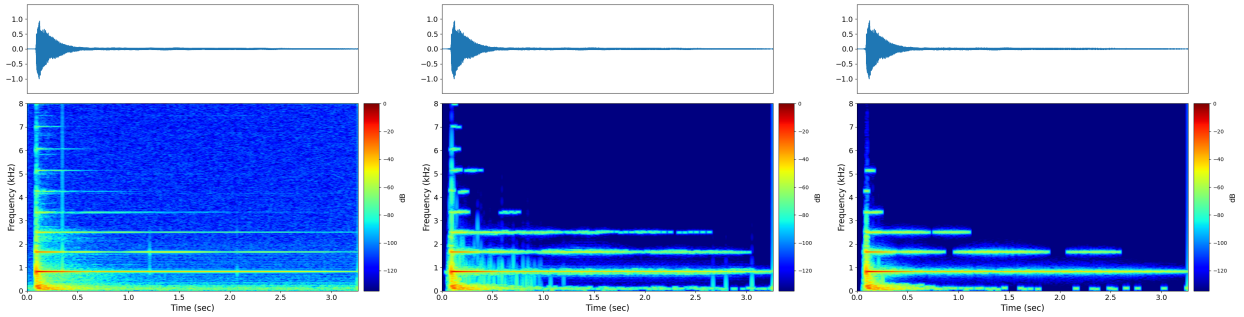
For our NN-OMP experiments we compare the decomposition using the atoms corresponding to points in the HD dictionary $\mathbf{\Phi}_{\text{HD}}$, with the atoms refined by HD Newton's method from these initial estimates. We first test the model on a mixture of synthetic transient and tonal-like sounds plus noise, like the signals it was trained on, shown in Figure 5.17. Then we test the model on a real recording of the piano playing G#5, shown



(a) Input signal. **(b)** Unrefined. Sparsity: 2245. **(c)** Refined. Sparsity: 1275.

Figure 5.17: Mixture of synthetic tonal and transient structures. Unrefined does not reach 30 dB SRR, refined does.

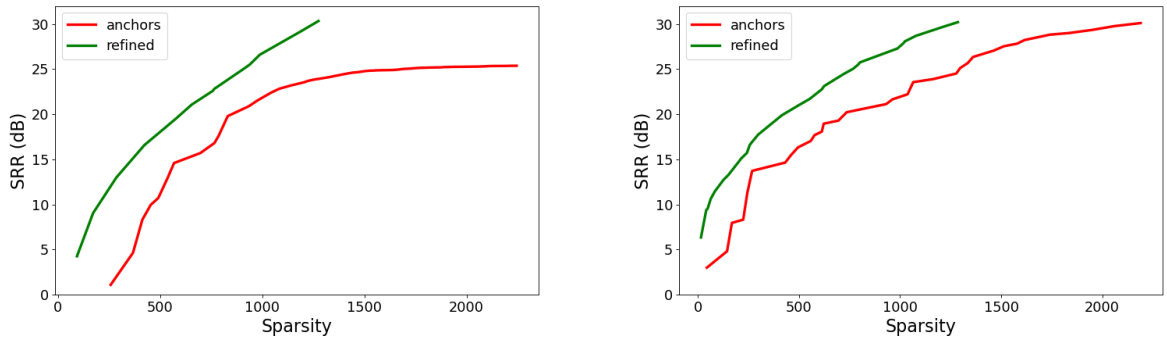
in Figure 5.18.



(a) Input signal. **(b)** Unrefined. Sparsity: 2190. **(c)** Refined. Sparsity: 1286.

Figure 5.18: Piano playing G#5.

In Figure 5.19 we plot the SRR of NN-OMP with and without HD Newton's method used to refine the neural network generated encodings, versus sparsity. HD Newton's method allows the decompositions to achieve a higher SRR in fewer iterations and with greater sparsity.



(a) **(b)**

Figure 5.19: Refined HD encodings vs. raw encodings. (**a**) Synthetic mixture of tonal transient-like structures. (**b**) Piano playing G#5.

In particular, HD Newton's method greatly helps approximate the time parameter of transient Gabor atoms, w_{short}. Because the window is so short, a time error of 16 samples or more will result in an atom which will not be selected by OMP. This illustrates the advantage of distributed representations which we discussed at the end of Chapter 3 and shows that parameter refinement in HD space is preferred in some scenarios depending on the quality of the signal encoder.

We stop the unrefined experiment on the synthetic signal in Figure 5.19 due to the high number of atoms and slow progress of signal approximation. For the synthetic mixture of tonal and transient structures, 30 dB SRR cannot be achieved unless HD Newton's method is used. For the real piano signal, Newton's method results in an increase in sparsity by 904 atoms, or 41.28% fewer atoms, and achieves 30 dB SRR in 5 fewer NN-OMP iterations.

To conclude, in this chapter we presented atomic decompositions generated by HD-AD,

NN-MP, and NN-OMP. The signals being decomposed were the mixture of atom prototypes plus noise, synthesizer instrument sounds, and real musical recordings. The decompositions were generated with DS and REDS asymmetric atoms, as well as symmetric Gabor atoms, from both the additive and source-filter model point of view. We showed that the signal can be modified using the atom parameters in the decomposition, resulting in high-quality signal transformation. Results from our HD-AD system show that it can generate decompositions with asymmetric atoms, from the additive synthesis point of view, faster than real-time.

Chapter 6

Conclusion

In this thesis we addressed the problems that long-duration asymmetric atoms cause to traditional atomic decomposition algorithms. We did this by creating a new atomic decomposition system, Hyperdimensional Atomic Decomposition (HD-AD) which completely avoids time-domain correlations. We showed that HD-AD can quickly approximate parameters which are typically difficult to estimate such as attack and decay. In addition to generating its own atomic decomposition, we showed that HD-AD fit into existing atomic decomposition algorithms like matching pursuit (MP) and orthogonal matching pursuit (OMP) by pre-selecting atoms in the dictionary. We call these systems neural network accelerated matching pursuit (NN-MP) and neural network accelerated orthogonal matching pursuit (NN-OMP). We presented a solution to the VFA decoding problem based on Newton's method, which we referred to as HD Newton's method.

We showed that for the additive model of audio synthesis, HD-AD is able to generate decompositions in much faster than real-time. These speeds are possible for both short and

long duration signals. Furthermore, HD-AD can approximate signals from the source-filter point of view. We demonstrated that our neural network pre-processing make MP and OMP more tractable for asymmetric atoms with NN-MP and NN-OMP.

6.1 Discussion

From the outset of this project, there were some areas of uncertainty which we thought might impede our progress, but that in the end caused us little trouble. Chief of these concerns was the absence of atomic decomposition "ground-truth" for audio signals. However, the ability of the neural network to identify coherent asymmetric atom structures in real-world audio after only ever seeing synthetic mixtures of atom prototypes plus noise we found to be extremely encouraging. We believe this bodes well for future applications of this work that may use other atom prototypes which we did not explore in this thesis.

In addition, we speculate if there could be some unexplored theoretical underpinnings to atomic decomposition of signals in HD space. In particular we wonder if there is a "frame"-like nature to a set of HD encodings corresponding to a time-frequency dictionary that constitutes a frame. Perhaps future explorations into vector function architecture will give hints to this question which would have interesting results from a signal processing point of view.

6.2 Future Work

We now present the future directions we see us, or others, taking this research in. First, we believe that many improvements can be made to the neural network encoder. For our

experiments we found architectures which seemed to work well, and throughout the duration of the project modified it little. However, we find it unlikely that our neural network is the optimal encoder, and that there is still a lot of improvement to be made in this aspect of the research, either with optimizing the network's performance and making it smaller/faster, or boosting the network's performance by making it bigger.

In addition, we believe the sub-sampling of the time-frequency plane can be done in a more intelligent manner in order to ensure atoms are easy to estimate for the model. As we noted in Chapter 5, it can be the case that an atom is dealt a bad hand in terms of its placement in an STFT tile making estimating its parameters difficult. More intelligent STFT sub-sampling would help reduce this issue. One approach we looked at was using Persistent Homology [54], a method for computing topological features which could be used to identify asymmetric atoms in an STFT.

Finally, minimal to no post-processing was done to the HD-AD identified atoms. It is possible that the performance could be improved by, for example, adjusting an atom's time shift τ based on onset detection from another system. In the case of the source-filter model, we found that often many atoms can be identified in a short time frame which likely are being inferred from only one atom-like structure in the input signal. Post-processing these detections to select the best-fitting of those identified could help reduce this problem, as well as potentially allow for over-sampling of the time-frequency plane for processing with HD-AD.

Bibliography

[1] J. Neri, "Sparse Representations of Audio Signals with Asymmetric Atoms," Master's thesis, McGill University, 2018.

[2] P. Kanerva, "Hyperdimensional Computing: An Introduction to Computing in Distributed Representation with High-Dimensional Random Vectors," *Cognitive Computation*, vol. 1, no. 2, p. 139–159, 2009.

[3] M. Plumbley, T. Blumensath, L. Daudet, R. Gribonval, and M. Davies, "Sparse Representations in Audio and Music: From Coding to Source Separation," *Proceedings of the IEEE*, vol. 98, no. 6, pp. 995–1005, 2010.

[4] B. Sturm, C. Roads, A. Mcleran, and J. Shynk, "Analysis, Visualization, and Transformation of Audio Signals Using Dictionary-based Methods," *Journal of New Music Research*, vol. 38, no. 4, pp. 325–341, 2009.

[5] S. Mallat and Z. Zhang, "Matching Pursuit with Time-Frequency Dictionaries," *Signal Processing, IEEE Transactions on*, vol. 41, no. 12, pp. 3397–3415, 1993.

[6] E. Frady, D. Kleyko, C. Kymn, B. Olshausen, and F. Sommer, "Computing on Functions Using Randomized Vector Representations," 2021. `https://arxiv.org/abs/2109.03429`.

[7] Y. Pati, R. Rezaiifar, and P. Krishnaprasad, "Orthogonal Matching Pursuit: Recursive Function Approximation with Applications to Wavelet Decomposition," in *Proceedings of 27th Asilomar Conference on Signals, Systems and Computers*, (Pacific Grove, CA), pp. 40–44 vol.1, 1-3 Nov. 1993.

[8] P. Flandrin, *Time-Frequency/Time-Scale Analysis, Volume 10*. USA: Academic Press, Inc., 1st ed., 1998.

[9] M. Jafari and M. Plumbley, "Fast Dictionary Learning for Sparse Representations of Speech Signals," *IEEE Journal of Selected Topics in Signal Processing*, vol. 5, no. 5, pp. 1025–1031, 2011.

[10] D. Gabor, "Theory of Communication," *Journal of the Institution of Electrical Engineers*, vol. 93, no. 3, p. 429–457, 1946.

[11] J. Allen and L. Rabiner, "A Unified Approach to Short-Time Fourier Analysis and Synthesis," *Proceedings of the IEEE*, vol. 65, no. 11, pp. 1558–1564, 1977.

[12] R. Blackman and J. Tukey, "The Measurement of Power Spectra from the Point of View of Communications Engineering — Part I," *The Bell System Technical Journal*, vol. 37, no. 1, pp. 185–282, 1958.

[13] P. Depalle, G. García, and X. Rodet, "Tracking of Partials for Additive Sound Synthesis Using Hidden Markov Models," in *Proceedings of the 1993 IEEE International Conference on Acoustics, Speech, and Signal Processing*, vol. 1 of *ICASSP'93*, (Minneapolis, MN), pp. 225–228, 27-30 April 1993.

[14] G. M. R. de Prony, "Essai Expérimental et Analytique sur les Lois de la Dilatabilité des Fluides Élastiques," *Journal de l'École Polytechnique*, vol. 1, no. 22, pp. 24–76, 1795.

[15] M. Mathews and J. Miller, *The Technology of Computer Music*. The MIT Press, 1969.

[16] J. L. Flanagan, "Models for Approximating Basilar Membrane Displacement," *The Bell System Technical Journal*, vol. 39, no. 5, pp. 1163–1191, 1960.

[17] X. Rodet, Y. Potard, and J.-B. Barrière, "The CHANT Project: From the Synthesis of the Singing Voice to Synthesis in General," *Computer Music Journal*, vol. 8, no. 3, pp. 15–31, 1984.

[18] M. Sadeghi, M. Babaie-Zadeh, and C. Jutten, "Learning Overcomplete Dictionaries Based on Atom-by-Atom Updating," *IEEE Transactions on Signal Processing*, vol. 62, no. 4, pp. 883–891, 2014.

[19] B. Mailhé, R. Gribonval, P. Vandergheynst, and F. Bimbot, "Fast Orthogonal Sparse Approximation Algorithms Over Local Dictionaries," *Signal Processing*, vol. 91, no. 12, pp. 2822–2835, 2011.

[20] H. Zhu, W. Chen, and Y. Wu, "Efficient Implementations for Orthogonal Matching Pursuit," *Electronics*, vol. 9, no. 9, 2020.

[21] M. Imani, D. Kong, A. Rahimi, and T. Rosing, "VoiceHD: Hyperdimensional Computing for Efficient Speech Recognition," in *2017 IEEE International Conference on Rebooting Computing (ICRC)*, (Washington, DC, USA), pp. 1–8, 8-9 Nov. 2017.

[22] A. Rahimi, P. Kanerva, and J. Rabaey, "A Robust and Energy-Efficient Classifier Using Brain-Inspired Hyperdimensional Computing," in *Proceedings of the 2016 International Symposium on Low Power Electronics and Design*, ISLPED'16, (San Fransisco Airport, CA, USA), pp. 64–69, 08-10 August 2016.

[23] B. Komer, T. Stewart, A. Voelker, and C. Eliasmith, "A Neural Representation of Continuous Space Using Fractional Binding," (Montréal, Canada), 2019.

[24] P. Neubert, S. Schubert, and P. Protzel, "Learning Vector Symbolic Architectures for Reactive Robot Behaviours," in *Workshop on Machine Learning Methods for High-Level Cognitive Capabilities in Robotics in conjunction with IROS*, (Daejeon, Korea), August 2017.

[25] M. Imani, *Machine Learning in IoT Systems: From Deep Learning to Hyperdimensional Computing*. PhD thesis, UC San Diego, 2020.

[26] R. Gayler, "Vector Symbolic Architectures Answer Jackendoff's Challenges for Cognitive Neuroscience," in *ICCS/ASCS International Conference on Cognitive Science*, (Sydney, Australia), 13-17 July 2003.

[27] P. Smolensky, "Tensor Product Variable Binding and the Representation of Symbolic Structures in Connectionist Systems," *Artificial Intelligence*, vol. 46, no. 1, pp. 159–216, 1990.

[28] T. Plate, "Holographic Reduced Representations," *IEEE Transactions on Neural Networks*, vol. 6, no. 3, pp. 623–41, May 1995.

[29] P. Frady, S. Kent, B. Olshausen, and F. Sommer, "Resonator Networks, 1: An Efficient Solution for Factoring High-Dimensional, Distributed Representations of Data Structures," *Neural Computation*, vol. 32, no. 12, pp. 2311–2331, 2020. PMID: 33080162.

[30] T. Plate, "Holographic Reduced Representations: Convolution Algebra for Compositional Distributed Representations," in *Proceedings of the 12th International Joint Conference on Artificial Intelligence (IJCAI)*, (San Mateo, CA), pp. 30–35, August 1991.

[31] P. Kanerva, "Binary Spatter-Coding of Ordered K-Tuples," in *Artificial Neural Networks - ICANN 96*, (Bochum, Germany), pp. 869–873, July 1996.

[32] A. Rahimi, S. Benatti, P. Kanerva, L. Benini, and J. Rabaey, "Hyperdimensional Biosignal Processing: A Case Study for EMG-based Hand Gesture Recognition," in *2016 IEEE International Conference on Rebooting Computing (ICRC)*, (San Diego, CA), pp. 1–8, October 2016.

[33] C. Eliasmith, T. Stewart, X. Choo, T. Bekolay, T. Dewolf, C. Tang, and D. Rasmussen, "A Large-Scale Model of the Functioning Brain," *Science (New York, NY)*, vol. 338, pp. 1202–5, November 2012.

[34] G. Karunaratne, M. Schmuck, M. Gallo, G. Cherubini, L. Benini, A. Webastian, and A. Rahimi, "Robust High-Dimensional Memory-Augmented Neural Networks," *Nature Communications*, vol. 12, 2021.

[35] H. Purwins, B. Li, T. Virtanen, J. Schlüter, S. Chang, and S. Sainath, "Deep Learning for Audio Signal Processing," *IEEE Journal of Selected Topics in Signal Processing*, vol. 13, no. 2, pp. 206–219, 2019.

[36] Y. Lecun, "Generalization and Network Design Strategies," in *Connectionism in perspective* (R. Pfeifer, Z. Schreter, F. Fogelman, and L. Steels, eds.), Elsevier, 1989.

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

[38] A. Krizhevsky, I. Sutskever, and G. Hinton, "Imagenet Classification with Deep Convolutional Neural Networks," *Communications of the ACM*, vol. 60, pp. 84–90, 2012.

[39] V. Dumoulin and F. Visin, "A Guide to Convolution Arithmetic for Deep Learning," 2016. `https://arxiv.org/abs/1603.07285`.

[40] X. Glorot, A. Bordes, and Y. Bengio, "Deep Sparse Rectifier Neural Networks," in *14th International Conference on Artificial Intelligence and Statistics*, vol. 15, (Fort Lauderdale, FL, USA), pp. 315–323, April 2011.

[41] D. Clevert, T. Unterthiner, and S. Hochreiter, "Fast and Accurate Deep Network Learning by Exponential Linear Units (elus)," in *4th International Conference on Learning Representations*, ICLR 2016, (San Juan, Puerto Rico), 02-04 May 2016.

[42] L. Bottou, "Large-Scale Machine Learning with Stochastic Gradient Descent," *Lechevallier Y., Saporta G. (eds) Proceedings of COMPSTAT*, 2010. doi: https://doi.org/10.1007/978-3-7908-2604-3_16.

[43] D. Rumelhart, G. Hinton, and R. Williams, "Learning Representations by Back-Propagating Errors," *Nature*, vol. 323, pp. 533–536, 1986.

[44] M. Ravanelli and Y. Bengio, "Speaker Recognition from Raw Waveform with SincNet," *2018 IEEE Spoken Language Technology Workshop (SLT)*, pp. 1021–1028, 2018.

[45] Z. Zhang, Y. Wang, C. Gan, J. Wu, J. Tenenbaum, A. Torralba, and W. Freeman, "Deep Audio Priors Emerge From Harmonic Convolutional Networks," in *International Conference on Learning Representations*, (Addis Ababa, Ethiopia (Virtual)), 2020.

[46] K. He, X. Zhang, S. Ren, and J. Sun, "Deep Residual Learning for Image Recognition," in *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, (Las Vegas, NV, USA), pp. 770–778, June 2016.

[47] M. Ranzato, F. Huang, Y. Boureau, and Y. LeCun, "Unsupervised Learning of Invariant Feature Hierarchies with Applications to Object Recognition," in *Proceedings of the 2007 IEEE Conference on Computer Vision and Pattern Recognition*, (Minneapolis, MN), pp. 1–8, 06 2007.

[48] A. van den Oord, S. Dieleman, H. Zen, K. Simonyan, O. Vinyals, A. Graves, N. Kalchbrenner, A. Senior, and K. Kavukcuoglu, "Wavenet: A Generative Model for Raw Audio," (Sunnyvale, CA, USA), September 2016. `https://arxiv.org/abs/1609.03499`.

[49] K. Tan and D. Wang, "Learning Complex Spectral Mapping with Gated Convolutional Recurrent Networks for Monaural Speech Enhancement," *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, vol. 28, pp. 380–390, 2020.

[50] W. Ma, Y. Hu, and H. Huang, "Dual Attention Network for Pitch Estimation of Monophonic Music," *Symmetry*, vol. 13, p. 1296, July 2021. doi: 10.3390/sym13071296.

[51] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala, "PyTorch: An Imperative Style, High-Performance Deep Learning Library," in *Advances in Neural Information Processing Systems 32* (H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, eds.), pp. 8024–8035, Curran Associates, Inc., 2019.

[52] D. Toledano, M. Fernández-Gallego, and A. Lozano-Diez, "Multi-Resolution Speech Analysis for Automatic Speech Recognition Using Deep Neural Networks: Experiments on TIMIT," *PLoS ONE*, vol. 13, 2018. doi: https://doi.org/10.1371/journal.pone.0205355.

[53] L. Ge and K. Parhi, "Classification using Hyperdimensional Computing: A Review," *IEEE Circuits and Systems Magazine*, vol. 20, pp. 30–47, Second quarter 2020. doi: 10.1109/MCAS.2020.2988388.

[54] H. Edelsbrunner, D. Letscher, and A. Zomorodian, "Topological Persistence and Simplification," in *Proceedings of the 41st Annual Symposium on Foundations of Computer Science*, (Redondo Beach, CA, USA), pp. 454–463, 2000.

Appendices

Appendix A

Experiment Settings

Here we detail the exact settings for each experiment from Chapter 5. The neural network architectures and parameter encoding ranges were chosen largely through trial and error. The synthesis parameter ranges were chosen to represent a wide range of sounds common in audio.

Model		**Data**	
sub-layers	0	**sampling rate**	16000
sub-layer filters	–	**FFT size**	1024
activation function	ELU	**hop size**	256
filters	ConvGLU	**tile** Δt	128 STFT frames = 2.096s
$\mathbf{ds}_k$	$[2,2]^6 + [1,2]$	**tile** Δf	64 bins = 1000 Hz
$\mathbf{channels}_k$	2^i for $i = 5, ..., 11$	α **range**	[8ms, 3s]
FC channels	2048	$\max(\dot{\tau})$	8
N_{HD}	1000	$\max(\dot{f})$	8
		$\max(\dot{\alpha})$	3

Table A.1: Damped Sinewave Experiment Overview. $[2,2]^6 + [1,2]$ denotes down-sampling by $[2,2]$ for the first 6 layers and then by $[1,2]$ in the following layer. α range shown as the atom's length in seconds $= \frac{-1}{\alpha}\log(0.001)$.

Model		Data	
sub-layers	0	**sampling rate**	16000
sub-layer filters	–	**FFT size**	1024
activation function	ELU	**hop size**	256
filters	ConvGLU	**tile** Δt	128 STFT frames = 2.096s
$\mathbf{ds}_k$	$[2,2]^6 + [1,2]$	**tile** Δf	64 bins = 1000 Hz
$\mathbf{channels}_k$	2^i for $i = 5, ..., 11$	α **range**	[8ms, 3s]
FC channels	2048	β **range**	[9ms, 3.477s]
N_{HD}	1000	$\max(\dot{\tau})$	8
		$\max(\dot{f})$	8
		$\max(\dot{\alpha})$	3
		$\max(\dot{\beta})$	3

Table A.2: REDS Experiment Overview. β range shown as the atom's attack envelope influence time $n_I = \frac{-1}{\beta}\log\left(1-(1-\delta)^{\frac{1}{p}}\right)$ for $\delta = 0.001$ and $p = 3$.

Model		Data	
sub-layers	1	**sampling rate**	16000
sub-layer filters	3	**FFT size**	128
activation function	ELU	**hop size**	32
filters	ConvGLU	**tile** Δt	8 STFT frames = 22ms
$\mathbf{ds}_k$	$[2,2]^3$	**tile** Δf	8 bins = 1000 Hz
$\mathbf{channels}_k$	2^i for $i = 5, 6, 7$	B_w **range**	[200, 300] Hz
FC channels	2048	n_I **range**	[1, 10] ms
N_{HD}	1000	$\max(\dot{\tau})$	4
		$\max(\dot{f})$	4
		$\max(\dot{\alpha})$	1
		$\max(\dot{\beta})$	1

Table A.3: REDS source-filter experiment overview. $\alpha = \frac{2B_w}{\text{sr}}$.

Model		Data	
sub-layers	0	**sampling rate**	16000
sub-layer filters	–	**FFT size**	1024
activation function	ELU	**hop size**	256
filters	ConvGLU	**tile** Δt	128 STFT frames = 2.096s
$\mathbf{ds}_k$	$[2,2]^6 + [1,2]$	**tile** Δf	64 bins = 1000 Hz
$\mathbf{channels}_k$	2^i for $i = 5, ..., 11$	α **range**	[8ms, 3s]
FC channels	2048	β **range**	[9ms, 3.477s]
N_{HD}	1000	s **range**	$[0, 250]$ Hz
		r **range**	$[1, 8]$ Hz
		$\max(\dot{\tau})$	4
		$\max(\dot{f})$	4
		$\max(\dot{\alpha})$	2
		$\max(\dot{\beta})$	2
		$\max(\dot{s})$	2
		$\max(\dot{r})$	2

Table A.4: Vibrato REDS experiment overview. s range and r range shown in Hertz.

Model		Data	
sub-layers	0	**sampling rate**	16000
sub-layer filters	–	**FFT size**	1024
activation function	ELU	**hop size**	256
filters	ConvGLU	**tile** Δt	8 STFT frames = 176ms
$\mathbf{ds}_k$	$[2,2]^3 + [2,1]^3$	**tile** Δf	64 bins = 1000 Hz
$\mathbf{channels}_k$	2^i for $i = 5, ..., 10$	**window sizes**	2ms, 32ms
FC channels	2048	$\max(\dot{\tau})$	4
N_{HD}	1000	$\max(\dot{f})$	8

Table A.5: Gabor experiment overview.

Appendix B

NN-MP/OMP Details

Here we show detailed figures of NN-MP and NN-OMP which plots the residual as well as the SRR evolution as the system iterates on the residual.

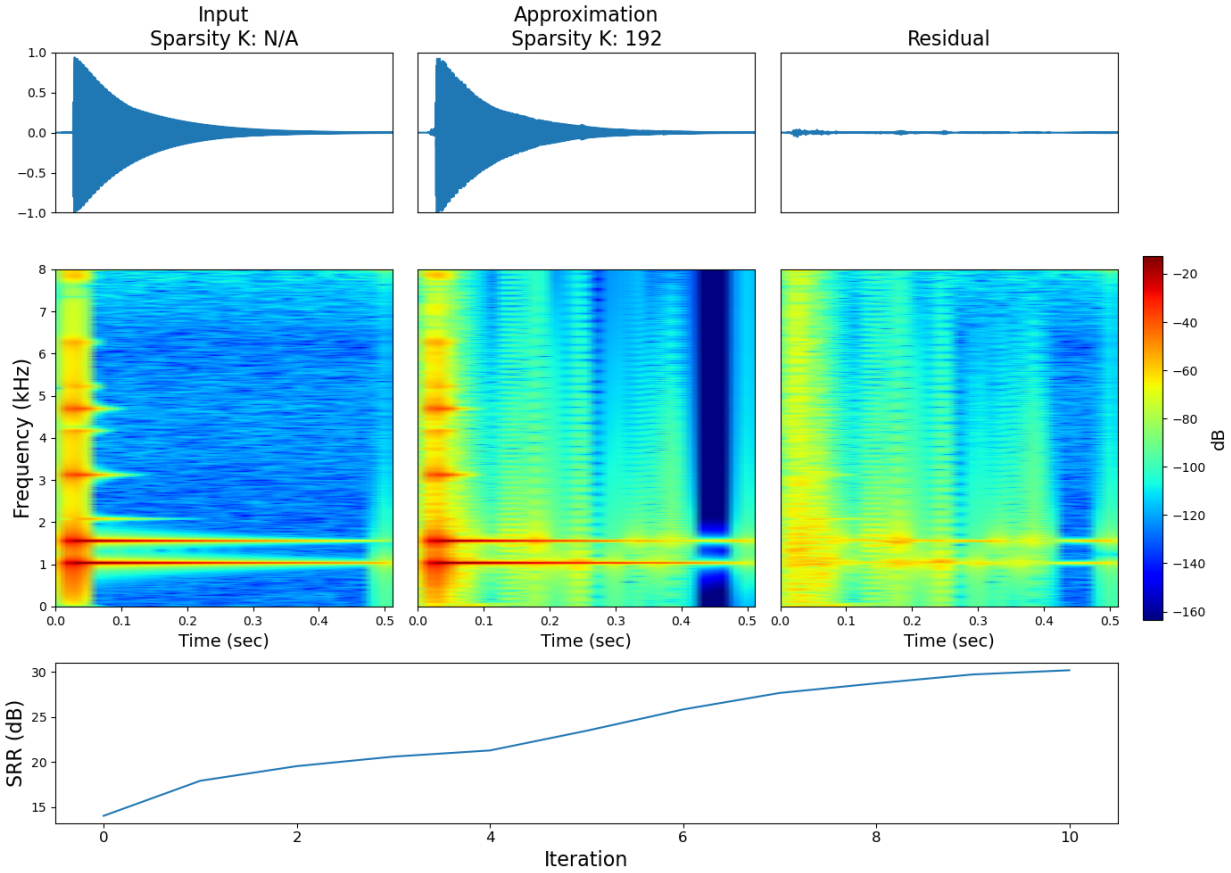


Figure B.1: Decomposing a kalimba C5 note with DS atoms using NN-MP.

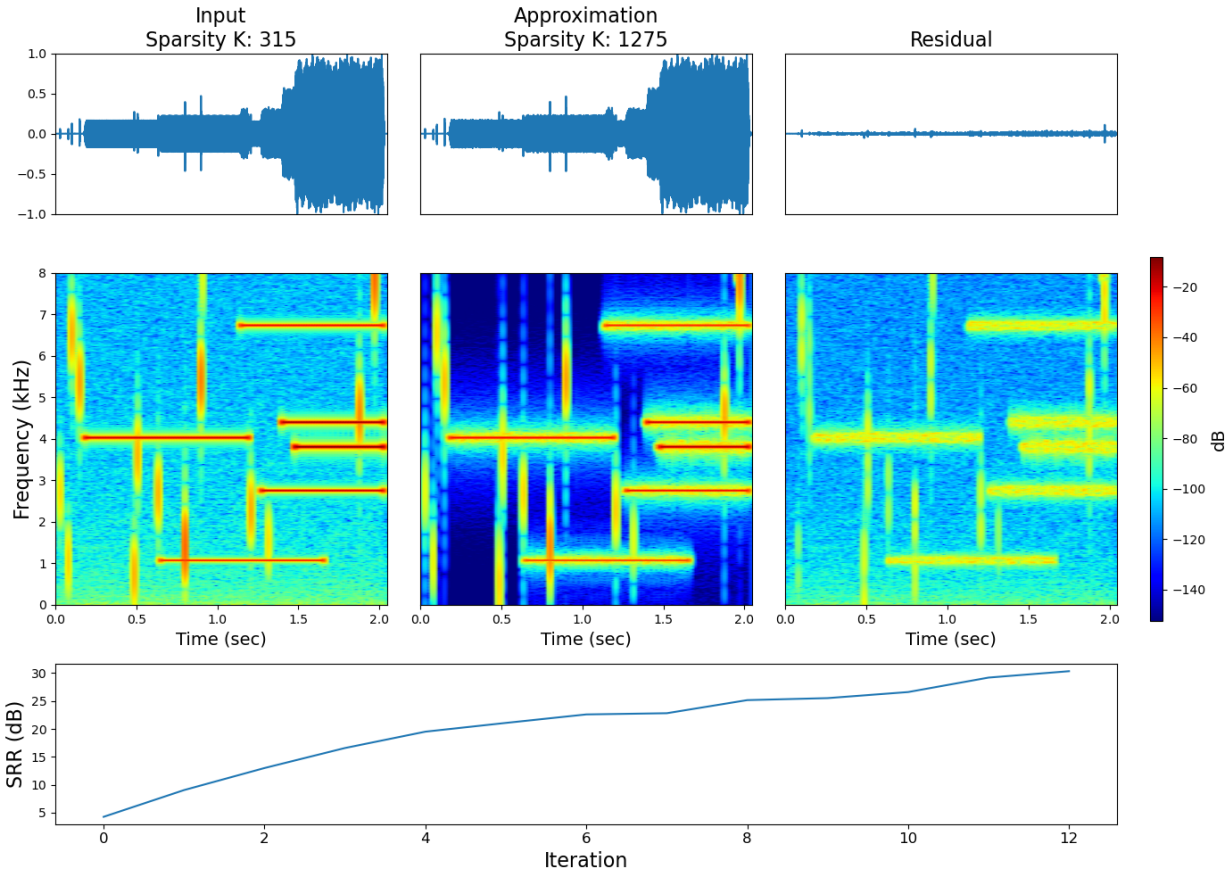


Figure B.2: Decomposing synthetic transient and tonal like structures with Gabor atoms using NN-OMP.

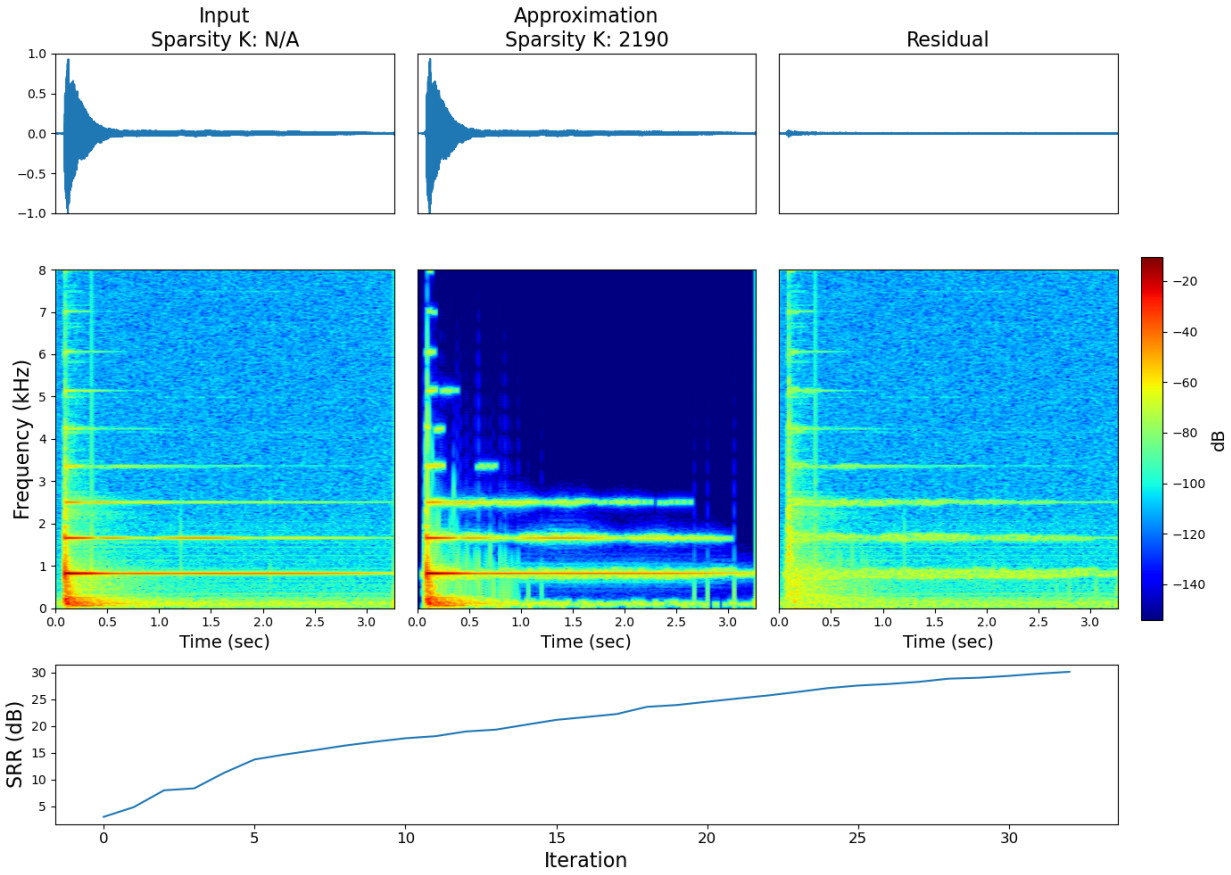


Figure B.3: Decomposing a piano signal playing G#5 with Gabor atoms without parameter refinement using NN-OMP.

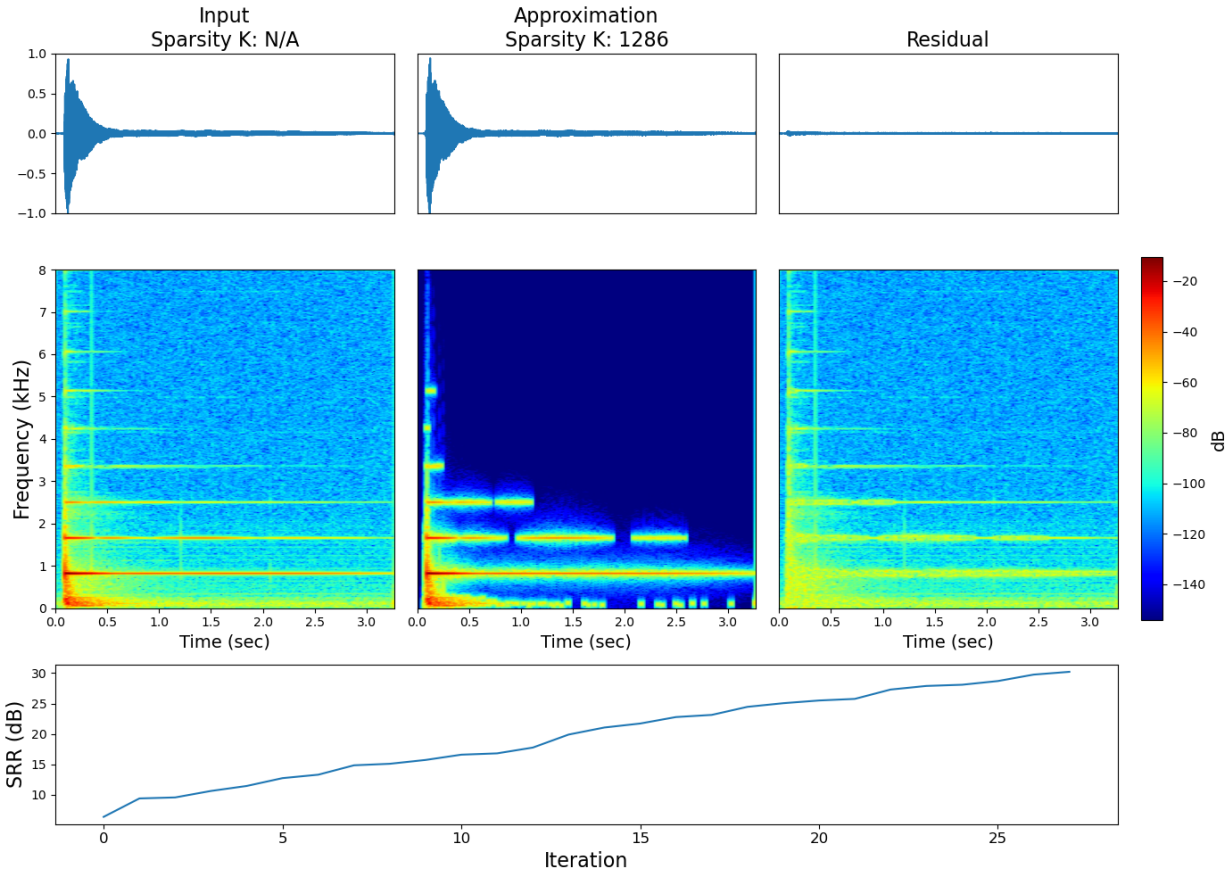


Figure B.4: Decomposing a piano signal playing G#5 with Gabor atoms with parameter refinement using HD Newton's method with NN-OMP.