Learning-Based Methods for Comparing Sequences, with Applications to Audio-to-MIDI Alignment and Matching

Colin Raffel

Submitted in partial fulfillment of the requirements for
the degree of Doctor of Philosophy
in the Graduate School of Arts and Sciences

COLUMBIA UNIVERSITY

2016

© 2016
Colin Raffel
Some Rights Reserved

This work is licensed under the Creative Commons Attribution 4.0 License. To view a copy of this license, visit https://creativecommons.org/licenses/by/4.0/ or send a letter to Creative Commons, 171 Second Street, Suite 300, San Francisco, California, 94105, USA.

Abstract

Learning-Based Methods for Comparing Sequences, with Applications to Audio-to-MIDI Alignment and Matching

Colin Raffel

Sequences of feature vectors are a natural way of representing temporal data. Given a database of sequences, a fundamental task is to find the database entry which is the most similar to a query. In this thesis, we present learning-based methods for efficiently and accurately comparing sequences in order to facilitate large-scale sequence search. Throughout, we will focus on the problem of matching MIDI files (a digital score format) to a large collection of audio recordings of music. The combination of our proposed approaches enables us to create the largest corpus of paired MIDI files and audio recordings ever assembled.

Dynamic time warping (DTW) has proven to be an extremely effective method for both aligning and matching sequences. However, its performance is heavily affected by factors such as the feature representation used and its adjustable parameters. We therefore investigate automatically optimizing DTW-based alignment and matching of MIDI and audio data. Our approach uses Bayesian optimization to tune system design and parameters over a synthetically-created dataset of audio and MIDI pairs. We then perform an exhaustive search over DTW score normalization techniques to find the optimal method for reporting a reliable alignment confidence score, as required in matching tasks. This results in a DTW-based system which is conceptually simple and highly accurate at both alignment and matching. We also verify that this system achieves high performance in a large-scale qualitative evaluation of real-world alignments.

Unfortunately, DTW can be far too inefficient for large-scale search when sequences are very long and consist of high-dimensional feature vectors. We therefore propose a method for mapping sequences of continuously-valued feature vectors to downsampled sequences of binary vectors. Our approach involves training a pair of convolutional networks to map paired groups of subsequent feature vectors to a Hamming space where similarity is preserved. Evaluated on the task of matching MIDI files to a large database of audio recordings, we show that this technique enables 99.99% of the database to be discarded with a modest false reject rate while only requiring 0.2% of the time to compute.

Even when sped-up with a more efficient representation, the quadratic complexity of DTW greatly hinders its feasibility for very large-scale search. This cost can be avoided by mapping entire sequences to fixed-length vectors in an embedded space where sequence similarity is approximated by Euclidean distance. To achieve this embedding, we propose a feed-forward attention-based neural network model which can integrate arbitrarily long sequences. We show that this approach can extremely efficiently prune 90% of our audio recording database with high confidence.

After developing these approaches, we applied them together to the practical task of matching 178,561 unique MIDI files to the Million Song Dataset. The resulting "Lakh MIDI Dataset" provides a potential bounty of ground truth information for audio content-based music information retrieval. This can include transcription, meter, lyrics, and high-level musicological features. The reliability of the resulting annotations depends both on the quality of the transcription and the accuracy of the score-to-audio alignment. We therefore establish a baseline of reliability for score-derived information for different content-based MIR tasks. Finally, we discuss potential future uses of our dataset and the learning-based sequence comparison methods we developed.

Contents

List of Figures

List of Tables

List of Abbreviations

- BPTT - Backpropagation Through Time
- CQT - Constant-Q Transform
- DFT - Discrete Fourier Transform
- DHS - Downsampled Hash Sequence
- DTW - Dynamic Time Warping
- EI - Expected Improvement
- FFT - Fast Fourier Transform
- GPU - Graphical Processing Unit
- LMD - Lakh MIDI Dataset
- MIDI - Musical Instrument Digital Interface
- MIR - Music Information Retrieval
- MIREX - Music Information Retrieval Evaluation eXchange
- MRR - Mean Reciprocal Rank
- MSD - Million Song Dataset

- NAG - Nesterov's Accelerated Gradient
- PI - Probability of Improvement
- PSE - Pairwise Sequence Embedding
- RNN - Recurrent Neural Networks
- RMS - Root Mean Squared
- SGD - Stochastic Gradient Descent
- SPL - Sound Pressure Level
- STFT - Short-Time Fourier Transform
- TPAA - Thresholded Piecewise Aggregate Approximation
- UCB - Upper Confidence Bound

Acknowledgments

This thesis is dedicated to Joanna Percher. Thanks for being with me every step of the way, and I look forward to our next adventure. I love you.

I'm lucky to have overall had an extremely positive PhD experience, and I have a lot of people to thank for that.

Incidentally, I will be the last person in my immediate family to receive a PhD, though I overlapped some with my brother and Mom. Combined with the fact that they have known me my whole life, this gave them an unmatched ability to guide me through this process. Their support, advice, and love has brought me to where I am. Mom and Dad, thank you for giving me my inquisitiveness and an occasional one-track mind, and always being available to talk. Forrest, thanks for checking in so often and for your humor when I needed it.

DAn is a strange advisor in that he only really gives you advice if you ask for it, and he certainly has never insisted that I work on one project or another. In this sense, his relationship to me more closely resembled that of an interested bystander, who gently provided insight and guidance when necessary. Nevertheless, looking back at the work I did at LabROSA reveals his strong influence in my ideas and their execution. In other words, he managed to excel as an advisor while never acting the part, which in the end has gained me both a mentor and a friend. DAn, thank you for this opportunity.

I'm fortunate to have a thesis committee whose members I can also call friends. Brian undoubtedly became a sort of "second advisor" to me during his years as a postdoc at LabROSA, and much of what I learned during that time and my research philosophy is thanks to him. John's instruction early in my PhD made me certain that I had chosen the right program. Nima supported my creation of the Columbia Neural Network Reading Group/Seminar Series, which proved to be an excellent way for me to learn much of the background needed for my thesis work. Michael provided a sort of spiritual predecessor at LabROSA, and more concretely furnished the thesis template I am using. Thank you all for being a part of this journey.

I joined LabROSA at the same time as two other PhD students, Zhuo and Dawen. It's been great to have comrades to share this experience with. While we seldom worked together directly, we certainly benefitted from each other's company, discussions, and insight. I hope our paths continue to cross in the future.

Every summer, I hosted interns who each contributed in one way or another to the work in this thesis. Kitty, Hilary, Dylan, and Rafael, thank you for your help. LabROSA itself has also hosted many people who I'm happy to have met and worked with over the years. Matt, Thierry, Courtenay, Rachel, Cyril, Hélène, Diego, Minshu, James, Andy, and Douglas, thanks for making LabROSA a fun place to do my PhD. Our proximity to the Music and Audio Research Lab at NYU has also gained me friends and occasional collaborators. Eric, Justin, Uri, and Rachel, it has been great to get to know you all.

While I don't think I would otherwise have ever moved to New York City, it became my home over the course of three years thanks to my friends there. Thank you Alec, Annie, Hunter, Emelio, Josh, Sarah, Laura, Emma, and everyone else who I ever had fun with while living in NYC.

The work in this thesis would have been much harder and taken much longer

if not for the huge amount of open-source software I used which is distributed free of charge and restrictions. I particularly want to thank the developers of `Python`, `IPython`, `numpy`, `scipy`, `librosa`, `Theano`, `lasagne`, `numba`, `matplotlib`, `Spearmint`, `python-midi`, and `whoosh`. I also deeply appreciate the feedback and contributions to my own software projects from users and contributors.

I am grateful to the National Science Foundation for providing the majority of funding for my PhD.

Finally, I'd like to thank my cat Dexter, for his unwavering support.

Chapter 1

Introduction

Sequences of feature vectors or symbols are a natural way of representing information in a wide variety of fields, including multimedia, medicine, and natural language processing. For examples of a few common types of data represented as sequences, see figure 1.1. As the cost of storing data continues to decrease and more and more aspects of our lives are recorded, databases of sequential data are becoming larger and more prevalent. This motivates the need for methods to efficiently organize and extract useful information from this data.

An exemplary domain is musical data. Digital audio is commonly represented either as a sequence of sampled amplitude values corresponding to the sonic waveform, or as a sequence of time-frequency spectra computed over short snippets of the audio signal. Musical scores can also be conveniently represented digitally as a sequence of notes played on a collection of instruments. This has allowed for extremely large collections of musical audio recordings and scores to be assembled. The availability of this data has facilitated the nascent field of music information retrieval (MIR), which focuses on inferring high-level information from music through computational means.

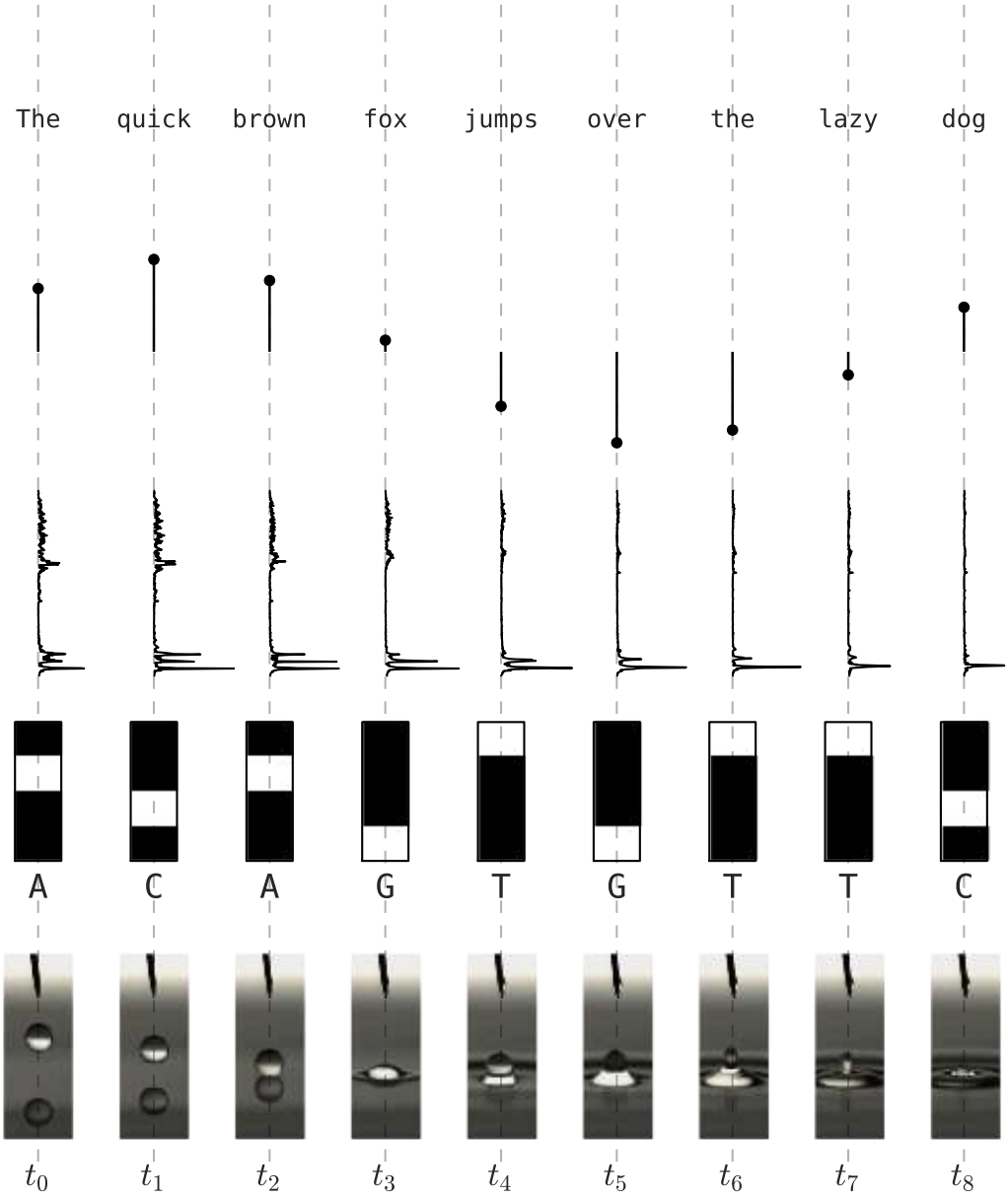

Figure 1.1: Examples of sequential data. From top to bottom: A sentence, which can be considered a sequence of words; a signal, which generally refers to a unidimensional real-valued sequence; a magnitude spectrogram (sequence of spectra) of an audio recording of the word "say"; a DNA sequence, which represents the order of nucleotides (G, C, A, or T) in a DNA molecule; and a video, which is a sequence of images.

1.1 Large-Scale Sequence Retrieval

Perhaps the most fundamental task one can perform with a database is indexing or nearest-neighbor retrieval, i.e. finding the entry in the database which is the most similar to a query. Given a way of measuring similarity, the simplest way to index a database is to compute the similarity between the query and every entry in the database and simply choose the entry which has the highest similarity score. One pervasive way of framing this problem is to represent database entries and queries as vectors in a space where their similarity is encoded by a distance metric. This makes item-to-item comparison simple and enables efficient nearest-neighbor retrieval.

In order to extend this method to sequential data, a natural approach would be to compute the sum of the distances of co-occurring (i.e. appearing at the same time) elements in two sequences. Figure 1.2 visualizes this method on two one-dimensional sequences. While straightforward, this method is made impractical for most problems of interest due to a few factors. First, it cannot be used to compare sequences which differ in length, which is a natural problem in many domains. Second, even when sequences have the same length, in many cases it is not necessarily true that two sequences are similar only when co-occurring elements in the sequences are similar. For example, the sentences “My favorite color is red” and “Red is my favorite color” have the same meaning (i.e. are extremely similar) but have no co-occurring words.

These issues have prompted the use of warping or edit distance measures, which allow for non-co-occurring elements of sequences to be compared when computing a similarity score. Such measures find a correspondence or alignment between elements of the two sequences being compared, and then compute the sum of the distances between aligned sequence elements. The alignment is typically defined to find a correspondence between sequence elements such that the total distance between aligned elements is minimized, and is required to follow some set of constraints

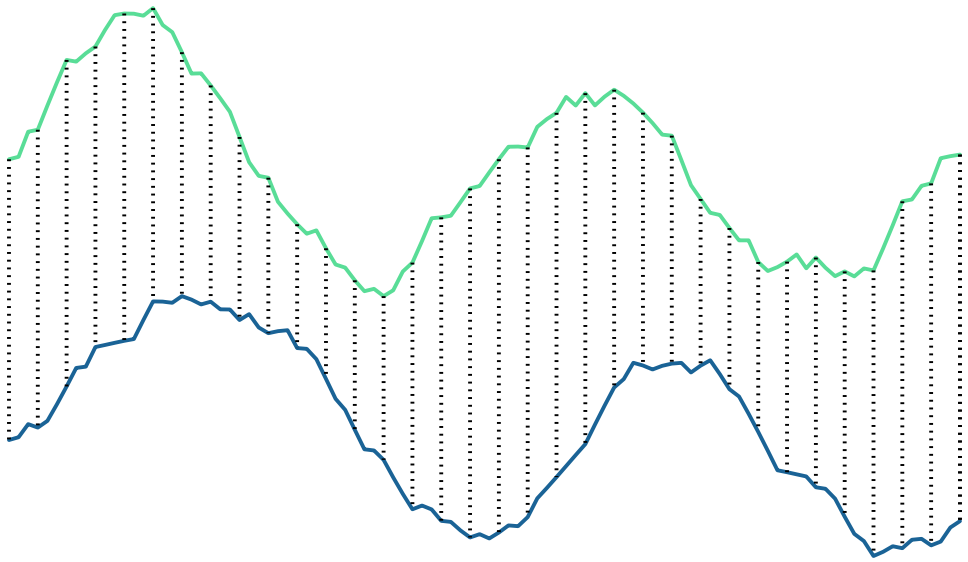

Figure 1.2: Comparing two equal-length sequences by summing the distances between co-occurring elements in each sequence. Dashed lines indicate which elements are being compared.

to ensure useful solutions. Because they enable comparison of non-co-occurring elements, these methods can often be applied straightforwardly to sequences which differ in length. In many cases, the number of possible alignments is exponential in the lengths of the sequences being compared. However, in most cases of interest (and used in practice), the optimal alignment can be found in time quadratic in the sequence lengths using dynamic programming. An example of this kind of comparison can be seen in figure 1.3.

While the application of dynamic programming makes many warping-based measures tractable, they can be made impractical in practice by a variety of factors. First, quadratic complexity can still be prohibitive when comparing very long sequences. This can be particularly problematic if sequences are oversampled, i.e. there is an

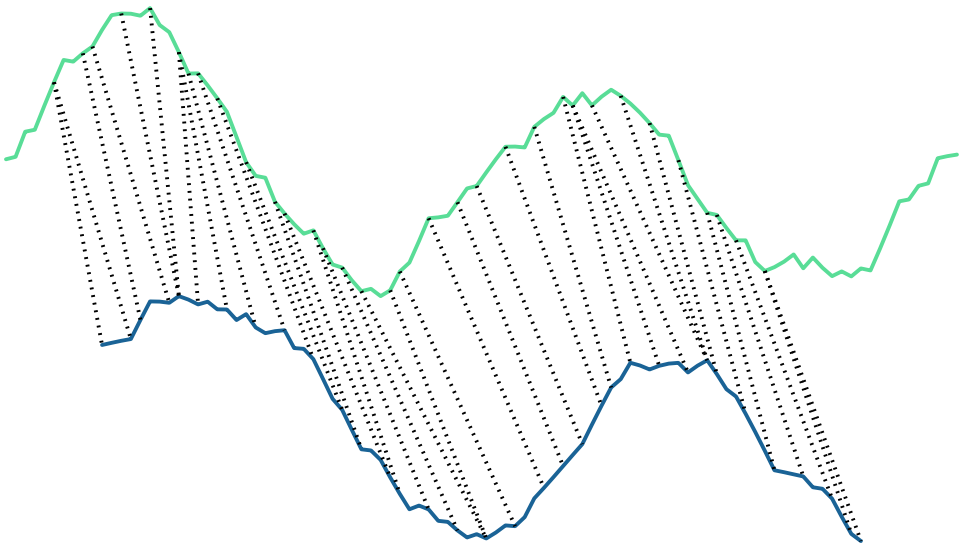

Figure 1.3: Comparing two sequences which differ in length by warping them to find an optimal alignment (under some constraints) and summing the distances between aligned elements. Dashed lines indicate which elements are being compared.

unnecessarily high correlation between consecutive sequence elements. Second, in some cases comparing individual sequence elements is itself prohibitively expensive, which is a necessary operation for any warping-based measure. Furthermore, in some domains (e.g. multimodal data), there is no natural way to compare individual sequence elements, which also precludes sequence comparison. Finally, if the database being indexed is very large, comparing a query against every entry in it can be prohibitively expensive.

A pervasive method for mitigating some of these issues is "pruning". Pruning describes the general technique of using a very cheap surrogate method to discard most entries in a database and then applying a more expensive, but more accurate,

method to the remaining entries. The effectiveness of a pruning method is quantified by how much of the database it is able to discard without falsely rejecting the correct match according to the more expensive operation.

One straightforward way to achieve pruning for indexing databases of sequential data is to use an alternate representation for the sequences which accelerates the warping similarity measure. This may include making the sequence shorter to mitigate oversampling issues or transforming the individual feature vectors which make up the sequence in such a way that comparing them is less expensive. Utilizing a different representation can also facilitate the case when the query sequence, in its raw form, is not comparable to entries in the database. To fit into the framework of pruning, the transformed sequence representation can discard some information and sacrifice some accuracy so long as it is substantially faster than the "native" warping measure and rarely discards the correct match.

Many of these issues are caused by the fact that the data is sequential and a warping measure must be used; they are not present when data is represented as fixed-length vectors whose distance measures similarity. This motivates another pruning method: Transform sequences of feature vectors to fixed-length vectors embedded in a space where their distance approximates the warping measure. Provided that the distance in the embedded space suitably approximates sequence similarity, these methods can also serve as effective pruning techniques.

1.2 Matching and Aligning Scores and Audio Recordings

Many of these issues are exhibited in the task of finding the entry in a large database of audio recordings which corresponds to a given musical score. Such a matching is of

great utility in the field of content-based music information retrieval due to the large amount of high-level information present in the score. This information can be used as ground truth for content-based MIR algorithms, whose success is often dependent on the quantity and quality of this data. Creating such ground truth by hand for a given song typically requires person-hours on the order of the duration of the song, and so training data availability is a frequent bottleneck in content-based MIR tasks. Leveraging existing collections of scores could greatly facilitate the curation of this data; through a large-scale web scrape, we obtained 178,561 unique MIDI files (a common digital score format) – orders of magnitude larger than any available ground truth dataset for MIR.

The mere existence of a large collection of scores is not enough: In order to use information from a score as ground truth, it needs to be matched (paired with a corresponding audio recording), aligned (adjusted so that the timing of the events in the file match the audio), and relevant information must be inferred or extracted. Given large corpora of audio recordings of music and scores, the task of matching entries of each type may seem to be a simple matter of fuzzy text matching of the files' metadata. However, MIDI files almost never contain structured metadata, and as a result the best-case scenario is that the artist and song title are included in the file or directory name. While we found some examples of this in our collection of scraped MIDI files, the vast majority of the files had effectively no metadata information. Figure 1.4 shows a random sampling of directory and filenames from our collection.

Since the goal of matching MIDI and audio files is to find pairs that have *content* in common, we can in principle identify matches regardless of metadata availability or accuracy. However, the task of matching a single MIDI file to a large corpus of audio recordings exhibits all of the issues described in the previous section, for the

```
J/Jerseygi.mid
V/VARIA18O.MID
Carpenters/WeveOnly.mid
2009 MIDI/handy_man1-D105.mid
G/Garotos Modernos - Bailanta De Fronteira.mid
Various Artists/REWINDNAS.MID
GoldenEarring/Twilight_Zone.mid
Sure.Polyphone.Midi/Poly 2268.mid
d/danza3.mid
100%sure.polyphone.midi/Fresh.mid
rogers_kenny/medley.mid
2009 MIDI/looking_out_my_backdoor3-Bb192.mid
```

Figure 1.4: Random sampling of 12 MIDI filenames and their parent directories from our corpus of 178,561 unique MIDI files scraped from the Internet.

following reasons: First, scores and audio recordings are not directly comparable in their raw forms. Second, in order to capture relevant fine-grained temporal information, time-frequency sequence representations of a typical length song may consist of thousands of feature vectors. Furthermore, individual spectra in a time-frequency representation are high-dimensional real-valued feature vectors, which makes comparing them expensive. Given the size of our MIDI file collection, we require a highly-efficient scheme to match the content of MIDI and audio files.

A given score which has been matched to an audio recording of the song it is a transcription of often does not share the exact timing of the recording. In order to fully leverage the information in a given score, a mapping between the timing in the score and the audio recording must therefore be constructed. Fortunately, score-to-audio alignment is a thoroughly studied research topic. However, most systems are hand-designed to maximize performance on a single task or small corpus of music. Comparison of the different alignment systems which have been proposed

is greatly inhibited by the lack of a large corpus of known-correct alignments to evaluate performance on. The creation of a task or dataset for evaluating score-to-audio alignment systems would help determine which system design performs best in general.

Finally, once matched and aligned, the relevant information for content-based MIR must be extracted from the score. This can include transcription, meter, lyrics, and high-level musicological features. Extracting score-derived ground truth requires parsing the score's format (e.g. the MIDI protocol) and then performing any post-processing needed to derive higher-level information. The reliability of the resulting annotations will depend both on the quality of the transcription and the accuracy of the score-to-audio alignment. It would therefore be useful to establish a baseline of reliability for score-derived information for different content-based MIR tasks.

1.3 Overview

The remainder of this thesis focuses on methods for efficient large-scale sequence comparison and tests them on the specific problem of matching scores to corresponding audio recordings. Because this task exhibits all of the problematic characteristics of large-scale sequence comparison problems, we consider it an appropriate benchmark to evaluate the usefulness of the proposed methods. In addition, by developing an effective system for matching a MIDI file to its corresponding entry in a database of audio recordings, we will be able to leverage our large collection of MIDI files to create valuable ground truth information for content-based MIR. Throughout, we will focus on *learning-based methods*, i.e. algorithms which are optimized to achieve high performance over a collection of exemplary data.

The rest of this thesis is organized as follows. First, in chapter 2, we present a collection of "tools" which will be used throughout this thesis, including machine

learning and signal processing methods. To motivate the central problem of this thesis, in chapter 3 we discuss the various sources of information available in MIDI files and provide a broad overview of the availability of this information in the 178,561 unique MIDI files we obtained through a large-scale web scrape. In chapter 4 we propose a synthetic task to measure the performance of score-to-audio alignment systems and use Bayesian optimization to choose a system design which has the best general performance. We also address the problem of reporting a reliable alignment confidence score and validate the resulting system on real-world data. Chapter 5 describes our first technique for pruning large-scale sequence similarity search. This method learns a mapping from sequences of continuously-valued feature vectors to downsampled sequences of binary vectors. The resulting sequences are orders of magnitude faster to compare and only result in a small decrease in precision. We propose a second pruning method in chapter 6 which maps sequences to fixed-length vectors in a Euclidean space where their distance approximates sequence similarity. This provides an estimate of the similarity of two sequences by an inexpensive Euclidean distance calculation. Finally, chapter 7 utilizes all of these methods to construct a large collection of MIDI files which have been matched and aligned to corresponding audio recordings. We also address the practical question of the quality of ground truth information extracted from MIDI files. To wrap up in chapter 8, we enumerate our contributions, discuss future work, and outline some interesting applications for our new dataset.

1.4 Notation

In this thesis, we will not make an explicit notational distinction between variables with differing numbers of dimensions, but we will always state the dimensionality of variables when they are introduced. For example, a real-valued scalar x would

be introduced as $x \in \mathbb{R}$, a vector y with N entries in a Hamming space (i.e. having binary values) would be introduced as $y \in \mathbb{H}^N$, and a natural number-valued matrix z with dimensions M and N would be introduced as $z \in \mathbb{N}^{M \times N}$. To refer to entries in these variables, we will use "indexing" notation, e.g. $y[n]$ would denote the n^{th} entry of the just-introduced vector y. We will always count these entries starting from one, e.g. $y = [y[1], \ldots, y[N]]$. On the other hand, when referring to members of a collection of things, we will use subscript notation, e.g. a collection of K variables might be denoted $x_1, x_2, \ldots, x_K$. Finally, we will use $\jmath$ to refer to the imaginary number $\sqrt{-1}$ and use an overline to refer to complex conjugation, e.g. $\overline{a + b\jmath} = a - b\jmath$.

For visualization, we will use the following color palette:

In captions, we will refer to these colors (from left to right) as "blue", "green", "gray", "red", and "orange". When visualizing two-dimensional data, we will use the following colormap:

These colors will reflect smaller to larger values from left to right respectively.

Chapter 2

Tools

Throughout this thesis, we will use a common collection of tools to help solve problems of interest. Broadly speaking, we will make use of techniques from the fields of machine learning and signal processing. In this chapter, we give a high-level overview of these fields and a more specific definition of the various techniques we will utilize.

2.1 Machine Learning

Machine learning is broadly defined as a suite of methods for enabling computers to carry out particular tasks without being explicitly programmed to do so. Such methods make use of a collection of data, from which patterns or the desired behaviors are automatically determined. In this sense, machine learning techniques allow computers to "learn" the correct procedure based on the data provided. In addition to data, machine learning algorithms also often require an objective which quantifies the extent to which they are successfully carrying out a task. This objective function allows them to be optimized in order to maximize their performance.

Traditionally, machine learning techniques are divided into three types: Supervised, unsupervised, and reinforcement learning. Supervised learning requires a collection of training data that specifies both the input to the algorithm and the desired output. Unsupervised learning, on the other hand, does not utilize target output data, and thus is limited to finding structure in the input data. Finally, reinforcement learning refers to the broader problem of determining a policy to respond to input data in a dynamic "environment" based on a potentially rare signal which tells the algorithm whether it is successful or not. All of the problems in this thesis fall into the category of supervised learning, so we will not discuss unsupervised or reinforcement learning further.

2.1.1 Supervised Learning

In the supervised setting, our machine learning algorithm produces a function f (called a *model*) which, given input x and the function's parameters θ, produces an output $\hat{y}$:

$$\hat{y} = f(x, \theta) \tag{2.1}$$

Supervised learning requires a training set $\mathcal{S}$ of input-target pairs

$$(x_1, y_1), (x_2, y_2), \ldots, (x_{|\mathcal{S}|}, y_{|\mathcal{S}|}) \tag{2.2}$$

where $x_n \in \mathcal{X}$ (the input space) and $y_n \in \mathcal{Y}$ (the target space). We assume that these pairs are drawn from an unknown joint probability distribution $p(x, y)$, of which the only information we explicitly know is the training set $\mathcal{S}$. The goal of a supervised learning algorithm is to approximate the probability distribution $p(x, y)$ (or just salient characteristics of it) with f by adjusting the parameters θ based solely on $\mathcal{S}$.

One way to formulate this goal is to state that for a sample from $p(x, y)$ which is not in $\mathcal{S}$, the function f should produce the correct output. Of course, because we are only given the samples in $\mathcal{S}$, the parameters θ must be optimized solely according the training pairs in $\mathcal{S}$. This approach, where f is used directly to produce the predicted value for y, is referred to as discriminative training. An alternative approach, called generative or probabilistic training, is to use f to model $p(x, y)$ by estimating $p(y|x)$ and then choosing the most probable y. In either case, adjusting the parameters θ of f typically involves using a task-specific "loss function" $\mathcal{L}$ which computes a lower-bounded scalar value which measures the extent to which the model's output $\hat{y}$ agrees with the target y. Ideally, minimizing this loss over the training set by adjusting θ will well-approximate the minimum over the underlying distribution.

The choice of a loss function will depend on the characteristics of the problem, in particular the target space $\mathcal{Y}$. For example, in *classification* problems, the targets y may take one of K discrete values. A natural loss function in this case is the zero-one loss

$$\mathcal{L}(y, \hat{y}) = \begin{cases} 1, & \hat{y} = y \\ 0, & \hat{y} \neq y \end{cases} \tag{2.3}$$

In *regression* problems, the targets are continuously valued, and a common loss function is the squared error

$$\mathcal{L}(y, \hat{y}) = (y - \hat{y})^2 \tag{2.4}$$

Note that in both cases when $y = \hat{y}$, $\mathcal{L} = 0$; this is a common property of loss functions.

In general, a suitable goal for training a supervised learning algorithm is to adjust θ to minimize the average loss over the training set:

$$\frac{1}{|\mathcal{S}|} \sum_{(x,y)\in\mathcal{S}} \mathcal{L}(y, f(x, \theta)) \tag{2.5}$$

Minimizing this average loss is equivalent to minimizing the empirical risk of f. A common paradigm is to compute the derivative of the loss function $\mathcal{L}$ with respect to the model parameters θ and use this derivative information to incrementally change θ to make $\mathcal{L}$ smaller. In this case, it is beneficial to have a loss function which is continuous and smooth, which precludes the use of loss functions like the zero-one loss (equation (2.3)). A common solution to this problem is to use a surrogate loss function which has the desired properties (e.g. smoothness) and whose minimization approximates the minimization of $\mathcal{L}$.

There are supervised learning methods which do not follow the above framework. For example, k-nearest neighbors (Cover and Hart, 1967) is a classification algorithm which labels its input by assigning it the most common y among the k points in $\mathcal{S}$ which are closest to the input. This classifier has no loss function or indeed any parameters other than k and $\mathcal{S}$ itself (i.e. it is nonparametric). However, all of the methods we will utilize in this thesis follow the recipe of defining a model with parameters θ which are adjusted by minimizing the average loss over $\mathcal{S}$.

2.1.2 Sequential Data

A common scenario is that the data space $\mathcal{X}$ is fixed-dimensional, e.g. $\mathcal{X} \in \mathbb{R}^{D_1 \times D_2 \times \cdots D_N}$. However, in some cases, one (or more) dimensions may vary in size. An important example of this is sequential data, where a "temporal" dimension can differ between examples. In addition, for real-world data the temporal dimension often contains

strong dependencies and correlations. For some examples of common types of data typically represented as sequences, see figure 1.1.

An illustrative example is text data where each datapoint is a sentence. Clearly, the length of each sequence will vary (e.g. "I am happy." and "This sentence is a few words longer." differ in length). In addition, there may be a strong dependency between a given word in a sentence and the words that precede it. For example, "I am hungry, please give me" is much more likely to be followed by the word "food" than, say, "dirt".

Different possible tasks involving sequential data can be broadly divided into three categories: Sequence classification, embedding, and transduction. Sequence classification involves applying a single label to entire sequences; for example, classifying which of a finite number of people was speaking in an audio recording of an utterance. Sequence embedding involves representing sequences as a single point in a fixed-dimensional space with some desired properties. For example, instead of classifying which person was speaking, we might instead embed audio recordings as vectors in a Euclidean space such that recordings of the same speaker have a small embedded distance and recordings of different speakers have a large distance. Finally, sequence transduction involves converting an input sequence to a new sequence, potentially of different length and whose elements may lie in a different space. Speech recognition is an exemplary case, where a sequence of discrete words must be generated based on an audio recording.

2.1.3 Neural Networks

Most of the machine learning algorithms used in this thesis will come from a class of models known as "neural networks". Neural networks can be considered learnable functions which consist of a sequence of nonlinear processing stages. First proposed

in the 1940s (McCulloch and Pitts, 1943; Hebb, 1949), these models were designed as an artificial model of the way the brain was understood to work at that time. The modern understanding of the brain has diverged substantially from this model; nevertheless, the name "neural network" has endured.

A predecessor to modern neural network models was the perceptron algorithm (Rosenblatt, 1958). The perceptron is a simple linear model which can perform binary classification of its input. Given an input feature vector $x \in \mathbb{R}^D$, the perceptron computes

$$f(x) = \begin{cases} 1, & w^\top x + b > 0 \\ 0, & \text{otherwise} \end{cases} \tag{2.6}$$

where $w \in \mathbb{R}^D$ and $b \in \mathbb{R}$ are the parameters of the model. These parameters are adjusted so that the desired label (1 or 0) is produced given different inputs. A common visualization of the perceptron is shown in figure 2.1.

The perceptron algorithm can only correctly classify its input when the data is linearly separable by class. To remedy this, the "multilayer perceptron" was proposed, which can be viewed as a sequence of perceptrons organized in "layers", each taking its input from the previous perceptron. In the multilayer perceptron, each perceptron can output a vector of values rather a single value. Furthermore, rather than producing a binary 1 or 0 depending on the sign of $w^\top x + b$, the layers in a multilayer perceptron may use any of a variety of nonlinear functions. In summary, a given layer's output is computed as

$$a = f(x) = \sigma(Wx + b) \tag{2.7}$$

where σ is a nonlinear "activation" function, $x \in \mathbb{R}^D$ is the layer's input, $W \in \mathbb{R}^{M \times D}$ is the weight matrix, $b \in \mathbb{R}^M$ is the bias vector, $a \in \mathbb{R}^M$ is the layer's output, and M

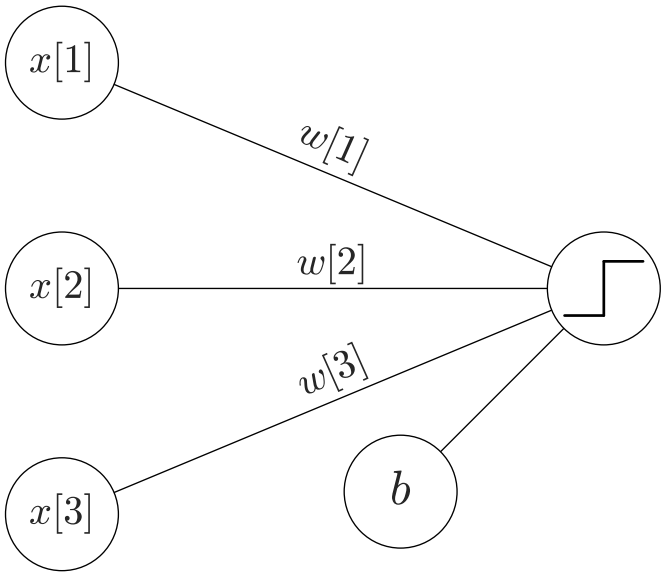

Figure 2.1: Diagram visualizing the perceptron of equation (2.6). Each entry of $x \in \mathbb{R}^3$ is multiplied by an entry of the weight vector w, and the result is summed along with the bias b before being passed into a Heaviside step function.

is the output dimensionality of the layer. The individual entries of a are referred to as the layer's "units". The original perceptron is recovered when σ is the Heaviside step function and $M = 1$. Under this generic definition, multilayer perceptrons are, in fact, equivalent to what are now called feedforward networks; we will use this terminology for the remainder of this thesis. Figure 2.2 shows a common schematic used to represent feedforward networks.

In parallel seminal works (Cybenko, 1989; Hornik et al., 1989), it was shown that feedforward networks with at least two layers with sufficiently many units are *universal function approximators* when the activation function of each layer is a "squashing" function (i.e. is monotonically increasing, $\lim_{x\to\infty} \sigma(x) = 1$ and $\lim_{x\to-\infty} \sigma(x) = 0$). This implies that such models can approximate any function to arbitrary precision.

Figure 2.2: Schematic of a feedforward network (realized as a belt buckle). Each column of nodes represents the number of units in each layer; this particular diagram shows two input units, a single hidden layer with three units, and a single output unit. Lines indicate connections between units in each layer, i.e. entries of each weight matrix. Aside from those in the input layer, each node indicates summing of its input, adding a bias, and applying a nonlinearity.

Of course, the ability of a model to approximate any function is only valuable when there is a reliable method to adjust its parameters so that the desired function is suitably approximated. Currently, the most pervasive method to achieve this is "backpropagation", which computes the gradient of an error function which measures the deviation between the network's output and the target output with respect to the model parameters (Rumelhart et al., 1986). We give a derivation of backpropagation for feedforward networks in the following section. In the supervised learning context, the error function computes the deviation between the network's output and the target for input-target pairs in the training set $\mathcal{S}$. As a result, for supervised learning, universal approximation only implies that the multilayer feedforward network can "memorize" the target output for each input in the training set $\mathcal{S}$ but not that it

can produce the correct y for $x \notin \mathcal{S}$. This is referred to as "overfitting", and is an undesirable property of machine learning models.

Despite this issue, feedforward networks have proven to be extremely effective models in a variety of machine learning tasks. This success is thanks to a suite of methods and model designs which allow them to be trained effectively, over potentially very large datasets, while avoiding overfitting. It has also led to the development of variants which are more appropriate for sequential data (recurrent networks) and data with local regularities (convolutional networks). In the following sections, we give an overview of these different developments and approaches.

2.1.3.1 Backpropagation

Backpropagation is a method for efficiently computing the gradient of a loss function applied to a neural network with respect to its parameters. These partial derivatives can then be used to update the network's parameters using gradient descent methods. In truth, backpropagation is a special case of the more generic "reverse-mode automatic differentiation" method. Deriving backpropagation involves numerous clever applications of the chain rule for functions of vectors.

While originally introduced earlier, the widespread adoption of backpropagation for training neural networks is largely thanks to Rumelhart et. al's seminal paper (Rumelhart et al., 1986). Backpropagation is so successful and widely applied that in some sense the term "neural network" has become genericized to mean "a machine learning model for which backpropagation is applicable". Because it is fundamental in many of the approaches proposed in this thesis, we give a full derivation of backpropagation for a simple multilayer feedforward network in this section. A similar, higher-level and more verbose derivation can be found in (Nielsen, 2015).

By way of review, the chain rule is a way to compute the derivative of a function whose variables are themselves functions of other variables. If $\mathcal{L}$ is a scalar-valued function of a scalar z and z is itself a scalar-valued function of another scalar variable w, then the chain rule states that

$$\frac{\partial \mathcal{L}}{\partial w} = \frac{\partial \mathcal{L}}{\partial z}\frac{\partial z}{\partial w} \tag{2.8}$$

For scalar-valued functions of more than one variable (e.g. a scalar-valued function of a vector), the chain rule essentially becomes additive. In other words, if L is a scalar-valued function of a vector $z \in \mathbb{R}^N$, whose entries are each a function of some variable w, the chain rule states that

$$\frac{\partial \mathcal{L}}{\partial w} = \sum_{i=1}^{N} \frac{\partial \mathcal{L}}{\partial z[i]}\frac{\partial z[i]}{\partial w} \tag{2.9}$$

In the following, we'll discuss backpropagation through an L-layer feedforward network, where N_n is dimensionality of layer $n \in \{0, \ldots, L\}$. N_0 is the dimensionality of the input; N_L is the dimensionality of the output. The m^{th} layer, for $m \in \{1, \ldots, L\}$ has parameters $W_m \in \mathbb{R}^{N_m \times N_{m-1}}$, the weight matrix, and $b_m \in \mathbb{R}^{N_m}$, the bias vector. $W_m[i, j]$ is the weight between the i^{th} unit in layer m and the j^{th} unit in layer $m-1$. Each layer computes $z_m \in \mathbb{R}^{N_m}$, a linear mix of its inputs, by $z_m = W_m a_{m-1} + b_m$, and the activation $a_m = \sigma_m(z_m)$ where σ_m is the layer's nonlinear activation function. a_L is the output of the network. We define the special case a_0 as the input of the network. We denote $y \in \mathbb{R}^{N_L}$ as the target output, treated as a constant. The loss function $\mathcal{L}(a_L, y)$ is used to measure the error of the network's output compared to the target.

In order to train the network using a gradient descent algorithm, we need to know the gradient of each of the parameters with respect to the loss function $\mathcal{L}$; that is,

we need to know $\partial\mathcal{L}/\partial W_m$ and $\partial\mathcal{L}/\partial b_m$. It will be sufficient to derive an expression for these gradients in terms of the following terms, which we can compute based on the neural network's architecture: $\partial\mathcal{L}/\partial a_L$, the derivative of the loss function with respect to the output of the network, and $\partial a_m/\partial z_m$, the derivative of the nonlinearity used in layer m with respect to its argument z_m.

To compute the gradient of our loss function $\mathcal{L}$ with respect to $W_m[i,j]$ (a single entry in the weight matrix of the layer m), we can first note that $\mathcal{L}$ is a function of a_L, which is itself a function of the linear mix output entries $z_m[k]$, which are themselves functions of the weight matrices W_m and biases b_m. With this in mind, we can use the chain rule as follows:

$$\frac{\partial\mathcal{L}}{\partial W_m[i,j]} = \sum_{k=1}^{N_m} \frac{\partial\mathcal{L}}{\partial z_m[k]} \frac{\partial z_m[k]}{\partial W_m[i,j]} \tag{2.10}$$

Note that by definition

$$z_m[k] = \sum_{l=1}^{N_m} W_m[k,l]a_{m-1}[l] + b_m[k] \tag{2.11}$$

It follows that $\partial z_m[k]/\partial W_m[i,j]$ will evaluate to zero when $k \neq i$ because $z_m[k]$ does not interact with any elements in W_m except for those in the k^{th} row, and we are only considering the entry $W_m[i,j]$. When $k = i$, we have

$$\frac{\partial z_m[i]}{\partial W_m[i,j]} = \frac{\partial}{\partial W_m[i,j]} \left(\sum_{l=1}^{N_m} W_m[i,l]a_{m-1}[l] + b_m[i] \right) \tag{2.12}$$

$$= a_{m-1}[j] \tag{2.13}$$

$$\rightarrow \frac{\partial z_m[k]}{\partial W_m[i,j]} = \begin{cases} 0 & k \neq i \\ a_{m-1}[j] & k = i \end{cases} \tag{2.14}$$

Combining equation (2.14) with equation (2.10) causes the summation to collapse, giving

$$\frac{\partial \mathcal{L}}{\partial W_m[i,j]} = \frac{\partial \mathcal{L}}{\partial z_m[i]} a_{m-1}[j] \tag{2.15}$$

or in vector form

$$\frac{\partial \mathcal{L}}{\partial W_m} = \frac{\partial \mathcal{L}}{\partial z_m} a_{m-1}^{\top} \tag{2.16}$$

Similarly for the bias variables b_m, we have

$$\frac{\partial \mathcal{L}}{\partial b_m[i]} = \sum_{k=1}^{N_m} \frac{\partial \mathcal{L}}{\partial z_m[k]} \frac{\partial z_m[k]}{\partial b_m[i]} \tag{2.17}$$

As above, it follows that $\partial z_m[k]/\partial b_m[i]$ will evaluate to zero when $k \neq i$ because $z_m[k]$ does not interact with any element in b_m except $b_m[k]$. When $k = i$, we have

$$\frac{\partial z_m[i]}{\partial b_m[i]} = \frac{\partial}{\partial b_m[i]} \left(\sum_{l=1}^{N_m} W_m[il] a_{m-1}[l] + b_m[i] \right) \tag{2.18}$$

$$= 1 \tag{2.19}$$

$$\rightarrow \frac{\partial z_m[k]}{\partial b_m[i]} = \begin{cases} 0 & k \neq i \\ 1 & k = i \end{cases} \tag{2.20}$$

Combining equation (2.20) and equation (2.17) again causes the summation to collapse to give

$$\frac{\partial \mathcal{L}}{\partial b_m[i]} = \frac{\partial \mathcal{L}}{\partial z_m[i]} \tag{2.21}$$

or in vector form

$$\frac{\partial \mathcal{L}}{\partial b_m} = \frac{\partial \mathcal{L}}{\partial z_m} \tag{2.22}$$

Now, we must compute $\partial\mathcal{L}/\partial z_m[k]$. For the final layer ($m = L$), this term is straightforward to compute using the chain rule:

$$\frac{\partial\mathcal{L}}{\partial z_L[k]} = \frac{\partial\mathcal{L}}{\partial a_L[k]}\frac{\partial a_L[k]}{\partial z_L[k]} \tag{2.23}$$

or, in vector form

$$\frac{\partial\mathcal{L}}{\partial z_L} = \frac{\partial\mathcal{L}}{\partial a_L}\frac{\partial a_L}{\partial z_L} \tag{2.24}$$

The first term $\partial\mathcal{L}/\partial a_L$ is just the derivative of the loss function with respect to its argument, whose form depends on the loss function chosen. Similarly, $\partial a_m/\partial z_m$ (for any layer m including L) is the derivative of layer m's nonlinearity with respect to its argument and will depend on the choice of nonlinearity. For other layers, we again invoke the chain rule:

$$\frac{\partial\mathcal{L}}{\partial z_m[k]} = \frac{\partial\mathcal{L}}{\partial a_m[k]}\frac{\partial a_m[k]}{\partial z_m[k]} \tag{2.25}$$

$$= \left(\sum_{l=1}^{N_{m+1}} \frac{\partial\mathcal{L}}{\partial z_{m+1}[l]}\frac{\partial z_{m+1}[l]}{\partial a_m[k]}\right)\frac{\partial a_m[k]}{\partial z_m[k]} \tag{2.26}$$

$$= \left(\sum_{l=1}^{N_{m+1}} \frac{\partial\mathcal{L}}{\partial z_{m+1}[l]}\frac{\partial}{\partial a_m[k]}\left(\sum_{h=1}^{N_m} W_{m+1}[l,h]a_m[h] + b_{m+1}[l]\right)\right)\frac{\partial a_m[k]}{\partial z_m[k]} \tag{2.27}$$

$$= \left(\sum_{l=1}^{N_{m+1}} \frac{\partial\mathcal{L}}{\partial z_{m+1}[l]}W_{m+1}[l,k]\right)\frac{\partial a_m[k]}{\partial z_m[k]} \tag{2.28}$$

$$= \left(\sum_{l=1}^{N_{m+1}} W_{m+1}^{\top}[k,l]\frac{\partial\mathcal{L}}{\partial z_{m+1}[l]}\right)\frac{\partial a_m[k]}{\partial z_m[k]} \tag{2.29}$$

$$\tag{2.30}$$

Nonlinearity	$a_m = \sigma_m(z_m)$	$\frac{\partial a_m}{\partial z_m}$
Logistic	$\frac{1}{1+e^{z_m}}$	$a_m(1-a_m)$
tanh	$\frac{e^{z_m}-e^{-z_m}}{e^{z_m}+e^{-z_m}}$	$1-(a_m)^2$
Rectifier	$\max(0, z_m)$	$\begin{cases} 0, & z_m < 0 \\ 1, & z_m \geq 0 \end{cases}$

Table 2.1: Common nonlinearities $\sigma_m(z_m)$ used in neural networks and their derivatives with respect to their input z_m.

where the last simplification was made because by convention $\partial\mathcal{L}/\partial z_{m+1}$ is a column vector, allowing us to write the following vector form:

$$\frac{\partial \mathcal{L}}{\partial z_m} = \left(W_{m+1}^\top \frac{\partial \mathcal{L}}{\partial z_{m+1}}\right) \circ \frac{\partial a_m}{\partial z_m} \tag{2.31}$$

where $\circ$ is the element-wise (Hadamard) product.

We now have the ingredients to efficiently compute the gradient of the loss function with respect to the network's parameters: First, we compute $\partial\mathcal{L}/\partial z_L$ based on the choice of loss function and $\partial a_m/\partial z_m$ based on each layer's nonlinearity. Then, we recursively can compute $\partial\mathcal{L}/\partial z_m$ layer-by-layer based on the term $\partial\mathcal{L}/\partial z_{m+1}$ computed from the previous layer (this is called the "backward pass"). Finally, the resulting $\partial\mathcal{L}/\partial z_m$ terms are then used to compute $\partial\mathcal{L}/\partial W_m$ and $\partial\mathcal{L}/\partial b_m$ using equation (2.10) and equation (2.17) respectively. Using this recipe requires $\partial\mathcal{L}/\partial z_L$ and $\partial a_m/\partial z_m$ to be known ahead of time. By way of example, table 2.1 and table 2.2 list some common loss functions and nonlinearities and their appropriate derivatives.

In summary, backpropagation proceeds in the following manner for each training sample: For the "forward pass", given the network input a_0, compute a_m for $m \in \{1, \ldots, L\}$ recursively by $a_m = \sigma_m(W_m a_{m-1} + b_m)$. For the "backward pass",

Loss Function	$\mathcal{L}(a_L, y)$	$\frac{\partial \mathcal{L}}{\partial a_L}$
Squared Error	$\frac{1}{2}(y - a_L)^\top (y - a_L)$	$y - a_L$
Cross-Entropy	$(y-1)\log(1 - a_L) - y \log(a_L)$	$\frac{a_L - y}{a_L(1 - a_L)}$

Table 2.2: Common loss functions $\mathcal{L}(a_L, y)$ used in neural networks and their (sub)gradients with respect to their input a_L.

compute

$$\frac{\partial \mathcal{L}}{\partial z_L} = \frac{\partial \mathcal{L}}{\partial a_L} \frac{\partial a_L}{\partial z_L} \tag{2.32}$$

whose terms are all known a priori, then recursively compute

$$\frac{\partial \mathcal{L}}{\partial z_m} = \left(W_{m+1}^\top \frac{\partial \mathcal{L}}{\partial z_{m+1}} \right) \circ \frac{\partial a_m}{\partial z_m} \tag{2.33}$$

and use the resulting values to compute

$$\frac{\partial \mathcal{L}}{\partial W_m} = \frac{\partial \mathcal{L}}{\partial z_m} a_{m-1}^\top \tag{2.34}$$

and

$$\frac{\partial \mathcal{L}}{\partial b_m} = \frac{\partial \mathcal{L}}{\partial z_m} \tag{2.35}$$

From this procedure, we can see why backpropagation is an efficient way to compute these partial derivatives: By reusing terms, all of the partial derivatives can be computed by a single forward and backward pass. For a feedforward network, the computational cost of each of these passes is effectively L matrix multiplications. A simple alternative would be to measure the change in $\mathcal{L}$ by adjusting each of the parameters (i.e. individual elements of the weight matrices and bias vectors) by a small amount. However, this would require a separate forward pass for each

parameter. Modern neural networks contain millions of parameters, so clearly backpropagation is an efficient choice.

2.1.3.2 Tricks

Combined with a gradient descent technique, backpropagation provides an effective method for optimizing the parameters of a neural network ("training") to minimize a given loss function. However, training neural networks is made difficult in practice by a variety of factors. First, the composition of the many nonlinear processing layers in a neural network make the resulting loss function highly non-convex. Non-convexity means that, when minimizing the network's loss function with gradient descent methods, there is no guarantee that a given stationary point is a global minimum. It also implies that both the way the parameters are initialized prior to optimization and the gradient descent technique used may have a very strong effect on the best solution found during optimization.

As mentioned above, the universal approximation property of neural networks can also have negative implications for their utility. The fact that a possible outcome of training a neural network is the simple memorization of correspondences in the training set means that the resulting model may be of little use in practice. As a result, in many cases it is necessary to use some form of regularization when training the network to prevent it from overfitting to the training set.

Finally, feedforward networks with many layers are also prone to the problem of "vanishing and exploding gradients". This issue is caused by the fact that as gradients are backpropagated through layers, they are multiplied by each successive weight matrix and the derivative of each nonlinearity (as shown in equation (2.31)). When the spectral norm of the weight matrices and/or the gradients of the nonlinearities are substantially greater than or less than one, the resulting backpropagated gradients

can become exponentially smaller or larger as they approach "early" layers in the network. This can effectively prevent the parameters of the first few layers from being adjusted, which can greatly inhibit learning.

Fortunately, there has been a great deal of development into methods for mitigating these issues. Collectively, these techniques have been dubbed "deep learning" thanks to their applicability to networks with many layers. In this section, we motivate and describe those techniques which are used in this thesis. Practical and broader overviews can be found in (Bengio, 2012; LeCun et al., 2012, 2015).

Number and Size of Layers When designing a neural network model, one of the first choices which must be made is the number of layers and their sizes (output dimensionality). Because early theory proved that networks with a single hidden layer could approximate any function arbitrarily well given that the layer was sufficiently "wide" (having many units) (Cybenko, 1989; Hornik et al., 1989), and because networks with many layers can be more difficult to train due to the vanishing and exploding gradients problem (Glorot and Bengio, 2010; Erhan et al., 2009), early applications of neural networks typically only used a single hidden layer. However, recent results have shown that using many layers can help networks learn a hierarchical transformation to more and more abstract representations which, for real-world problems, can be highly beneficial (Bengio, 2009). Furthermore, it has been shown that for some simple problems, networks with many hidden layers are exponentially more efficient than networks with a single hidden layer with many units (Bengio and Delalleau, 2011). Coupled with recent advances in training very deep networks (Glorot et al., 2011; Glorot and Bengio, 2010; Bengio et al., 2007; Hinton et al., 2006, 2012; Hinton and Salakhutdinov, 2006; Sutskever et al., 2013; Ioffe and Szegedy, 2015; He et al., 2015; Ciresan et al., 2010), the paradigm of "deep"

networks (having many layers) has come to dominate.

Regardless of depth, the dimensionality of each intermediate representation (the "layer sizes") must be chosen. While increasing the dimensionality may increase modeling power, it also exacerbates overfitting and incurs a greater computational loss. However, Graphics Processing Units (GPUs) have recently been used to accelerate neural network models thanks to their ability to quickly perform large matrix multiplications. As a result, in modern practice "wide" layers are used in combination with regularization techniques (Bengio, 2012).

Nonlinearities A fundamental ingredient of neural networks is the elementwise nonlinearity applied at each layer; without them, the network would only be able to learn linear relationships between its input and output. Early work typically used sigmoidal nonlinearities such as the logistic or hyperbolic tangent functions (see table 2.1). However, these nonlinearities have a very small gradient for much of their domain which can cause vanishing gradient problems particularly for networks with many layers. Recently, the rectifier nonlinearity has become the standard for deep networks due to its computational efficiency and tendency to produce sparse representations (Jarrett et al., 2009; Glorot et al., 2011; Nair and Hinton, 2010). The rectifier nonlinearity has zero gradient for half of its domain, which can hinder training in some cases; as a result a number of variants have recently been proposed which have been shown empirically to produce better results in some cases (Maas et al., 2013; He et al., 2015).

Regularization Due to their tendency to overfit, some form of regularization is typically necessary to ensure that the optimized neural network's variance is not too high. A common regularizer which is used in many machine learning models is to include in the loss function a penalty on the norm of the model's parameters. In

neural networks, adding the L^2 penalty $\lambda \sum_i \theta_i^2$ where λ is a hyperparameter and θ_i are the model's parameters (e.g. individual entries of the weight matrices and bias vectors) is referred to as "weight decay" (Hanson and Pratt, 1989). This can prevent the weight matrix of a given layer from focusing too heavily (via a very large weight value) on a single unit of its input. A related term is $\lambda \sum_i |\theta_i|$ which effectively encourages parameter values to be zero (Bengio, 2012).

A completely different regularization method which has recently proven popular in neural networks is "dropout" (Hinton et al., 2012). In dropout, at each training iteration each unit of each layer is randomly set to zero with probability p. After training, the weights in each layer are scaled by $1/p$. Dropout intends to prevent the units of a given layer from being too heavily correlated with one another by randomly artificially removing connections. More simply, it provides a source of noise which prevents memorization of correspondences in the training set and has been shown empirically and theoretically (Wager et al., 2013) to be an effective regularizer.

In practice, the technique of "early stopping" is almost always used to avoid overfitting in neural network models (Prechelt, 2012). To utilize early stopping, during training the performance on a held-out "validation set" (over which the parameters of the network are not optimized by gradient descent) is computed. Overfitting is indicated by performance degrading on the validation set, which simulates real-world performance. As a result, early stopping effectively prevents overfitting by simply stopping training once the performance begins to degrade. The measure of performance and criteria for stopping my vary widely from task to task (and practitioner to practitioner) but the straightforwardness and effectiveness of this approach has led it to be nearly universally applied.

Of course, regularization can be rendered unnecessary when a huge amount of training data is available. This is commonly simulated by applying hand-designed re-

alistic deformations to the training data, which is referred to as "data augmentation". For example, applying elastic distortions to images can allow simple, unregularized neural network models to match the performance of more complex architectures (Ciresan et al., 2010). Data augmentation has also been explored in the domain of audio and music signals (McFee et al., 2015a; Schlüter and Grill, 2015).

Parameter Initialization Because the objective functions used for training neural networks tend to be highly non-convex, the way parameters are initialized prior to training can have a substantial effect on the solutions found after optimization (Sutskever et al., 2013). In fact, the recent success of neural network models was prompted by a new method for setting parameter values prior to supervised training called "unsupervised pretraining". Unsupervised pretraining starts by training an unsupervised model (e.g. a Restricted Boltzmann Machine (Hinton and Salakhutdinov, 2006; Hinton et al., 2006) or an autoencoder (Bengio et al., 2007; Erhan et al., 2009)) and then using the parameters from that model to initialize a supervised model. Ideally, the unsupervised model will take care of disentangling factors of variation and producing a more efficient representation of the input space. After initialization with values from the unsupervised model, training of the model proceeds in the normal supervised fashion. This can greatly accelerate learning, and is of particular use when there is a great deal more unlabeled data available than labeled data.

It has more recently been shown that unsupervised pretraining can be skipped (provided that there is a sufficient amount of labeled data) by carefully initializing the network's parameters. This typically involves sampling the values of the weight matrices from zero-mean Gaussian distributions whose variances are set so that the representation at each layer's output has unit variance. The actual variances used for the random variables depends on the input and output dimensionality as well as

the nonlinearity; (Glorot and Bengio, 2010) provides a derivation for linear networks, (He et al., 2015) for rectifier networks and (Mishkin and Matas, 2015) gives a generic empirical procedure for any network architecture. An alternative is to use orthogonal matrices (Saxe et al., 2013) or to use a sparse initialization so that a fixed number of entries are non-zero (Sutskever et al., 2013).

Input Normalization Most of the analysis performed on neural networks in order to determine how to facilitate training rely on the assumption that the feature dimensions of the input to the network have zero mean and unit variance. A necessary step in order to take advantage of many of these tricks is then to standardize the feature dimensions of the input, typically using statistics computed on the training set. An alternative which can also facilitate convergence is to whiten data using Principal Component Analysis before presenting it to the network (LeCun et al., 2012).

Model Architecture Beyond the simple feedforward network model we have focused on thus far, a variety of more specialized models have been proposed which intend to exploit different types of structure present in real-world data. For example, presenting sequential data one step at a time to a feedforward network assumes temporal independence, which is invalid for most real-world temporal data. Other types of data, such as images, exhibit local dependencies which are invariant to translation. Modeling this type of structure can be greatly facilitated by the important choice of model architecture. We give an overview of the two most relevant architectures for sequential data, recurrent and convolutional networks, in the next two sections.

2.1.3.3 Recurrent Networks

A natural extension to feedforward networks are *recurrent* connections, which feed the output of parts of the model back into their input. The resulting model can be considered a learnable function which at each time step produces an output based on its current input and previous output. This class of models, called recurrent networks or RNNs, are well-suited for sequential data due to their ability to model temporal dependencies.

Training of recurrent networks is typically accomplished using "backpropagation through time" (BPTT) (Werbos, 1990). BPTT is a simple extension of backpropagation where, given a sequence of length T, the recurrent network is "unrolled" over time and is treated as a feedforward network with T layers, each of which corresponds to a single time step. While BPTT provides a straightforward way to train recurrent networks, this approach can suffer dramatically from vanishing and exploding gradients for large values of T for the same reasons that networks with many layers do (Bengio et al., 1994; Hochreiter and Schmidhuber, 1997; Pascanu et al., 2013). The use of gating architectures (Hochreiter and Schmidhuber, 1997; Cho et al., 2014), sophisticated optimization techniques (Martens and Sutskever, 2011; Sutskever et al., 2013), gradient clipping (Pascanu et al., 2013; Graves, 2013), and/or careful initialization (Sutskever et al., 2013; Jaeger, 2012; Mikolov et al., 2014; Le et al., 2015) can help mitigate this issue and has facilitated the success of RNNs in a variety of fields (see e.g. (Graves, 2012) for an overview). However, these approaches don't *solve* the problem of vanishing and exploding gradients for recurrent networks, and as a result these models are in practice typically only applied in tasks where sequential dependencies span at most hundreds of time steps. Because the representations used for audio data are typically at least thousands of time steps in length, we will not use recurrent networks in this thesis, although we will use

them as a point of comparison and motivation in some places.

2.1.3.4 Convolutional Networks

For many types of data (represented as multi-dimensional arrays), one or more of the dimensions may have an explicit ordering. Most relevant to this thesis is sequential data, which always has a temporal dimension. Traversing this dimension corresponds to the passing of time; if one event occurred before another, it will appear first. Similarly, images have "width" and "height" dimensions, which correspond to left-right and top-bottom traversal. In many cases, the correlations along these dimensions are highly localized. For example, in a text document the probability that a given word appears in a certain place depends much more strongly on the words appearing around it than it does on words which appeared hundreds of words ago.

The "fully-connected" feedforward networks we have discussed so far cause the output of each layer to depend on each unit of the layer's input due to the fact that the weight matrix has an entry which connects each input unit to each output unit. This ignores the structure of data with an ordered dimension and may be inefficient when most of the dependencies are local. The *convolutional network* architecture is formulated specifically to take advantage of this common structure. As their name suggests, rather than utilizing an affine transformation using a fully-connected matrix, layers in these models convolve small filter "kernels" across their input. Networks with this structure were first proposed by Fukushima (Fukushima, 1980) and training them with backpropagation was popularized by LeCun et. al (LeCun et al., 1989). Recently, convolutional networks have provided breakthroughs in a variety of domains, including image classification (Krizhevsky et al., 2012; He et al., 2015; Szegedy et al., 2015), speech recognition (Abdel-Hamid et al., 2012; Sainath

et al., 2013), music information retrieval (Humphrey and Bello, 2012; Van den Oord et al., 2013; Schlüter and Böck, 2014; Ullrich et al., 2014; McFee et al., 2015a; Schlüter and Grill, 2015), and natural language processing (Kim, 2014; Zhang and LeCun, 2015).

In the two-dimensional case (as used in this thesis), a convolutional layer takes as input $x \in \mathbb{R}^{M \times I_x \times I_y}$ and outputs $y \in \mathbb{R}^{N \times J_x \times J_y}$ by computing

$$y[n, j_x, j_y] = \sigma \left(\sum_{m=1}^{M} \sum_{k_x=1}^{K_x} \sum_{k_y=1}^{K_y} w[n, m, k_x, k_y] x[m, j_x + k_x, j_y + k_y] + b[n] \right) \quad (2.36)$$

where $w \in \mathbb{R}^{N \times M \times K_x \times K_y}$ are the layer's filters (also referred to as kernels), σ is a nonlinearity, and $b \in \mathbb{R}^N$ are the layer's biases. This operation is visualized in figure 2.3. The first dimension of x and y is referred to as the "channels" and can be considered different views or representations of the data (e.g. R, G, and B channels in images or left and right channels in a stereo audio recording). The remaining dimensions are feature dimensions, each of which is assumed to have an inherent ordering which is exploited by the translation inherent in the convolution operation. The exact form of equation (2.36) depends on a variety of design choices, such as the convolution's stride, whether the convolution is zero-centered, whether biases are shared across input channels, whether a correlation or convolution is computed, and whether the input will be zero-padded. We forgo a complete enumeration of these differences here and instead refer to (Dumoulin and Visin, 2016), which provides a thorough discussion of the arithmetic involved in convolutional networks.

From this definition, we can observe a number of beneficial properties of using convolutional networks. First, the number of parameters per layer is potentially dramatically decreased: Following the common practice of "flattening" dimensions, a fully-connected layer with the same input and output dimensionality would have

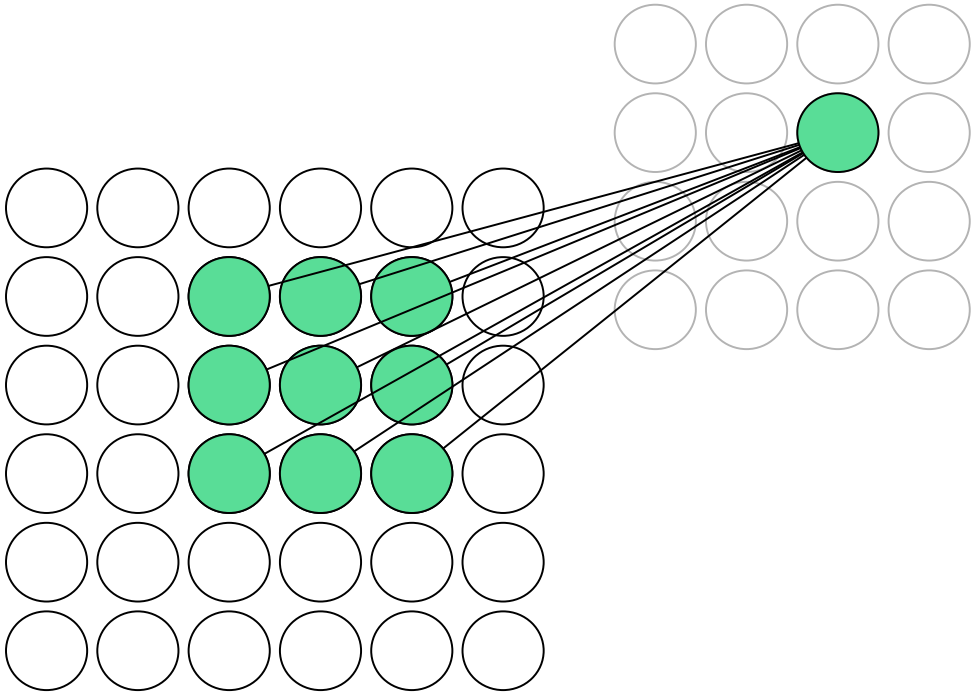

Figure 2.3: Visualization of the convolution operation defined in equation (2.36). The input and output are shown as 6×6 and 4×4 grids of units respectively. The output unit highlighted in green depends on the green input units via a 3×3 filter. Convolution involves evaluating this filtering operation at all possible positions in the input.

a weight matrix of shape $NJ_xJ_y \times I_xI_y$, compared with $N \times M \times K_x \times K_y$ with $K_x < J_x \leq I_x$ and $K_y < J_y \leq I_x$. This is commonly referred to as "parameter sharing", because the same set of weights are applied at different locations in the input rather than using a unique weight for each position. This property also gives the convolutional layer translational invariance, i.e. the ability to detect a certain pattern regardless of where it appears in the image. A final advantage to using a convolution operation is that the layer can naturally handle inputs whose feature dimensions vary in size – the kernels simply slide over the input regardless of its shape. This makes convolutional networks well-suited for sequential data whose

length varies.

An additional ingredient commonly found in convolutional networks are *pooling* layers, which provide additional translational invariance by aggregating their input over small regions. The most common pooling layer, and the only one used in this thesis, is the max-pooling layer, which outputs the maximum value within each input region. Given input $x \in \mathbb{R}^{M \times I_x \times I_y}$, a max-pooling layer with stride p produces $y \in \mathbb{R}^{M \times \frac{I_x}{p} \times \frac{I_y}{p}}$ by

$$y[m, j_x, j_y] = \max_{\substack{n_x \in \{1,\ldots,p\} \\ n_y \in \{1,\ldots,p\}}} (x[m, pj_x + n_x, pj_y + n_y]) \tag{2.37}$$

for $j_x \in 1, \ldots, \frac{I_x}{p}$ and $j_y \in 1, \ldots, \frac{I_y}{p}$. The value of $p = 2$ is almost uniformly used. The max-pooling layer can thus be seen as indicating whether a given feature is present or absent over a region of the input. Because pooling layers effectively downsample their input, they can be used to increase the "receptive field" (region of the input affecting a single unit of the output) of layers in networks consisting of many convolutional and pooling layers.

Finally, in convolutional networks it is common for the last few layers to be standard fully-connected layers. This gives the output of the network access to the entire input region. In this framework, the series of convolutional and pooling layers transform and downsample the input, providing appropriate invariances and often learning a hierarchy of higher and higher-level representations (Zeiler and Fergus, 2014). In domains where none of the dimensions of the input vary from sample to sample, all of the dimensions of the output of the convolutional portion of the network are flattened prior to being passed to the fully-connected layers. When one or more dimensions are variable in size, only the fixed-size dimensions are processed by the fully-connected layers. For example, in sequences where the sequence length

varies, the output of the convolutional layers for each time step are passed to the fully-connected layers independently.

2.1.4 Stochastic Optimization

In modern practice, the datasets used for training neural networks are typically huge, so optimizing their parameters over the entire training set is often inefficient and sometimes infeasible. The prevailing solution is to train over a tiny subset of the training set (called a "minibatch") which is sampled randomly at each iteration. This causes the objective to change stochastically at each iteration of optimization. Beyond practical limitations, using a stochastic objective has also been shown to help avoid stationary points because the objective surface is constantly changing (LeCun et al., 2012).

So far, we have only alluded to the fact that the gradient information produced by backpropagation is used to train a neural network. This is achieved using gradient descent methods, which specify a formula to adjust the network's parameters to minimize an objective. In many problems, it is beneficial to leverage second-order gradient information to estimate the local curvature of the objective function during optimization (for intuition as to why this is true, see section 2.1.4.1). However, the number of entries in the Hessian (matrix of second-order derivatives) is quadratic in the number of parameters, which makes computing it infeasible for modern neural network models which typically have millions of parameters. As a result, neural networks are typically trained using gradient descent methods which only utilize first-order gradient information.

A final difficulty in training neural networks is that changing different parameters can have substantially different effects on the objective value. For example, even in simple feedforward networks, adjusting a value in the final layer's bias vector may

induce a considerable change in typical outputs of the network while changing a value in the first layer's weight matrix may only produce a small effect in a limited number of cases. This problem can be exacerbated when different structures are used in different parts of the network, e.g. convolutional or recurrent layers. As a result, it can be beneficial to update different parameters by different amounts according to the extent to which they effect the objective value.

The stochastic objective arising from training on minibatches and the requirement of only utilizing first-order gradients prompts the use of a special class of gradient descent methods called stochastic optimization. Effective stochastic optimization techniques typically include mechanisms for smoothing the updates, which helps mitigate noise from stochasticity; for estimating second-order information, which improves convergence; and for adaptively updating different parameters by different amounts. In addition, most optimization methods have at least one hyperparameter (i.e. a model parameter not adjusted by gradient descent); good optimization techniques are able to be effective without being particularly sensitive to their hyperparameter values. In this section, we discuss some of the more popular stochastic optimization methods, particularly those which are used or referenced to in this thesis. Similar overviews are available in (Ruder, 2016; Bayer et al., 2015), and (Schaul et al., 2013) provides a comparison of many of these techniques on a variety of toy optimization problems.

In the following, we will use $\mathcal{L}$ to refer to the loss function being minimized. $\mathcal{L}$ is typically a function of the target output, the input, and the parameters being optimized; because all but the latter can be treated as a constant, we omit those terms in this section. In addition, because the parameters change from iteration to iteration we will use θ_t to denote the parameters of the model at iteration t. Finally, $\nabla\mathcal{L}(\theta_t)$ denotes the gradient of the loss function with respect to the parameters at

iteration t.

2.1.4.1 Stochastic Gradient Descent

Stochastic gradient descent (SGD) simply updates each parameter by subtracting the gradient of the loss with respect to the parameter, scaled by the "learning rate" hyperparameter η:

$$\theta_{t+1} = \theta_t - \eta \nabla \mathcal{L}(\theta_t) \tag{2.38}$$

SGD provides the fundamental ingredient on which other stochastic optimization methods are built. Provided that η is set correctly, SGD will decrease the loss function at each iteration because it is adjusting the parameters in the negative gradient direction. However, if η is too large, SGD can diverge (increase the loss function or oscillate between increasing and decreasing it); if it's too small, it will decrease the objective value very slowly (LeCun et al., 2012). This sensitivity to the setting of η is visualized in figure 2.4. In addition, when the loss surface is ill-conditioned (i.e. the curvature of the loss surface is dramatically different along different parameter axes), convergence can also be extremely slow because η must be set small enough so that it does not diverge along any parameter axis. Nevertheless, SGD can be effective when η is chosen correctly; a common heuristic is to choose the largest value of η which does not cause divergence by starting with a large η and decreasing it by a factor of 3 until optimization no longer diverges (Bengio, 2012).

2.1.4.2 Momentum

In SGD, the gradient $\nabla \mathcal{L}(\theta_t)$ often changes rapidly at each iteration due to the fact that the loss is being computed over different data. It can therefore be beneficial to smooth the updates at each iteration. This can be achieved by re-using the gradient value from the previous iteration, scaled by a "momentum" hyperparameter μ, as

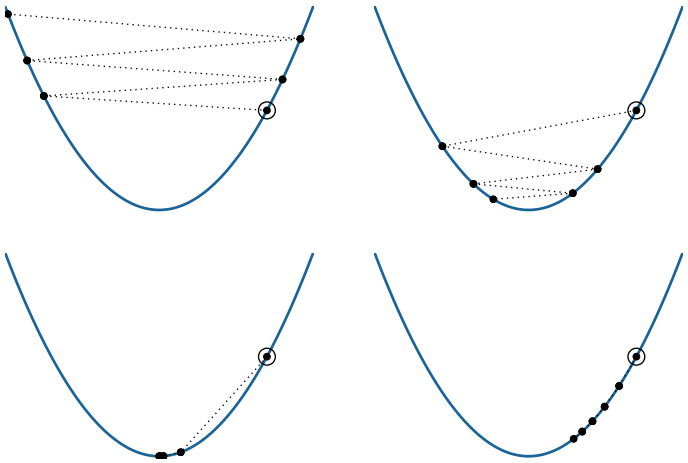

Figure 2.4: Visualization of the sensitivity of gradient descent to the learning rate η. The function being optimized is a simple quadratic function represented as a blue line, and each iteration of optimization is represented as a black dot with the initial point circled. In the top left, the learning rate is set too high and the optimization will diverge. In the top right, the learning rate is small enough that optimization will eventually converge, albeit slowly due to oscillations around the minimum. In the bottom left, the learning rate is set near-optimally and the minimum has effectively been reached in two iterations. In the bottom right, the learning rate is set too small and progress towards the minimum is slow.

follows (Polyak, 1964):

$$v_{t+1} = \mu v_t - \eta \nabla \mathcal{L}(\theta_t) \tag{2.39}$$

$$\theta_{t+1} = \theta_t + v_{t+1} \tag{2.40}$$

In addition, it has been argued that including the previous gradient step has the effect of approximating some second-order information about the gradient because directions with low curvature will tend to persist across iterations and accumulate, which can mitigate conditioning issues (Sutskever et al., 2013). In practice, μ is often set close to 1; this will cause the gradient updates to change slowly from

iteration to iteration. While early theory and intuition suggested that some of the benefits of momentum are lost in the stochastic setting (LeCun et al., 2012), it can be particularly beneficial in the early stages of learning before the loss function is close to any minima.

2.1.4.3 Nesterov's Accelerated Gradient

In Nesterov's Accelerated Gradient (NAG) (Nesterov, 1983), the gradient of the loss at each step is computed at $\theta_t + \mu v_t$ instead of θ_t. The update rules are then as follows:

$$v_{t+1} = \mu v_t - \eta \nabla \mathcal{L}(\theta_t + \mu v_t) \tag{2.41}$$

$$\theta_{t+1} = \theta_t + v_{t+1} \tag{2.42}$$

In momentum, the parameter update could be written $\theta_{t+1} = \theta_t + \mu v_t - \eta \nabla \mathcal{L}(\theta_t)$, so NAG effectively computes the gradient at the new parameter location but without considering the gradient term (i.e. using the first two terms of this update only). In practice, this causes NAG to behave more stably than regular momentum in many situations, and so it is often chosen over classical momentum. A more thorough analysis can be found in (Sutskever et al., 2013).

2.1.4.4 Adagrad

Adagrad (Duchi et al., 2011) effectively rescales the learning rate for each parameter according to the history of the gradients for that parameter. This is done by dividing

each term in $\nabla\mathcal{L}$ by the square root of the sum of squares of its historical gradient:

$$g_{t+1} = g_t + \nabla\mathcal{L}(\theta_t)^2 \tag{2.43}$$

$$\theta_{t+1} = \theta_t - \frac{\eta\nabla\mathcal{L}(\theta_t)}{\sqrt{g_{t+1}} + \epsilon} \tag{2.44}$$

where division is elementwise and ϵ is a small constant included for numerical stability. Rescaling in this way effectively lowers the learning rate for parameters which consistently have large gradient values, which can be particularly useful in ill-conditioned settings. It also effectively decreases the learning rate over time, because the sum of squares will continue to grow with the iteration. This can prove beneficial as the loss function becomes close to a minimum, where only small adjustments to the parameters are needed.

2.1.4.5 RMSProp

In its originally proposed form (Tieleman and Hinton, 2012), RMSProp is very similar to Adagrad. The only difference is that the g_t term is computed as an exponentially decaying average instead of an accumulated sum. This makes g_t an estimate of the second moment of $\nabla\mathcal{L}$ and prevents the learning rate from effectively shrinking over time. The name "RMSProp" comes from the fact that the update step is normalized by a decaying approximate root-mean-squared (RMS) of recent gradients. The update is as follows:

$$g_{t+1} = \gamma g_t + (1-\gamma)\nabla\mathcal{L}(\theta_t)^2 \tag{2.45}$$

$$\theta_{t+1} = \theta_t - \frac{\eta\nabla\mathcal{L}(\theta_t)}{\sqrt{g_{t+1}} + \epsilon} \tag{2.46}$$

In the original lecture slides where it was proposed, γ is set to .9, but it is most commonly treated as a variable hyperparameter. In (Dauphin et al., 2015), it is shown

that the $\sqrt{g_{t+1}}$ term approximates (in expectation) the diagonal of the absolute value of the Hessian matrix (assuming the update steps are $\mathcal{N}(0,1)$ distributed). It is also argued that the absolute value of the Hessian is better to use for non-convex problems which may have many saddle points.

Alternatively, (Graves, 2013) defines RMSProp with an additional first moment approximator m_t and a momentum term v_t:

$$m_{t+1} = \gamma m_t + (1-\gamma)\nabla\mathcal{L}(\theta_t) \tag{2.47}$$

$$g_{t+1} = \gamma g_t + (1-\gamma)\nabla\mathcal{L}(\theta_t)^2 \tag{2.48}$$

$$v_{t+1} = \mu v_t - \frac{\eta\nabla\mathcal{L}(\theta_t)}{\sqrt{g_{t+1} - m_{t+1}^2 + \epsilon}} \tag{2.49}$$

$$\theta_{t+1} = \theta_t + v_{t+1} \tag{2.50}$$

Including the first-order moment estimate in the denominator in this way causes the learning rate to be effectively normalized by the standard deviation of $\nabla\mathcal{L}$ in each direction.

2.1.4.6 Adam

Adam (Kingma and Ba, 2015) is somewhat similar to Adagrad and RMSProp in that it computes a decayed moving average of the gradient and squared gradient (first and second moment estimates) at each time step. It differs mainly in two ways: First, the separate moving average coefficients γ_1 and γ_2 are used for first and second moments respectively (equation (2.52) and equation (2.53)). Second, because the first and second moment estimates are initialized to zero, some bias-correction is used to counteract the resulting bias towards zero (equation (2.54) and equation (2.55)). Given hyperparameters γ_1, γ_2, λ, and η, and setting $m_0 = 0$ and $g_0 = 0$, the update

rule is as follows:

$$m_{t+1} = \gamma_1 m_t + (1 - \gamma_1)\nabla\mathcal{L}(\theta_t) \tag{2.51}$$

$$g_{t+1} = \gamma_2 g_t + (1 - \gamma_2)\nabla\mathcal{L}(\theta_t)^2 \tag{2.52}$$

$$\hat{m}_{t+1} = \frac{m_{t+1}}{1 - \gamma_1^{t+1}} \tag{2.53}$$

$$\hat{g}_{t+1} = \frac{g_{t+1}}{1 - \gamma_2^{t+1}} \tag{2.54}$$

$$\theta_{t+1} = \theta_t - \frac{\eta\hat{m}_{t+1}}{\sqrt{\hat{g}_{t+1}} + \epsilon} \tag{2.55}$$

The use of the estimated first and second moments ensure that, in most cases, the step size is $\approx \pm\eta$ and that in magnitude it is less than η. However, as θ_t approaches a true minimum, the uncertainty of the gradient will increase and the step size will decrease. It is also invariant to the scale of the gradients. These properties make it well-suited for neural network architectures, and as a result Adam has seen rapid adoption.

2.1.5 Bayesian Optimization

From the discussion above, it is clear that beyond the parameters of neural network models which are optimized by gradient descent (weight matrices, bias vectors, etc.), there are additional "hyperparameters" which govern specifics of the model architecture, loss function, and training. These hyperparameters are so called because they are not optimized through gradient descent with the rest of the parameters. Nevertheless, the success or failure of training a given model will often heavily depend on these hyperparameters being set appropriately.

This problem fits into the general framework of black-box optimization, where we would like to maximize an objective function which produces different values based

on different parameter settings. In this framework, we assume that we do not have a way to compute an analytical solution or have access to gradient information about the function and thus can only make decisions about what parameter settings to try based on the history of function evaluation results for different settings. For example, the function may be a mapping between hyperparameters governing the structure of a neural network model (number and width of layers, weight initialization scheme, etc.), its loss function (which loss function to use, whether to include regularization), and training procedure (which stochastic optimization technique to use, each of which has its own hyperparameters such as the learning rate) and the objective value could be the trained model's accuracy on a held-out validation set. Alternatively, one could consider the process of baking a cake in this framework, where the parameters are the amount and quantity of each ingredient as well as the baking temperature and time and the objective value is how good the cake tastes.

In practice, this type of problem is often tackled by adjusting parameters by hand based on a series of trials of different settings. This approach can be extremely time consuming and expensive and may produce suboptimal results, so various approaches for automating this process have been proposed. A popular choice is "grid search", which first quantizes any continuously-valued parameters and then exhaustively tries all combinations of all settings. This approach is only feasible when there are few parameters and possible settings due to the exponential number of trials which must be run. In addition, it has been shown that grid search is often less efficient than simply randomly trying different parameter values, particularly when the setting of one or more parameters does not have a strong effect on the objective value (Bergstra and Bengio, 2012). While grid search and random search can be effective, neither utilize the results of previous trials to inform subsequent parameter settings.

A valuable alternative is Bayesian optimization, which utilizes all of the information available in previous trials to produce a probabilistic model of the black-box function being optimized. This requires more computation than methods which only use local approximations of the function, but can produce good solutions to highly non-convex functions after making only a small number of function evaluations. This is particularly valuable when the function itself is expensive to evaluate, as is the case when training machine learning models which take hours, days, or weeks (or when baking cakes, due to the price of ingredients). The use of a probabilistic model of the black-box function being optimized both allows for candidate parameter settings to be chosen in a principled way and for prior information to be integrated into the optimization process. We utilize Bayesian optimization techniques throughout this thesis for black-box optimization of different algorithms and systems. This section provides a brief overview of Bayesian optimization; for more thorough treatments see (Brochu et al., 2010; Snoek et al., 2012). A visualization of Bayesian optimization is shown in figure 2.5.

2.1.5.1 Gaussian Processes

Utilizing Bayesian optimization requires first choosing a prior over functions which expresses beliefs about the behavior of the function being optimized. A common choice are Gaussian processes, which associate every point in the (continuous) parameter space with a normally distributed random variable. An alternative definition is that every finite collection of points in the parameter space induces a multivariate Gaussian distribution. In the setting of Bayesian optimization, this corresponds to assuming that the objective values are characterized by Gaussian distributions.

More formally, given parameter settings $\theta \in \Theta$ where Θ is the parameter space,

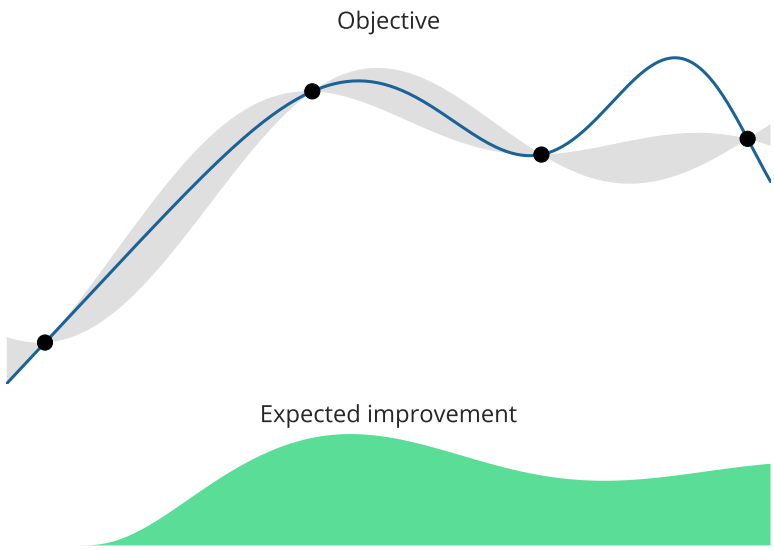

Figure 2.5: Visualization of Bayesian optimization of a synthetic one-dimensional objective function, shown as a blue line. Points at which the objective has been evaluated are shown as black dots. The gray shaded area shows the posterior estimate of the function, modeled by a Gaussian process with a squared exponential kernel. The acquisition function, calculated using expected improvement, is shown in green. Due to the locations of the objective evaluations, the posterior estimate has failed to account for the true maximum at the right of the plot.

a Gaussian process is characterized by a mean function $m : \Theta \to \mathbb{R}$ and covariance function $k : \Theta \times \Theta \to \mathbb{R}$. The mean function is (without loss of generality) typically assumed to be zero, while the choice of the covariance function allows strong assumptions to be made about the function being estimated. A common choice is the squared exponential kernel

$$k(\theta_i, \theta_j) = \exp\left(-\frac{1}{2\omega^2}\|\theta_i - \theta_j\|^2\right) \tag{2.56}$$

where ω is a hyperparameter controlling the width of the kernel. (Snoek et al., 2012) argues that this kernel is too smooth for practical optimization problems and

recommends the use of the Matérn 5/2 kernel instead. In general, the covariance function should be defined so that it approaches 1 as its arguments get closer together and 0 as they get farther apart. The hyperparameters of the covariance function function can either be set according to prior beliefs about the function being modeled, or can be estimated by optimizing the marginal likelihood of the observed function evaluations under the Gaussian process.

Once mean and covariance functions have been chosen, the Gaussian process can be fit to training data and a posterior can be obtained by the following process: Denoting the t parameter settings and observed objective values as $\theta_T = \{\theta_1, \ldots, \theta_t\}$ and $y_T = \{y_1, \ldots, y_t\}$ respectively and an unseen evaluation of the objective function y_n, the joint marginal likelihood of y_n and the observed objective values is given by

$$p(y_T, y_n) = \mathcal{N}\left(0, \begin{bmatrix} K_{T\times T} & K_{n\times T}^\top \\ K_{n\times T} & k(\theta_n, \theta_n) \end{bmatrix}\right) \tag{2.57}$$

where

$$K_{T\times T} = \begin{bmatrix} k(\theta_1, \theta_1) & \cdots & k(\theta_1, \theta_t) \\ \vdots & \ddots & \vdots \\ k(\theta_t, \theta_1) & \cdots & k(\theta_t, \theta_t) \end{bmatrix} \in \mathbb{R}^{t\times t} \tag{2.58}$$

$$K_{n\times T} = \begin{bmatrix} k(\theta_n, \theta_1) & \cdots & k(\theta_n, \theta_t) \end{bmatrix} \in \mathbb{R}^{1\times t} \tag{2.59}$$

$$\tag{2.60}$$

This can be used to obtain the predictive distribution (see (Rasmussen, 2006) for a derivation)

$$p(y_n | y_T, \theta_T, \theta_n) = \mathcal{N}(\mu_T(\theta_n), \sigma_T^2(\theta_n)) \tag{2.61}$$

where

$$\mu_T(\theta_n) = K_{n\times T} K_{T\times T}^{-1} \begin{bmatrix} y_1 \\ \vdots \\ y_t \end{bmatrix} \tag{2.62}$$

$$\sigma_T^2(\theta_n) = k(\theta_n, \theta_n) - K_{n\times T} K_{T\times T}^{-1} K_{n\times T}^{\top} \tag{2.63}$$

Note that the covariance matrix of the joint marginal likelihood of y_n and y_T will have 1 all along its diagonal because any point is perfectly correlated with itself. This is an invalid assumption in the presence of noise, so a common modification to include a term σ_{noise}^2 for the variance of the noise in the covariance, which yields the predictive distribution

$$p(y_n|y_T, \theta_T, \theta_n) = \mathcal{N}(\mu_T(\theta_n), \sigma_T^2(\theta_n) + \sigma_{noise}^2) \tag{2.64}$$

$$\mu_T(\theta_n) = K_{n\times T}(K_{T\times T} + \sigma_{noise}^2 I)^{-1} \begin{bmatrix} y_1 \\ \vdots \\ y_t \end{bmatrix} \tag{2.65}$$

$$\sigma_T^2(\theta_n) = k(\theta_n, \theta_n) - K_{n\times T}(K_{T\times T} + \sigma_{noise}^2 I)^{-1} K_{n\times T}^{\top} \tag{2.66}$$

2.1.5.2 Acquisition Functions

We now have a recipe for predicting the mean and variance of the objective function's value at any point based on observed values. In order to perform optimization, we also need a principled approach for selecting new parameter settings to evaluate the function at, both in hopes of achieving higher objective values and for obtaining a better estimate of the objective function. This is achieved through acquisition functions, which utilize the predictive distribution to construct a function whose

maximum corresponds to the point at which the objective function should be evaluated at next. Acquisition functions can be considered a proxy function which can be optimized to estimate the objective function's maximum value under certain assumptions. Importantly, sampling values from acquisition functions is potentially much cheaper than evaluating the objective function at new parameter values, which is one of the core reasons that Bayesian optimization is beneficial. To facilitate optimization, acquisition functions should appropriately balance exploration and exploitation, i.e. the trade-off between reducing uncertainty and finding the absolute best value.

Probability of Improvement An intuitive choice is to maximize the probability that a given point is larger than the best observed value, which we denote $y_{best} = \max(y_T)$. This probability can be computed as

$$\mathrm{PI}(\theta) = \Phi\left(\frac{\mu_T(\theta) - y_{best}}{\sigma_T(\theta)}\right) \tag{2.67}$$

where Φ is the normal cumulative distribution function. In the case that $\sigma_T(\theta) = 0$, $\mathrm{PI}(\theta)$ is set to 0. This strategy will always choose the θ which has the highest probability of yielding a larger objective value regardless of whether this value will provide more information about the function, i.e. it is completely exploitation-based. A simple way to mitigate this is to add an exploration trade-off parameter ξ to instead compute

$$\mathrm{PI}(\theta) = \Phi\left(\frac{\mu_T(\theta) - y_{best} + \xi}{\sigma_T(\theta)}\right) \tag{2.68}$$

Expected Improvement Instead of simply choosing the point which is most likely to yield an improvement, we can also consider the expected magnitude of the improvement. This can be encapsulated using an "improvement function" which is

positive when the objective value for the predicted value of θ is larger than y_{best}. Maximizing the expectation of this improvement function has a closed-form solution, yielding

$$\mathrm{EI}(\theta) = (\mu_T(\theta) - y_{best})\Phi\left(\frac{\mu_T(\theta) - y_{best}}{\sigma_T(\theta)}\right) + \sigma(\theta)\phi\left(\frac{\mu_T(\theta) - y_{best}}{\sigma_T(\theta)}\right) \tag{2.69}$$

where ϕ is the normal probability density function. (Snoek et al., 2012) claims that Expected Improvement yielded the best performance among acquisition functions for hyperparameter optimization of machine learning systems.

Upper Confidence Bound An alternative approach, which is also conceptually simple, is to maximize the upper confidence bound

$$\mathrm{UCB}(\theta) = \mu_T(\theta) + \kappa\sigma_T(\theta) \tag{2.70}$$

$\kappa \geq 0$ is a hyperparameter which scales the confidence limits and therefore balances exploration and exploitation.

2.2 Digital Signal Processing

The field of signal processing encompasses the analysis, modification, and generation of "signals". Signals are broadly defined as sequential data, i.e. data where at least one dimension has an implicit ordering (for some examples, see figure 1.1). A digital signal is a sequence of numbers or vectors where each number represents the amplitude of the signal in a given dimension at a given point in time. In this setting, signals can be processed computationally, which allows for the development of algorithms to automatically manipulate, generate, and extract information from signals.

2.2.1 Audio Signals and Psychoacoustics

In this thesis, we will mainly deal with audio signals. An example audio signal, consisting of a series of notes played on an acoustic guitar, can be seen in figure 2.6. In the physical world, an audio signal is a time-varying change in air pressure called a sound wave. We "hear" sounds when our ear converts these sound waves into neural stimuli. The exact manner in which we perceive a sound wave is influenced by a wide variety of factors, including our sensory system and the brain's psychological response. The field of psychoacoustics attempts to measure and codify these factors, and the results can provide useful guidelines about the representation and processing of audio signals. In this section, we present a small overview of some important psychoacoustic principles; a more thorough treatment can be found in (Cook, 1999; Bosi and Goldberg, 2012).

Three of the most important psychoacoustic characteristics of audio signals are the pitch, loudness, and timbre.

Pitch The pitch of a signal mainly depends on its periodicity (e.g. how often the signal repeats itself); if a signal is has no periodic components, it might not be perceived as having a pitch. The longest periodicity, or the slowest vibration, is referred to as the fundamental frequency f_0 of a sound. Generally, we can perceive frequencies from 20 Hz to 20,000 Hz, although these limits depend on a variety of factors including the number of independent frequencies in a signal and the healthiness of the listener's auditory system. Below 20 Hz, we begin to hear the periodicities of an audio signal as distinct events rather than as a "pitch".

An additional important characteristic of pitch perception is that it is roughly logarithmic, i.e. we perceive the "distance" between signals with fundamental frequencies f_0 and $2f_0$ to be about the same as $2f_0$ and $4f_0$. This may be one reason

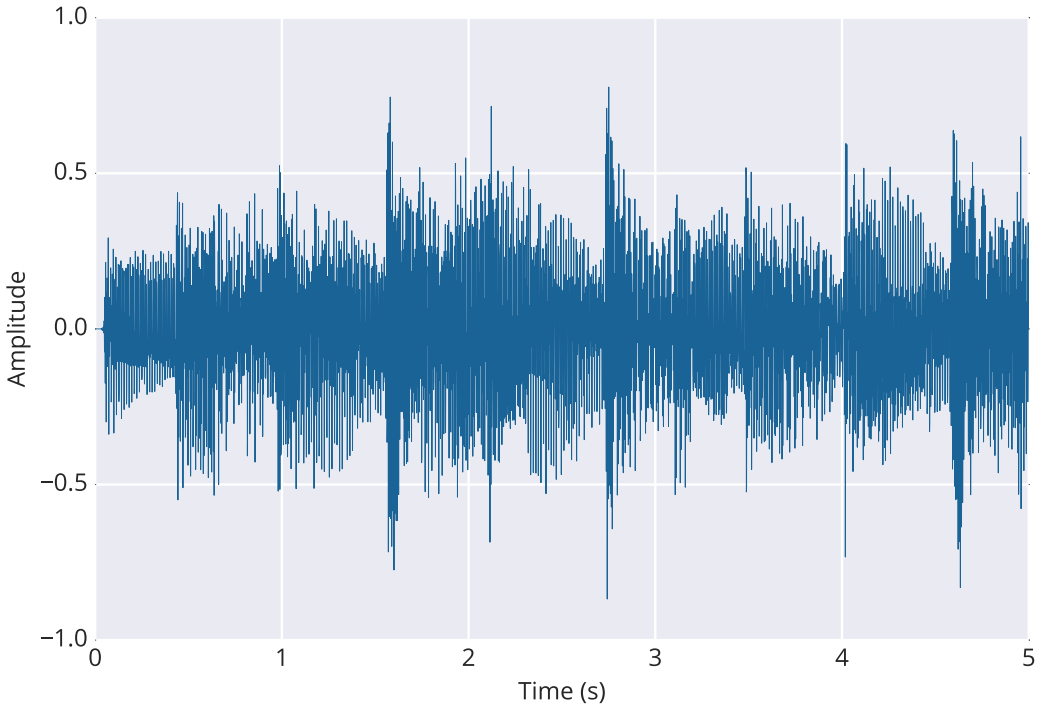

Figure 2.6: Audio signal of a few notes played on an acoustic guitar. The peaks of amplitude correspond to each individual note being plucked.

that the spacing between pitches (called semitones) in Western music is defined on a logarithmic scale. In addition, our inner ear has different regions which are sensitive to different ranges of frequencies, which are also spaced roughly on a logarithmic scale. This helps our auditory system distinguish between sources sounding simultaneously which differ in pitch. Whether or not a given signal is perceived as having multiple pitches or sources (e.g. in the case of multiple notes being played simultaneously on a single musical instrument) depends both on the characteristics of the signal and on psychological factors.

Loudness Loudness corresponds to how "strong" a sound is perceived as being and most directly depends on the signal's sound pressure level (SPL). The SPL is

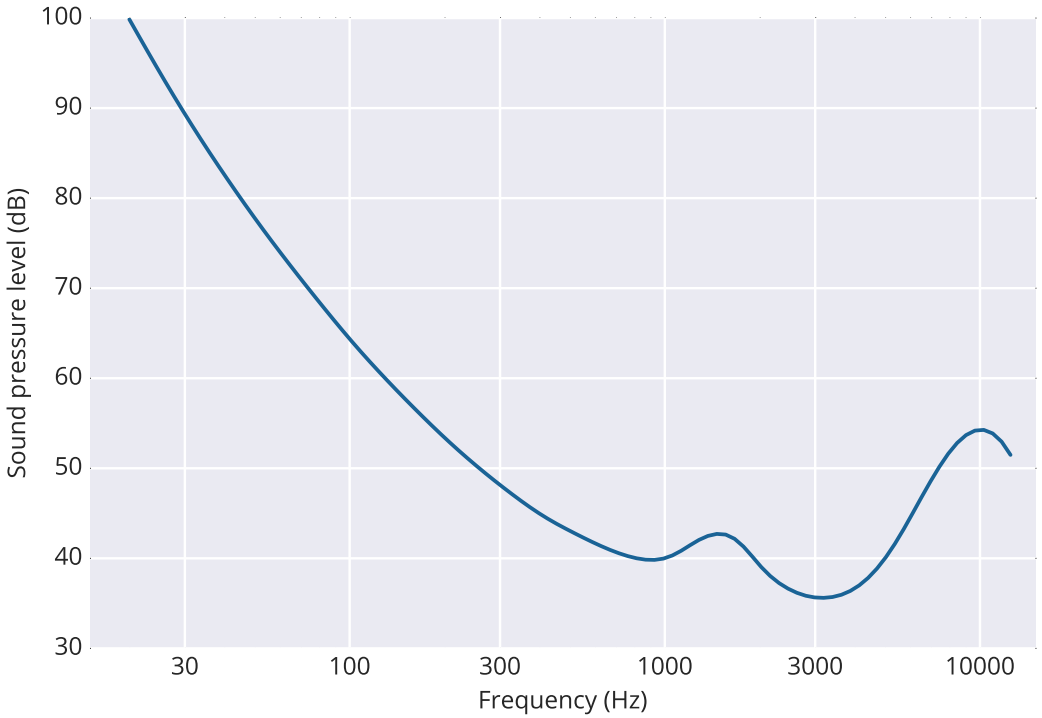

Figure 2.7: Equal loudness curve from the ISO 226 specification, along which all frequencies are perceived to have approximately the same loudness. These curves reveal important characteristics about our sensitivity to different frequencies; for example, in order for a low-frequency sound to be perceived loudly, they must have a relatively high sound pressure level.

computed as the logarithm of the ratio of a signal's pressure and a reference level of 20 microPascals and is measured in decibels (dB). As a result, perception of loudness is closer to logarithmic than linear in amplitude. Beyond the SPL, perception of loudness is also heavily frequency-dependent; for example, a 20 dB SPL signal with a fundamental frequency 1 kHz will be perceived to have the same loudness as a 100 Hz signal at about 45 dB SPL. This relationship is typically characterized by equal loudness curves, which relate the required sound pressure level for each frequency to be perceived as the same loudness. An example curve can be seen in figure 2.7.

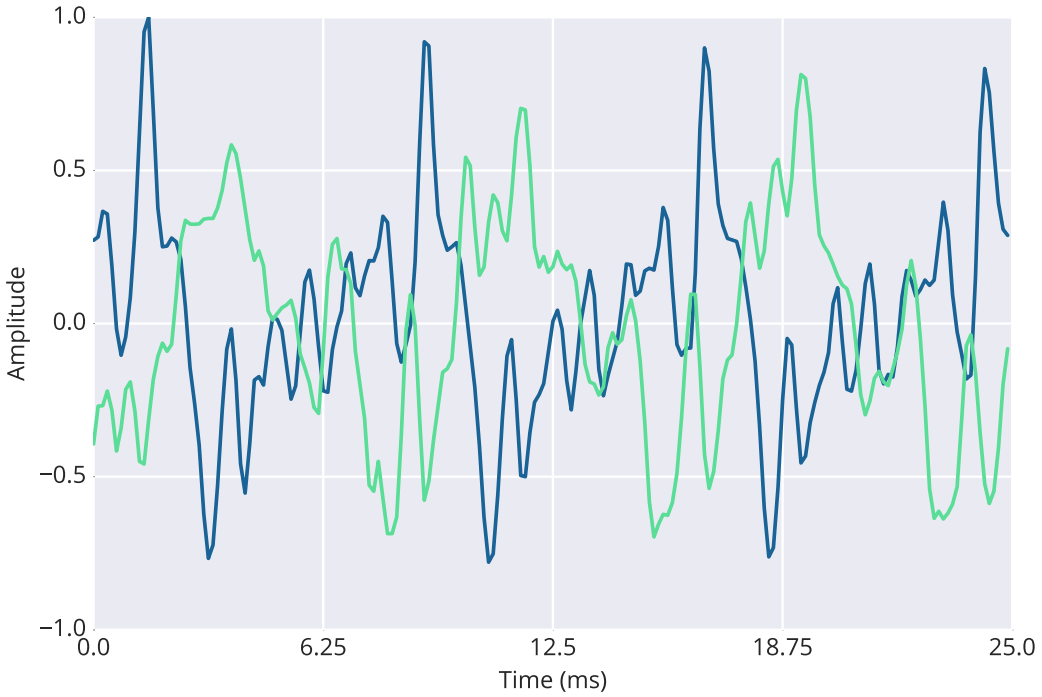

Figure 2.8: Waveforms of a few periods of the same note being played on a piano (blue) and a viola (green) at the same loudness. These two signals will be perceived as having the same pitch and loudness, but are distinguishable due to differences in the distribution and change over time of the frequencies present.

Timbre Even when pitch and loudness are matched, there is an unlimited possible variety of sound events. The remaining characteristics of a sound which cause it to be perceived differently from other sounds are referred to using the ambiguous catch-all term "timbre". The distinctions between sounds generally depend on their energy distribution in time and frequency. Being sensitive to these factors can help us distinguish between the voices of different people and notes played on different musical instruments. Figure 2.8 shows waveforms of a piano and a viola playing the same note at the same volume; despite having the same pitch and loudness, these sounds are easily distinguishable.

2.2.1.1 Sampling Frequency

Typically, the temporal spacing between subsequent elements in a digital signal is fixed and is determined by the signal's sampling frequency f_s. This is in contrast to continuous-time signals encountered in the real world, such as sonic waves, which can be considered functions of a continuous timebase. A fundamental relationship between the content of signals and their sampling frequency is codified in the Sampling Theorem. This theorem states that any continuous-time signal can be perfectly reconstructed from discrete-time samples at sampling frequency f_s as long as it has no frequency content above $f_s/2$.

Due to our hearing range, in audio we are mostly interested in reproducing signals up to 20,000 Hz. Following the sampling theorem, this implies that in order to perfectly perceptually reconstruct audio signals, we should utilize a sampling rate of at least 40,000 Hz. In practice, a common sampling rate for audio signals is 44,100 Hz, which is utilized on compact discs and other recorded audio formats. Because our sensitivity to high frequencies is limited, for analysis purposes it is common to use a lower sampling rate. Throughout this thesis, we will utilize a sampling rate of 22,050 Hz.

2.2.2 Time-Frequency Analysis

In many cases, it is useful to decompose a signal into the combination of simpler signals. An extremely common method is the Fourier Transform, which is capable of representing any signal as a sum of sinusoids. This representation is well-suited for signals from the natural world which are produced by vibrating objects such as musical instruments. In addition, as mentioned above, our inner ear performs a similar decomposition because its different regions vary in their sensitivity to different

frequencies. (Smith, 2011, 2007) give a comprehensive discussion of time-frequency analysis of audio signals.

For a discrete-time length-N single-dimensional signal x, the Discrete Fourier Transform (DFT) is computed as

$$X[k] = \sum_{n=1}^{N} x[n] e^{-2\pi\jmath(k-1)(n-1)/N}, k \in \{1, \ldots, N\} \tag{2.71}$$

The resulting complex values $X[k]$ are called the Fourier coefficients of x, and the collection of coefficients X is called the Fourier spectrum (or, simply, the spectrum) of x. From its definition, it is clear that the DFT can be computed in $\mathcal{O}(N^2)$ time; in practice, the DFT is computed using the Fast Fourier Transform (FFT) algorithm, which runs in $\mathcal{O}(N \log(N))$ time.

For real-valued signals (the only kind encountered in this thesis) where N is even, the Fourier coefficients are Hermitian symmetric so that

$$X[k] = \overline{X[N-k+2]}, k \in \{2, \ldots, N/2+1\} \tag{2.72}$$

For this reason, the DFT of real-valued signals of even length N is typically only computed for $k \in \{1, \ldots, N/2+1\}$. If the real-valued input signal has sampling rate f_s, then the k^{th} Fourier coefficient (also called a "frequency bin") corresponds to a frequency of

$$\frac{(k-1)f_s}{N}, k \in \{1, \ldots, N/2+1\} \tag{2.73}$$

The DFT provides us with a decomposition of a signal in the form of complex-valued coefficients (i.e. magnitude and phase) for sinusoids of varying frequency. However, for most audio signals, the human auditory system is much more sensitive to the *magnitude* of different frequencies than it is to the phase of these components.

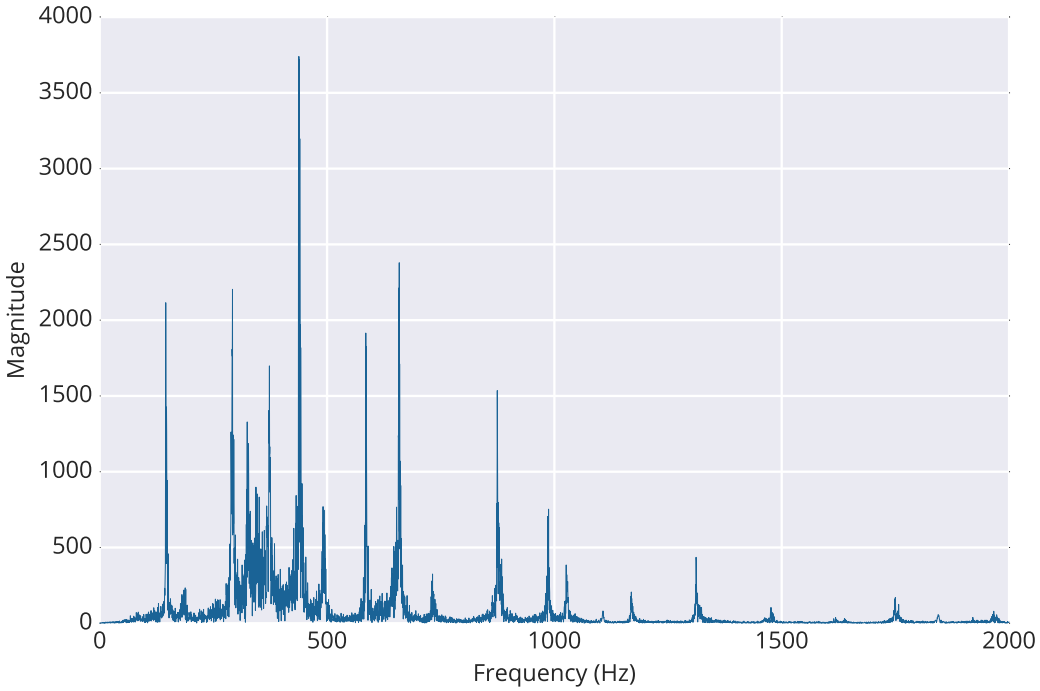

Figure 2.9: Magnitude spectrum of the audio recording of an acoustic guitar shown in figure 2.6. The tall peaks correspond to the dominant frequencies; for example, the largest peak around 440 Hz corresponds to an A note which is played frequently and loudly in the recording. For clarity, only the portion of the spectrum corresponding to frequencies up to 2000 Hz is shown.

For this reason, for analysis tasks the phase of the Fourier coefficients $X[k]$ is typically discarded and the magnitude $|X[k]|$ is used instead. As an example, the magnitude spectrum of the audio signal shown in figure 2.6 is shown in figure 2.9.

2.2.2.1 The Short-Time Fourier Transform

In many cases, we are most interested in the way the frequency content of a signal changes over time. The DFT as defined above can only provide a global summary in the form of a spectrum computed across the entire duration of a signal. This prompts the use of the Short-Time Fourier Transform (STFT), which computes a

series of spectra over short, potentially overlapping "frames" of the signal $x \in \mathbb{R}^N$:

$$S[n,k] = \sum_{m=1}^{M} w[m]x[m+nR]e^{-2\pi\jmath(k-1)(m-1)/M} \tag{2.74}$$

$$k \in \{1,\ldots,M\} \tag{2.75}$$

$$n \in \left\{1,\ldots,\left\lceil\frac{N-M+1}{R}\right\rceil\right\} \tag{2.76}$$

where $w \in \mathbb{R}^M$ is a "window" function and R is the "hop" size which determines how much (if at all) subsequent frames overlap. The resulting STFT S will consist of spectra with M Fourier coefficients.

The STFT is useful for audio signals because we are generally interested in how the audio content changes over time. For example, in music analysis we might be interested in where notes played in a recording start and stop. This information is hidden in the phase of the Fourier transform of an entire recording, but the individual spectra which encompass the start and end of notes retain this information in a localized manner. When the phase in the subsequent spectra in an STFT is discarded leaving only the magnitude, the resulting time-frequency representation is commonly called a spectrogram. The magnitude spectrogram of the guitar recording from figure 2.6 is shown in figure 2.10.

Important considerations when computing the STFT include the frame length M, the window function w, and the hop size R.

Frame Length Increasing the frame length M will result in higher-resolution spectra due to the fact that the frequencies corresponding to each bin are spaced according to equation (2.73). However, it will also result in lower time resolution, due to the fact that if multiple events happen within a single frame then it is impossible to determine their ordering by inspecting the magnitude spectrum. Choosing M is

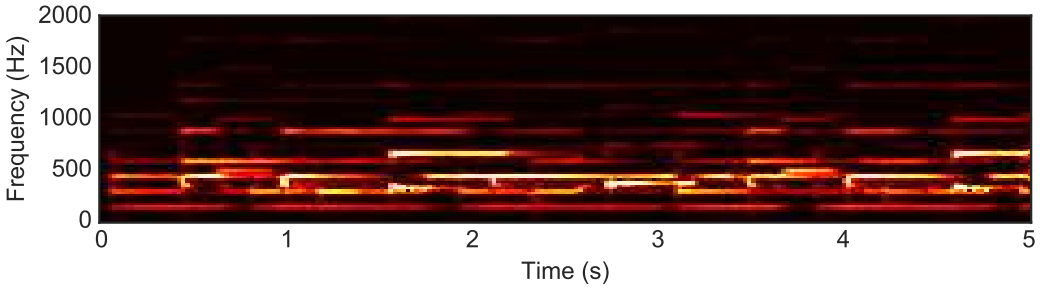

Figure 2.10: Magnitude spectrogram of the audio recording of an acoustic guitar shown in figure 2.6. As in figure 2.9, we only display frequencies up to 2,000 Hz. The horizontal tracks correspond to different notes which were played.

therefore a trade-off of time and frequency resolution, i.e. we would like to be able to resolve differences of frequency and the timing of events which are relevant to the task. A more practical consideration is that the FFT is optimized for frame lengths which are a power of two. Based on these constraints, a common choice is to utilize a frame size of 2048 samples at a 22,050 Hz sampling rate, which corresponds to 92.8 milliseconds with a 10.8 Hz resolution.

Window Function The choice of window function is generally a trade-off between the resolution and the sensitivity of the individual spectra. For example, given a signal which consists of a pure tone (i.e. a sinusoid) at some frequency f_0, using a windowing function with low resolution would result in spectral energy being spread amongst the frequency bins around the one corresponding to f_0. On the other hand, using a window function with low sensitivity will result in the inability to detect low-amplitude signals in the presence of noise or other high-amplitude signals. The rectangular window

$$w[m] = 1, m \in \{1, \ldots, M\} \tag{2.77}$$

has the highest-possible resolution, but has poor sensitivity due to "spectral leakage". Spectral leakage manifests itself as periodic "side lobes" in the frequency domain which decay away from each peak in the spectrum. For spectra with high-energy bins, this leakage can prevent the accurate detection of other peaks, which decreases the resulting sensitivity.

Leakage is caused by truncating each frame immediately to zero at the window edges. A common method for mitigating this is to use a window which tapers to zero. The Hann window

$$w[m] = \frac{1}{2}\left(1 - \cos\left(\frac{2\pi m}{M}\right)\right), m \in \{1, \ldots, M\} \tag{2.78}$$

provides a good trade-off between resolution and sensitivity; it has half of the resolution of the rectangular window but decreases side-lobe level and decay by a few orders of magnitude. As a result, we will use the Hann window throughout this thesis.

Hop Size Typically, there is some overlap between subsequent frames in an STFT. This is a necessity when using windows which taper to zero in order to avoid loss of information. For the Hann window in particular, in order to retain all information without distortion the hop size is typically set no larger than half of the frame size. Using more overlap provides smooth interpolation of temporal variations in the STFT, an easy way to achieve finer temporal resolution at higher computational cost. We therefore utilize a hop size of 1024 samples throughout this thesis which corresponds to 46.4 milliseconds at 22,050 Hz; this temporal resolution captures most of the perceptual significance of a sound's variation in time.

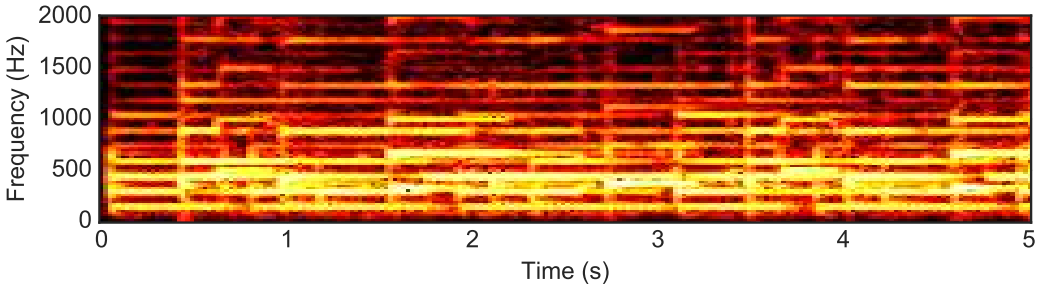

Figure 2.11: Log-magnitude spectrogram of the audio recording of an acoustic guitar shown in figure 2.6, for frequencies up to 2,000 Hz.

2.2.2.2 Log-Magnitude

As discussed in section 2.2.1, our sensitivity to loudness is closer to logarithmic than linear in amplitude. The magnitude spectrograms we have studied so far report the amplitude of different frequencies across time. We can make this representation more similar to our perception by using log-magnitude spectra, i.e. computing the logarithm of the magnitude in each frequency bin so that, for example, a doubling in amplitude corresponds to the same-size "step" regardless of initial level. While sound pressure level is defined on a specific scale based on a reference SPL of 0 dB corresponding to the quietest perceptible sound, it is unimportant for the scale to be calibrated in analysis tasks which are scale invariant.

A log-magnitude version of the spectrogram from figure 2.10 is shown in figure 2.11. Note that this representation shows much more energy in higher frequencies. These higher-frequency signals are the harmonics of the notes played, which were much less apparent in figure 2.10 where notes appeared almost as pure tones. These harmonics are an important factor in our ability to distinguish different sounds, and so a perceptually accurate representation must reflect their perceived relevance.

2.2.2.3 Constant-Q Transforms

An additional way in which the spectrograms we have discussed so far do not match perception is that they are linear in frequency (as defined by equation (2.73)). This is in contrast to our roughly logarithmic perception of pitch, as well as the spacing of semitones in the common Western music scale. As a result, it can be beneficial to use a time-frequency representation which is also logarithmic in frequency. A common choice is a constant-Q transform (CQT), which has frequency bins which are logarithmically spaced and all have equal center frequencies-to-bandwidth ratios (Brown, 1991). Fortunately, like the DFT, fast algorithms exist for computing the CQT (Brown and Puckette, 1992; Schörkhuber and Klapuri, 2010).

In this thesis, we will utilize the transform proposed by (Schörkhuber and Klapuri, 2010), which is defined as follows:

$$S[n,k] = \frac{1}{M_k} \sum_{m=1}^{M_k} w_k[m] x[m - M_k/2 + nR] e^{2\pi \jmath m f_k / f_s} \tag{2.79}$$

where f_k is the center frequency of each of the k bins, R is the hop size, $w_k \in \mathbb{R}^{M_k}$ is a window function, f_s is the sampling frequency, and

$$M_k = \frac{f_s}{f_k(\sqrt[B]{2} - 1)} \tag{2.80}$$

where B is the number of frequency bins per octave. For music signals, it is common to set B to 12 because there are 12 semitones per octave in the Western music scale and to set f_k to a sequence of subsequent semitone frequencies. Note that there is a different window size for each frequency bin; this is what allows the center frequency-to-bandwidth ratio to remain constant across bins. The choice of w_k depends on the desired time-frequency resolution trade-off. In this thesis, we set w_k

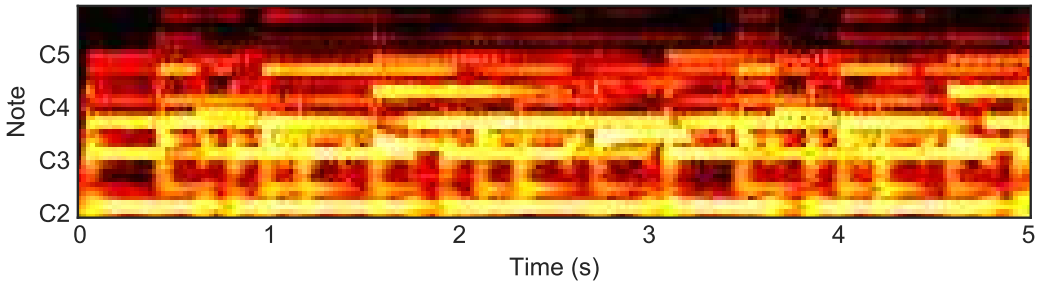

Figure 2.12: Log-magnitude constant-Q spectrogram of the audio recording of an acoustic guitar shown in figure 2.6. The constant-Q transforms are computed over logarithmically-spaced bins from C2 to B5 (65.4 to 987.8 Hz), with 12 bins per octave.

to the length-M_k Hann window, which provides a good balance for analysis tasks.

An important consideration when computing this formulation of the constant-Q transform is the hop size. If the hop size R is larger than $\min_k(M_k)$ (the window size of the highest frequency bin), it is possible that a portion of the signal will not be analyzed, and furthermore if $R > \min_k(M_k)/2$ then some distortion may result. The simplest way to mitigate this is to set $R < \min_k(M_k)/2$. However, if $\min_k(M_k)$ is by necessity very small, this can result in excessive oversampling. An alternative, advocated by (Schörkhuber and Klapuri, 2010), is to compute the high-frequency CQT bins at multiple offsets within each window and aggregate the results.

As with the DFT, the CQT produces complex-valued coefficients, and a more more perceptually accurate transform can be obtained by computing a log-magnitude CQT. For comparison, the log-magnitude CQT of the audio signal in figure 2.6 is shown in figure 2.12.

2.2.3 Dynamic Time Warping

A common task in signal processing is measuring the similarity between two sequences of feature vectors. For example, given a collection of recorded speech utterances with

word labels, we could perform rudimentary speech recognition by computing the similarity between a query utterance and all database entries and choosing the word corresponding to the most similar entry. In many domains, time-varying differences in phase can complicate direct comparison of sequences. For word utterances, this is realized by different speakers speaking different parts of words more quickly or slowly. As a result, it is often effective to use a warping similarity measure which allows for non-co-occuring sequence elements to be compared.

One such measure which will see extensive use in this thesis is dynamic time warping (DTW). First proposed for in our exemplary domain of comparing speech utterances (Sakoe and Chiba, 1978), DTW uses dynamic programming to find a monotonic alignment such that the sum of a distance-like cost between aligned feature vectors is minimized. Because DTW produces an optimal alignment, it is also useful in tasks where the actual correspondence between sequence elements is required, as in the problem of audio-to-MIDI alignment explored in chapter 4. In this section, we give an overview of DTW; a more thorough treatment can be found in (Müller, 2007).

Suppose we are given two sequences of feature vectors $X \in \mathbb{R}^{M \times D}$ and $Y \in \mathbb{R}^{N \times D}$, where D is the feature dimensionality and M and N are the number of feature vectors in X and Y respectively. DTW produces two nondecreasing sequences $p, q \in \mathbb{N}^L$ which define the optimal alignment between X and Y, such that $p[i] = m, q[i] = n$ implies that the m^{th} feature vector in X should be aligned to the n^{th} in Y. Finding p and q involves globally solving the following minimization problem:

$$p, q = \arg\min_{p,q} \sum_{i=1}^{L} \|X[p[i]] - Y[q[i]]\|_2 \tag{2.81}$$

The constraint of monotonicity is enforced by only allowing certain "step patterns", the simplest and most common being

$$p[i+1] = p[i] + 1, q[i+1] = q[i] + 1 \tag{2.82}$$

$$\text{or } p[i+1] = p[i] + 1, q[i+1] = q[i] \tag{2.83}$$

$$\text{or } p[i+1] = p[i], q[i+1] = q[i] + 1 \tag{2.84}$$

although many others have been proposed (Müller, 2007; Sakoe and Chiba, 1978). This minimization can be solved in $\mathcal{O}(MN)$ time using dynamic programming (Sakoe and Chiba, 1978). A common equivalent way to frame this problem which fits straightforwardly into the framework of dynamic programming is to construct a pairwise distance matrix $C \in \mathbb{R}^{M \times N}$ by computing $C[i,j] = \|X[i] - Y[i]\|_2$ and then finding the lowest-cost path (subject to appropriate constraints) through the matrix. This perspective prompts path entries where $p[i+1] = p[i] + 1, q[i+1] = q[i] + 1$ to be referred to as "diagonal moves".

DTW is often constrained so that p and q span the entirety of X and Y, i.e. $p[1] = q[1] = 1$ and $p[L] = M; q[L] = N$. However, this constraint is inapplicable when subsequence alignment is allowed in which case it is required that either $g \min(M,N) \leq p[L] \leq M$ or $g \min(M,N) \leq q[L] \leq N$ where $g \in (0,1]$ (the "gully") is a parameter which determines the proportion of the subsequence which must be successfully matched. In addition, the path is occasionally further constrained so that

$$p[i] - q[i] + R \leq M, q[i] - p[i] + R \leq N$$

for $i \in \{1, \ldots, L\}$ where $R = g \min(M,N)$ is the "radius", sometimes called the Sakoe-Chiba band (Sakoe and Chiba, 1978). An alternative way to encourage the

alignment to favor "diagonal" paths is to add a penalty to equation (2.81):

$$p, q = \arg\min_{p,q} \sum_{i=1}^{L} \|X[p[i]] - Y[q[i]]\|_2 + \begin{cases} \phi, p[i] = p[i-1] \text{ or } q[i] = q[i-1] \\ 0, \text{otherwise} \end{cases} \tag{2.85}$$

where ϕ is a tuneable parameter controlling how much to discourage non-diagonal path entries.

Once an optimal alignment path is found, a rough similarity measure can be obtained as the sum of the pairwise distances between aligned sequence elements from X and Y. In addition, to achieve alignment one sequence can be adjusted (via resampling or otherwise) so that so that $t_X[p[i]] = t_Y[q[i]], \forall i \in \{1, \ldots, L\}$, where $t_X \in \mathbb{R}^M, t_Y \in \mathbb{R}^N$ are the times corresponding to the feature vectors in X and Y respectively. An example visualization of DTW on two one-dimensional sequences can be seen in figures 2.13 and 2.14.

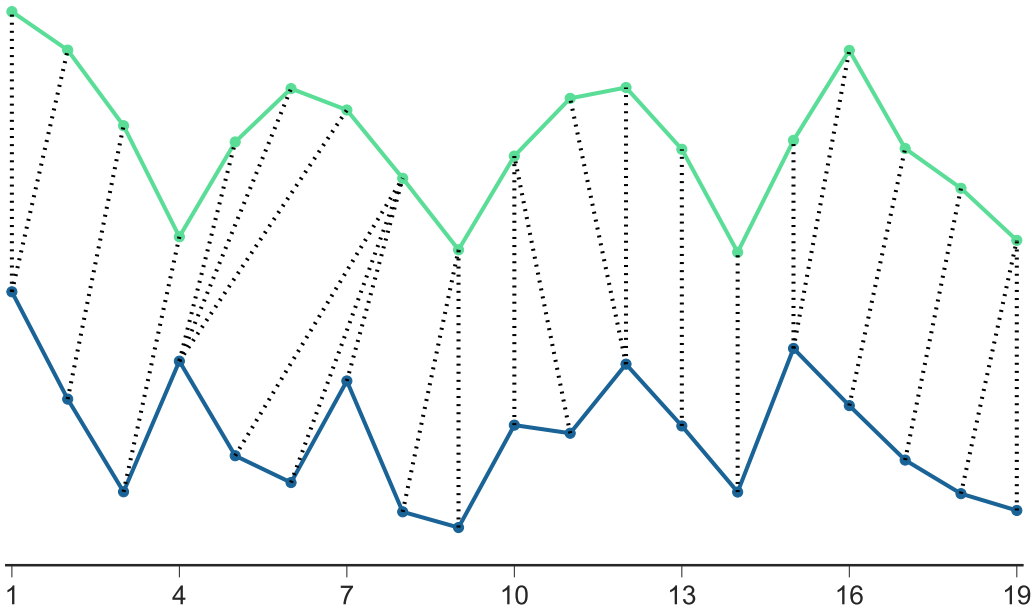

Figure 2.13: Example DTW alignment between two one-dimensional sequences. The determined correspondences between elements of the two sequences is shown as dashed lines.

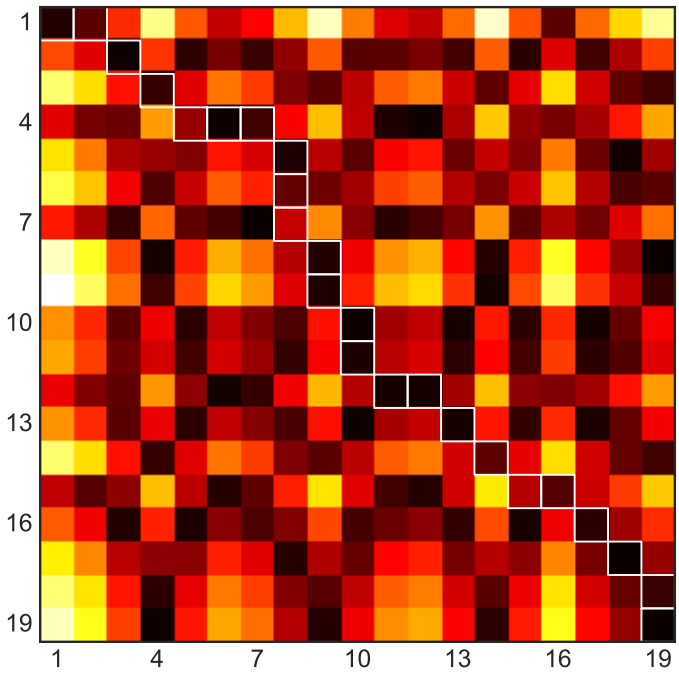

Figure 2.14: The pairwise distance matrix C and the lowest-cost path through it for the sequences shown in figure 2.13, which defines the optimal DTW alignment. The x- and y-axes of the distance matrix correspond to sequence indices in the green and blue sequences in figure 2.13 respectively.

Chapter 3

MIDI Files

MIDI (Music Instrument Digital Interface) is a hardware and software standard for communicating musical events. First proposed in 1983 (International MIDI Association, 1983), MIDI remains a highly pervasive standard both for storing musical scores and communicating information between digital music devices. Its use is perhaps in spite of its crudeness, which has been lamented since MIDI's early days (Moore, 1988); most control values are quantized as 7-bit integers and information is transmitted at the relatively slow (by today's standards) 31,250 bits per second. Nevertheless, its efficiency and well-designed specification make it a convenient way of formatting digital music information.

In this thesis, we will make extensive use of on MIDI files, which in a simplistic view can be considered a compact way of storing a musical score. MIDI files are specified by an extension to the MIDI standard (International MIDI Association, 1988) and consist of a sequence of MIDI messages organized in a specific format. A typical MIDI file contains timing and meter information in addition to a collection of one or more "tracks", each of which contains a sequence of notes and control messages. The General MIDI standard (International MIDI Association, 1991) further specifies

a collection of 128 instruments on which the notes can be played, which standardizes the playback of MIDI files and has therefore been widely adopted.

When paired with a General MIDI synthesizer, MIDI files have been used as a sort of *semantic* audio codec, with entire songs only requiring a few kilobytes of storage. The early availability of this "coding method", combined with the expense of digital storage in the 90s, made MIDI files a highly pervasive method of storing and playing back songs before the advent of the MP3. Even after high-quality perceptual audio codecs were developed and storage prices plummeted, MIDI files remained in use in resource-scarce settings such as karaoke machines and cell phone ringtones. As a result, there is an abundance of MIDI file transcriptions of music available today; through a large-scale web scrape, we obtained 178,561 MIDI files with unique MD5 checksums. Given their wide availability, we believe that MIDI files are underutilized in the music information retrieval community. The core motivating problem of this thesis is the question of how we can leverage this huge amount of data. To put this problem in context, this chapter provides an introduction to the sources of information available in MIDI files and discusses steps necessary for utilizing them.

3.1 Information Available in MIDI Files

While various aspects of MIDI files have been used in MIR research, to our knowledge there has been no unified overview of the information they provide, nor a discussion of the availability and reliability of this information in MIDI transcriptions found "in the wild". We therefore present an enumeration of the different information sources in a typical MIDI file and discuss their applicability to different MIR tasks. Because not all MIDI files are created equal, we also computed statistics about the presence and quantity of each information source across our collection of 176,141 MIDI files;

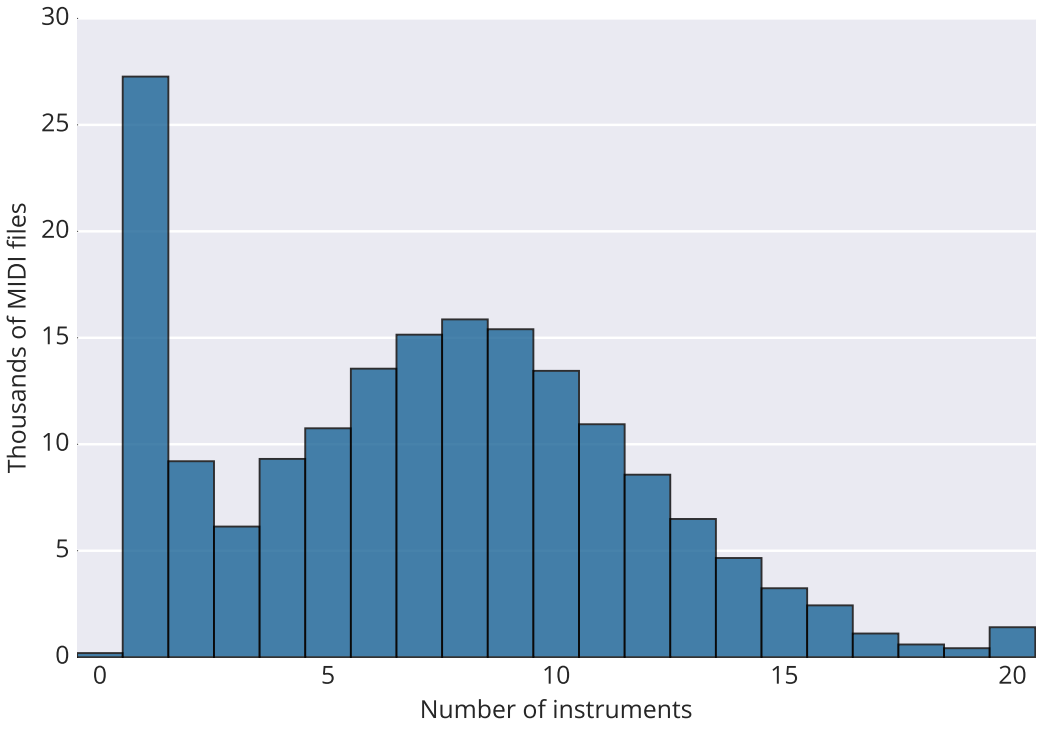

Figure 3.1: Histogram of the number of instruments per MIDI file in our collection. While many files contained only a single instrument, the majority were multi-voice transcriptions.

the results can be seen in figures 3.1 to 3.8 and will be discussed in the following sections.

3.1.1 Transcription

MIDI files are specified as a collection of "tracks", where each track consists of a sequence of MIDI events on one of 16 channels. Commonly used MIDI events are note-on and note-off messages, which together specify the start and end time of notes played at a given pitch on a given channel. Various control events also exist, such as pitch bends, which allow for finer control of the playback of the MIDI file. Program change events determine which instrument these events are sent to. The General

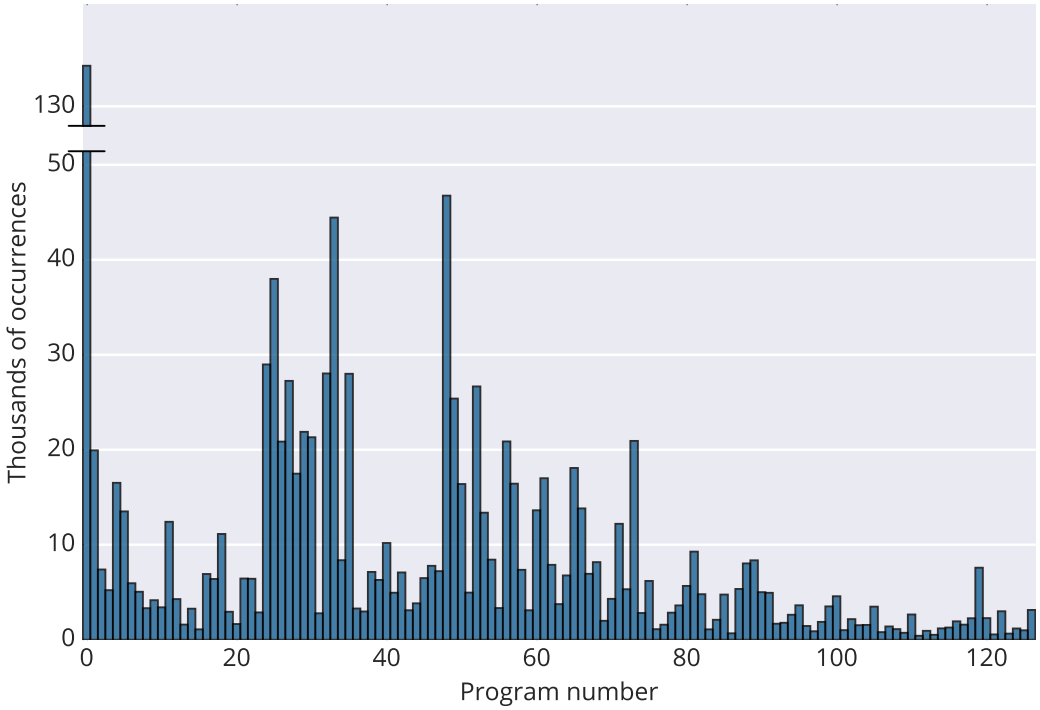

Figure 3.2: Histogram of the number of occurrences of different program numbers for non-drum instruments across all files in our MIDI file collection. Most common is program 0, corresponding to "acoustic grand piano", which is the default instrument on many keyboards and transcription software programs. Program numbers above 96 are relatively uncommon, and correspond to "Synth Effects", "Ethnic", "Percussive", and "Sound Effects" instruments.

MIDI standard defines a correspondence between program numbers and a predefined list of 128 instruments. General MIDI also specifies that all notes occurring on MIDI channel 10 play on a separate percussion instrument, which allows for drum tracks to be transcribed. The distribution of the total number of program change events (corresponding to the number of instruments) across the MIDI files in our collection and the distribution of these program numbers are shown in figure 3.1 and figure 3.2 respectively. The four most common program numbers (shown as the four tallest bars in figure 3.2) were 0 ("Acoustic Grand Piano"), 48 ("String Ensemble 1"), 33 ("Electric Bass (finger)"), and 25 ("Acoustic Guitar (steel)").

This specification makes MIDI files naturally suited to be used as transcriptions of pieces of music, due to the fact that they can be considered a sequence of notes played at different "velocities" (intensities) on a collection of instruments. As a result, many MIDI files are transcriptions and are thus commonly used as training data for automatic transcription systems (see (Turetsky and Ellis, 2003) for an early example). This type of data also benefits score-informed source separation methods, which utilize the score as a prior to improve source separation quality (Ewert et al., 2014). An additional natural use of this information is for "instrument activity detection", i.e. determining when certain instruments are being played over the course of a piece of music. Finally, the enumeration of note start times lends itself naturally to onset detection, and so MIDI data has been used for this task (Bello et al., 2005).

3.1.2 Music-Theoretic Features

Because many MIDI files are transcriptions of music, they can also be used to compute high-level musicological characteristics of a given piece. Towards this end, the software library `jSymbolic` (McKay and Fujinaga, 2006) includes functionality to extract a wide variety of features, including instrumentation, rhythm, and pitch

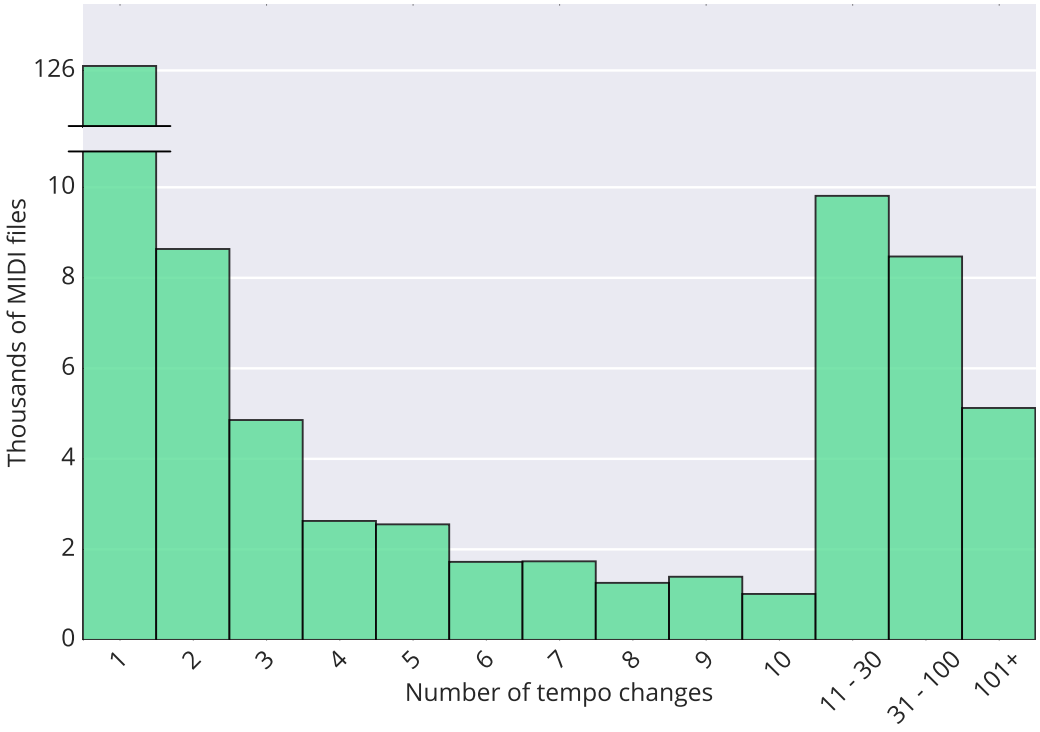

Figure 3.3: Histogram of the number of tempo change events per file in our collection. The vast majority (over 126,000) were transcribed at a fixed tempo.

statistics. Similarly, `music21` (Cuthbert and Ariza, 2010) provides a general-purpose framework for analyzing collections of digital scores (including MIDI files). Computing these features on a collection of MIDI transcriptions is valuable for computational musicology and can enable data-driven corpus studies. For example, (Cuthbert et al., 2011) discusses the use of `music21` and `jSymbolic` to extract features from scores and uses them to distinguish music from different composers and musical traditions.

3.1.3 Meter

Timing in MIDI files is determined by two factors: The MIDI file's specified "resolution" and tempo change events. Each event within the MIDI file specifies the

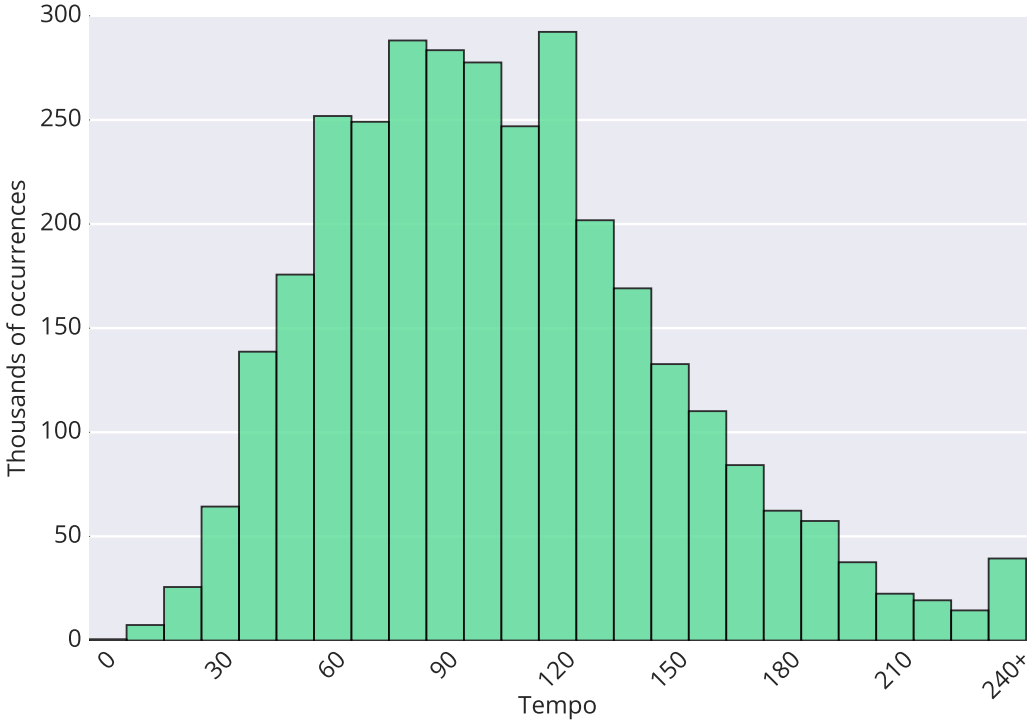

Figure 3.4: Histogram of tempos across all files in our MIDI file collection. Unlike the other histograms in this chapter, each bar corresponds to a range of values.

number of "ticks" between it and the preceding event. The resolution, which is stored in the MIDI file's header, sets the number of ticks which correspond to a single beat. The amount of time spanned by each tick is then determined according to the current tempo, as set by tempo change events. For example, if a MIDI file has a resolution of 220 ticks per beat and the current tempo is 120 beats per minute,[1] each tick would correspond to $60/(120 * 220) = 0.002\overline{27}$ seconds. If a MIDI event in this file is specified to occur 330 ticks after the previous event, then it would occur $330 * 0.002\overline{27} = .75$ seconds later.

The timing in a MIDI file can vary over time by including many tempo change

[1] Actually, tempo change events specify the number of microseconds per quarter beat, but this can be readily converted to beats per minute.

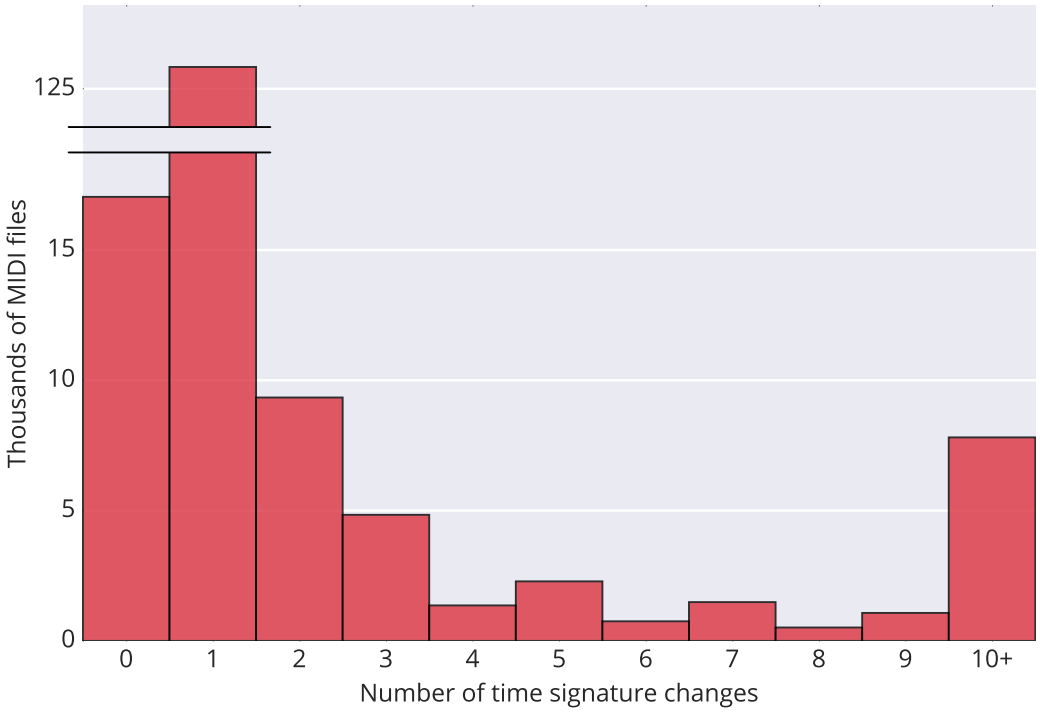

Figure 3.5: Histogram of the number of time signature change events per file in our collection. Although these events can be omitted, almost all of our MIDI files had at least one.

events. In practice, as shown in figure 3.3, most MIDI files only contain a single tempo change and are therefore transcribed at a fixed tempo. However, there are many MIDI files in our collection which have a large number of tempo change events (as indicated by the rightmost bars in figure 3.3). We have found that this is a common practice for making the timing of a MIDI transcription closely match that of an audio recording of the same song. Despite the fact that the default tempo for a MIDI file is 120 beats per minute, figure 3.4 demonstrates that a wide range of tempos are used. In practice, we find that this is due to the fact that even when a single tempo event is used, it is often set so that the MIDI transcription's tempo approximates that of an audio recording of the same song.

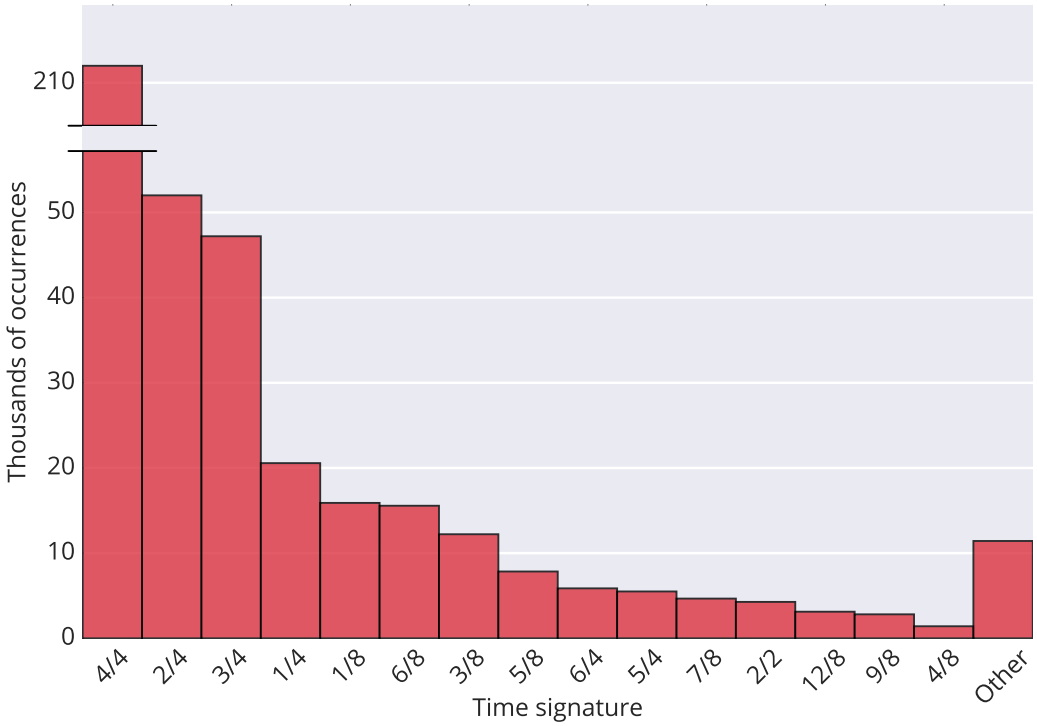

Figure 3.6: Histogram of different time signatures found in all files in our MIDI file collection. Surprisingly, a substantial number of non-4/4 time signatures were found.

Time signature change events further augment MIDI files with the ability to specify time signatures, and are also used to indicate the start of a measure. By convention, MIDI files have a time signature change at the first tick, although this is not a requirement. Because time signature changes are relatively rare in western popular music, the vast majority of the MIDI files in our collection contain a single time signature change, as seen in figure 3.5. Despite the fact that 4/4 is the default time signature for MIDI files and is pervasive in western popular music, a substantial portion (about half) of the time signature changes were not 4/4, as shown in figure 3.6.

Because MIDI files are required to include tempo information in order to specify

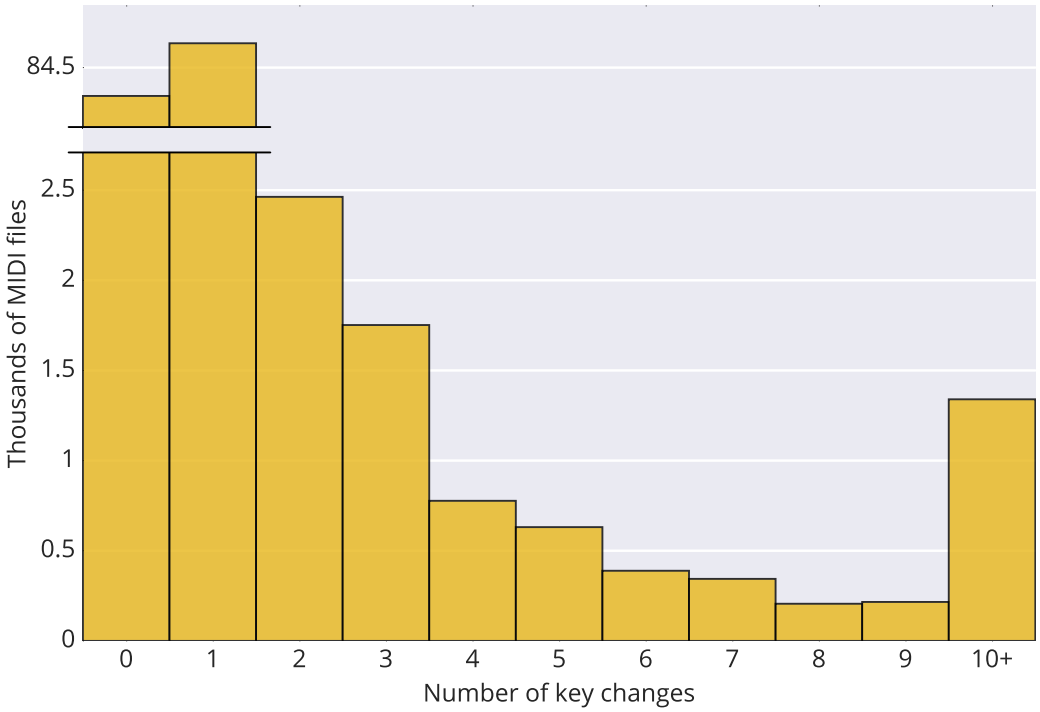

Figure 3.7: Histogram of the number of key change events per MIDI file in our collection. Like time signature changes, these events are optional, but the majority of our MIDI files had at least one.

their timing, it is straightforward to extract beat locations from a MIDI file. By convention, the first (down)beat in a MIDI transcription occurs at the first tick. Determining the beat locations in a MIDI file therefore involves computing beat locations starting from the first tick and adjusting the tempo and time signature according to any tempo change or time signature change events found. Despite this capability, to our knowledge MIDI files have not been used as ground truth for beat tracking algorithms. However, (Mauch and Dixon, 2012) utilized a large dataset of MIDI files to study drum patterns using natural language processing techniques.

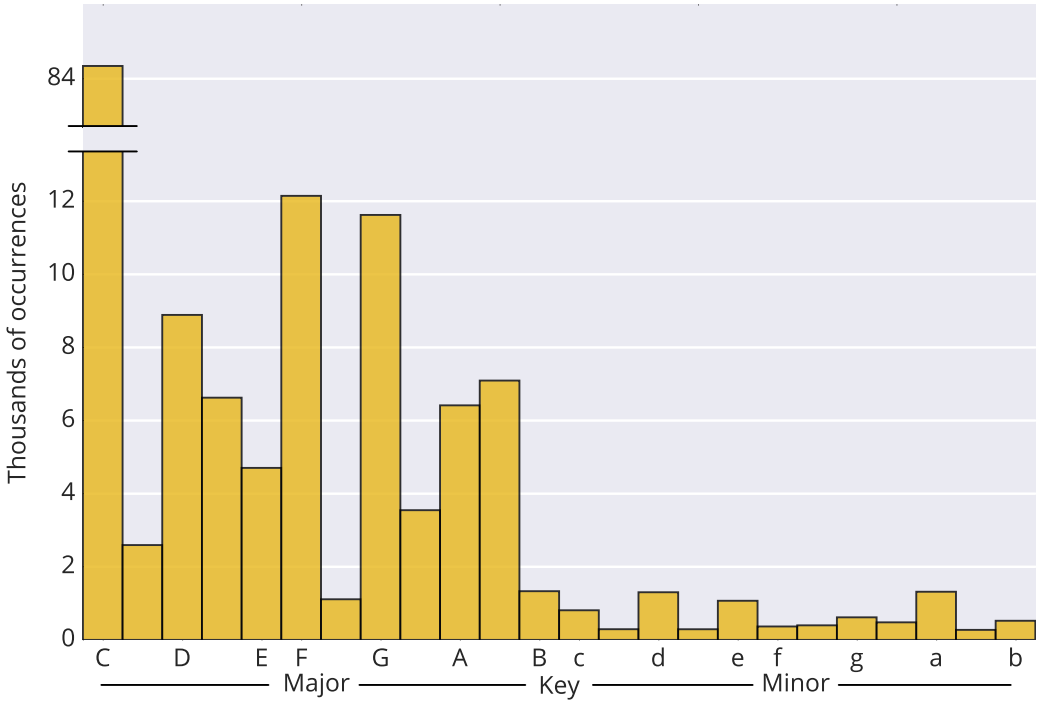

Figure 3.8: Histogram of different keys found in all MIDI files in our collection. An unrealistically large number of C major key annotation were found.

3.1.4 Key

An additional useful event in MIDI files is the key change event. Any of the 24 major or minor keys may be specified. Key changes simply give a suggestion as to the tonal content and do not affect playback, and so are a completely optional meta-event. As seen in figure 3.7, this results in many MIDI files omitting key change events altogether. A further complication is that a disproportionate number (about half) of the key changes in the MIDI files in our collection were C major, as shown in figure 3.8. This disagrees with corpus studies of popular music, e.g. (Carlton, 2012) which found that only about 26% of songs from the Billboard 100 were in C major. We believe this is because many MIDI transcription software packages automatically

insert a C major key change at the beginning of the file.

3.1.5 Lyrics

Lyrics can be added to MIDI transcriptions by the use of lyrics meta-events, which allow for timestamped text to be included over the course of the song. This capability enables the common use of MIDI files for karaoke; in fact, a separate file extension ".kar" is often used for MIDI files which include lyrics meta-events. Occasionally, the generic text meta-event is also used for lyrics, but this is not its intended use. In our collection, we found 23,801 MIDI files (about 13.3%) which had at least one lyrics meta-event.

3.1.6 What's Missing

Despite the wide variety of information sources available in MIDI files outlined in the previous sections, there are various types of information which are not possible (or not common) to store in MIDI files. While the General MIDI specification includes the vocal instruments "Choir Aahs", "Voice Oohs", "Synth Choir", "Lead 6 (voice)" and "Pad 4 (choir)", in practice there is no specific program number (or numbers) which is consistently used to transcribe vocals. As a result, in a given MIDI file there is no reliable way of determining which instrument is a transcription of the vocals in a song. Furthermore, because a substantial portion of MIDI files were designed for karaoke, the vocals may not be transcribed at all.

While the MIDI specification does include "track name", "program name", and "instrument name" meta-events, they are not standardized and so are not used consistently. It follows that there is no simple way to retrieve the "melody" from a MIDI transcription, although the fact that all instruments are transcribed separately can make its estimation more straightforward than for audio files. For example,

(Tang et al., 2000) explores the use of simple features such as the average velocity and note range within a track to predict whether it is a melody, and also finds that in a small dataset the track name reliably indicates a melody track 44.3% of the time. Similarly, (Ponce de León Amador et al., 2008) uses heuristic features and a random forest classifier to predict with high accuracy whether a track is a melody.

There is also no explicit way for MIDI files to include chord labels or structural segmentation boundaries (e.g. "verse", "chorus", "solo"). While this would in principle be possible thanks to the generic MIDI "text" meta-event, we have yet to find any MIDI files which store this information. Nevertheless, estimating chords in particular is greatly facilitated by the presence of a ground truth transcription. Both `music21` (Cuthbert and Ariza, 2010) and `melisma` (Sleator and Temperley, 2001) include functionality for estimating chord sequences from symbolic data. Rhodes et al.(Rhodes et al., 2007) also proposed a symbolic chord estimation method using Bayesian Model Selection, which was shown to outperform `melisma` on a dataset of Beatles MIDI files in (Ewert et al., 2012).

While text meta-events could also be used to store song-level metadata (song title, artist name, etc.) in a MIDI file, we seldom encountered this. There is no standardized way to store this metadata in a MIDI file, although we found that a minority of the filenames in our collection indicated the song title and occasionally the artist name. The lack of a metadata specification also inhibits the attribution of MIDI transcriptions to the person who transcribed them.

3.2 Utilizing MIDI Files as Ground Truth

Utilizing MIDI files as ground truth information for audio content-based MIR tasks requires the following: First, the compact low-level binary format used by MIDI files must be parsed so that the information can be readily extracted. Second, the artist

and song of a MIDI file must be determined so it can be paired with a corresponding audio recording. Finally, for many uses, the MIDI file must be aligned in time with its matching audio.

3.2.1 Extracting Information

The information sources enumerated in Section 3.1 are not readily available from MIDI files due to fact that they follow a low-level binary protocol. For example, in order to extract the time (in seconds) of all onsets from a given instrument in a MIDI file, note events which occur on the same track and channel as program change events for the instrument must be collected and their timing must be computed from their relative ticks using the global tempo change events. Fortunately, various software libraries have been created to facilitate this process. `pretty_midi` (Raffel and Ellis, 2014), discussed in section B.3, simplifies the extraction of useful information from MIDI transcriptions by taking care of most of the low-level parsing needed to convert the information to a more human-friendly format. It contains functions for retrieving beats, onsets, and note lists from specific instruments, and the times and values of key, tempo, and time signature changes. It also can be used to modify MIDI files, as well as to convert them to synthesized audio or a spectrogram-like piano roll representation. The aforementioned `jSymbolic` contains an extensive collection of routines for computing musicological features from MIDI files. Finally, both `music21` and `melisma` are capable of inferring high-level music information from symbolic data of various types, including MIDI.

3.2.2 Matching

Apart from metadata-agnostic corpus studies such as (Mauch and Dixon, 2012), determining the song a given MIDI file represents is usually required. Matching a

given MIDI file to, for example, a corresponding entry in the Million Song Dataset (Bertin-Mahieux et al., 2011) can be beneficial even in experiments solely involving symbolic data analysis because it can provide additional metadata for the track including its year, genre, and user-applied tags. Utilizing information in a MIDI file for ground truth in audio content-based MIR tasks further requires that it be matched to an audio recording of the song, but this is made difficult by the lack of a standardized method for storing song-level metadata in MIDI files (as discussed in Section 3.1.6). Content-based matching offers a solution; for example, early work by Hu et al. (Hu et al., 2003) assigned matches by finding the smallest dynamic time warp (DTW) distance between spectrograms of MIDI syntheses and audio files across a corpus. However, this approach is prohibitively slow for very large collections of MIDI and/or audio files.

3.2.3 Aligning

There is no guarantee that a MIDI transcription for a given song was transcribed so that its timing matches an audio recording of a performance of the song. For the many types of ground truth data that depend on timing (e.g. beats, note transcription, or lyrics), the MIDI file must therefore have its timing adjusted so that it matches that of the performance. Fortunately, score-to-audio alignment, of which MIDI-to-audio alignment is a special "offline" case, has received substantial research attention. A common method is to use DTW or another edit-distance measure to find the best alignment between spectrograms of the synthesized MIDI and the audio recording; see (Raffel and Ellis, 2016b) or (Ewert et al., 2012) for surveys.

In practice, audio-to-MIDI alignment systems can fail when there are overwhelming differences in timing or deficiencies in the transcription, e.g. missing or incorrect notes or instruments. Ideally, the alignment and matching processes would automat-

ically report the success of the alignment and the quality of the MIDI transcription. We do not know of any research into automatically determining the quality of a MIDI transcription.

3.2.4 Roadmap

Given our extremely large collection of MIDI files and the possibility of leveraging the sources of information available in them outlined in section 3.1, the remainder of this thesis focuses on matching and aligning this collection to the Million Song Dataset (MSD) (Bertin-Mahieux et al., 2011). Due to the scale of this problem, we will develop a suite of learning-based methods for efficient large-scale comparison of sequences. We will then demonstrate how the combination of these approaches allows us to rapidly match and align our huge MIDI file collection to the MSD and discuss some possible applications for the resulting data.

Chapter 4

Optimizing Dynamic Time Warping

The central application of this thesis is the problem of matching musical scores to corresponding audio recordings in a large database. This is motivated by our collection of 178,561 unique MIDI transcriptions which, as we explored in chapter 3, can provide a bounty of ground truth data for content-based music information retrieval tasks. To maximize its value, a given MIDI file must be both matched and aligned in time to a corresponding music audio file. In both scenarios, it is of great value to produce a confidence score which can communicate both how well the content in a MIDI file matches a given audio file and the quality of the transcription.

Most previous research has focused on systems meant for either alignment or matching but not both. In the context of MIDI-to-audio alignment, a wide variety of techniques have been proposed to determine a correspondence between discrete times in the audio and MIDI files. While some approaches use edit distance measures such as Smith-Waterman (Ewert et al., 2012) and Needleman-Wunsch (Grachten et al., 2013), the most common approach is dynamic time warping (DTW) which

we described in section 2.2.3. Because DTW produces monotonic alignments, it is well-suited for audio-to-MIDI alignment when we expect that the MIDI is an accurate continuous transcription (i.e. without out-of-sequence or incorrect sections).

The total distance between aligned pairs of feature vectors produced by DTW provides a single global value which can be used as a natural measure of the "similarity" between two sequences. In fact, DTW has seen extensive use as a way of measuring sequence similarity in the data mining literature (Berndt and Clifford, 1994).

Despite its widespread use, DTW's success is dependent on its parametrization as well as system design choices such as the feature representation used for sequence elements. To our knowledge, there has been no large-scale quantitative comparison of different DTW-based alignment systems. This is likely due to the fact that evaluating a given system's performance would require either a large collection of MIDI and audio pairs for which the correct alignment is already known (which does not exist) or manual audition and rating of the output of the systems (which is time-consuming).

This chapter aims to remedy this by searching across a large space of DTW designs to optimize both alignment accuracy and confidence reporting. First, we enumerate some common design choices in DTW-based alignment systems in section 4.1. In section 4.2, we propose a method for creating a synthetic dataset of MIDI-audio pairs by applying realistic corruptions to MIDI files, allowing us to create an arbitrary amount of data where we know a priori the correct alignment. We then tune parameters for alignment using Bayesian optimization (section 4.3) and for confidence reporting using an exhaustive search (section 4.4). Finally, in section 4.5 we perform a large-scale qualitative evaluation of our proposed alignment system on real-world data and discuss possibilities for improvement.

4.1 DTW-Based Alignment

From the discussion in section 2.2.3, it is clear that there are many parameters which govern the behavior of DTW. In addition, because DTW involves aligning sequences of feature vectors, domain-specific considerations such as the data representation will also affect its outcome. In general, there is no consensus as to what parameter settings generally work best for MIDI-to-audio alignment. The following list enumerates some of the more relevant parameters which must be tuned in DTW-based audio-to-MIDI alignment systems and references prior works to give example settings. For reference, we follow the notation used in section 2.2.3.

Feature representation (X and Y): Prior to alignment, audio and MIDI data must be converted to an intermediate representation where their distance can be computed. This is often done by synthesizing the MIDI file to obtain an audio signal and computing a common spectral transform of the audio recording and synthesized MIDI audio. Chroma vectors, which represent the amount of energy in each semitone summed across octaves (Fujishima, 1999) are a common choice (Hu et al., 2003; Ewert et al., 2012). The constant-Q transform (CQT) (discussed in section 2.2.2.3), which represents the amount of energy in logarithmically spaced bins (Brown, 1991) has also been used (Raffel and Ellis, 2015b; Dixon and Widmer, 2005; Ellis, 2013). Occasionally, log-magnitude features are used in order to more closely mimic human perception (Raffel and Ellis, 2015b; Ellis, 2013; Turetsky and Ellis, 2003). In (Turetsky and Ellis, 2003) and (Hu et al., 2003) it is noted that Mel-Frequency Cepstral Coefficients result in poor performance for music signals because they obscure pitch information.

Time scale (t_X and t_Y): Feature vectors are frequently computed over short, overlapping frames of audio (Dixon and Widmer, 2005; Turetsky and Ellis, 2003;

Hu et al., 2003). Note that the spacing between feature vectors must be sufficiently small compared to the auditory temporal resolution (e.g. tens of milliseconds (Blauert, 1997)) in order for a perceptually accurate alignment to be possible. Occasionally, beat-synchronous feature vectors are used (Raffel and Ellis, 2015b; Ellis, 2013), which can reduce computation time and can produce accurate alignments provided that the beat tracking is correct.

Normalization: In (Rakthanmanon et al., 2012), it is argued that z-scoring (standardizing) the feature sequences is critical for data mining applications of DTW, which was used in audio-to-MIDI alignment in (Hu et al., 2003). In addition, various normalizations have been applied to the feature vectors in X and Y before computing their local distances. A common choice is normalizing each vector by its L^2 norm, which is equivalent to using the cosine distance (Turetsky and Ellis, 2003; Ewert et al., 2012; Raffel and Ellis, 2015b; Ellis, 2013).

Penalty (ϕ): In many applications of DTW to audio-to-MIDI alignment, no additive penalty is used, which corresponds to setting $\phi = 0$. However, as long as the MIDI and audio files have consistent tempos, non-diagonal moves should be discouraged. In addition, it has been argued (Raffel and Ellis, 2015b) that when subsequence alignment is allowed, an additive penalty can be crucial to ensure that the entire subsequence is used, and so ϕ is set to the 90th percentile of the distances $\|X[n] - Y[m]\|_2^2$ for $m \in \{1, \ldots, M\}; n \in \{1, \ldots, N\}$. In (Ellis, 2013), a fixed value of $\phi = .5$ is used.

Gully (g) and band path constraint: The "gully" and band path constraint are also often omitted, which corresponds to setting $g = 1$. A value of g which is close to 1 will afford some tolerance to the possibility that the beginning or

end of the MIDI transcription is incorrect (e.g. a fade-out or lead-in), so (Ellis, 2013) sets $g = .7$ and (Raffel and Ellis, 2015b) sets $g = .95$. In data mining applications (Ratanamahatana and Keogh, 2004) it is argued that the band radius path constraint both reduces computational complexity and results in more reliable alignments by avoiding paths with many non-diagonal moves.

4.2 Creating a Synthetic Alignment Dataset

The list above defines a large space of parameter settings/design choices for DTW-based audio-to-MIDI alignment systems. Previous research has typically involved manually tuning alignment parameters based on a modest-sized test set of MIDI/audio pairs and informally auditioning the aligned MIDI data. Because this method only facilitates small-scale comparisons, there is an obvious question of what parameter settings would yield the best general-purpose alignment system. Unfortunately, manual audition of even a tiny subset of possible parameter settings on a modestly-sized collection of paired MIDI and audio files is infeasible, let alone a collection large enough to make substantive judgements about the general performance of a given system. Furthermore, automatic evaluation has been blocked by the lack of a large ground truth dataset of "correct" alignments. We therefore propose a method for synthetically creating MIDI/audio pairs with known alignments by applying a series of corruptions to MIDI files to resemble real-world conditions. When applying the corruptions, we keep track of the adjustments needed to correctly align the corrupted MIDIs, which allows us to rapidly and automatically evaluate a huge number of possible systems.

To create this dataset, we first collected 1,000 MIDI files which were transcriptions of Western popular music songs. We then applied the following series of transformations, based on our experience with common differences between MIDI transcriptions

and audio recordings: First, to simulate differences in tempo, we adjusted the timing in each MIDI file by a low-frequency length-N random signal r, defined as

$$s[m] \sim \mathcal{N}(0, \sigma_t), m \in \{1, \ldots, N+1\} \tag{4.1}$$

$$R[k] = \begin{cases} s[k]e^{-k+1}, & k \in \{1, \ldots, N+1\} \\ R[2N-k+1], & k \in \{N+2, \ldots, 2N\} \end{cases} \tag{4.2}$$

$$r[n] = \sum_{k=1}^{2N} R[k]e^{j\pi(k-1)(n-1)/N}, n \in \{1, \ldots, N\} \tag{4.3}$$

i.e. the inverse discrete Fourier transform of an exponentially decaying Gaussian-distributed random spectrum with standard deviation σ_t. Second, the first 10% and last 10% of the transcription were each cropped out with probability 50%, which simulates the MIDI file being an incomplete transcription. In addition, 1% of each transcription was cut out at a random location with 10% probability, which simulates a missing measure. Third, because it is common for a MIDI transcription to be missing an instrument (for example, karaoke renditions in which the lead vocal line has been deleted are frequently distributed as MIDI files), we removed each instrument track in each MIDI file with probability p_r, making sure never to remove all instruments. Fourth, in all MIDI files, we randomly added 1 or -1 to the program number of each instrument. This simulated the fact that when comparing a synthesized MIDI file to an audio recording, the timbre of a synthesized MIDI instrument is always somewhat different from its real-world counterpart. Finally, for each note in each instrument, we multiplied the velocity by a random number sampled from $\mathcal{N}(1, \sigma_v)$ while keeping it in the MIDI velocity range $[1, 127]$. This was meant to further simulate differences in instrument characteristics in real-world vs. synthesized songs, and also simulated missing notes for large σ_v. All MIDI data

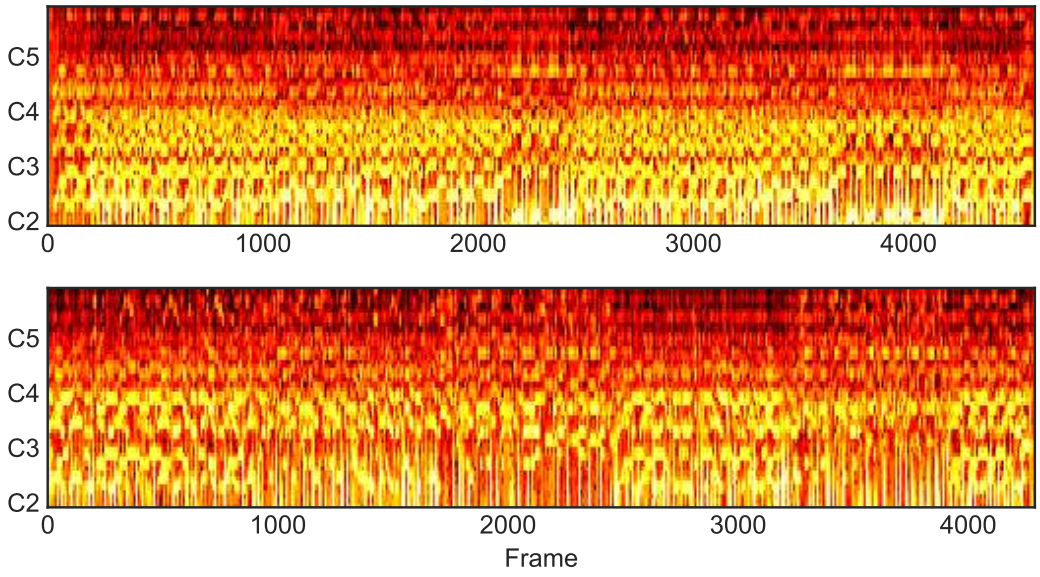

Figure 4.1: Constant-Q spectrograms of a synthesized MIDI file before (top) and after (bottom) undergoing the corruption process described in section 4.2. After synthesis, log-magnitude spectrograms with L^2-normalized spectra were computed every 46.4 milliseconds, with frequencies from MIDI note C2 (65.4 Hz) to B6 (987.8 Hz). This corruption was achieved using the parameters from the "hard" dataset, i.e. $\sigma_t = 20, p_r = .5, \sigma_v = 1$.

manipulations were realized with `pretty_midi` (Raffel and Ellis, 2014). Example constant-Q spectrograms of a synthesized MIDI file before and after undergoing this corruption process can be seen in figure 4.1. In addition, the timing offset curve which produced this corruption is shown in figure 4.2.

As described above, a DTW-based alignment system can serve two purposes: First, to align a MIDI transcription in time to an audio recording, and second, to produce a confidence score denoting the quality of the transcription or whether the MIDI file is a transcription of the recording at all. We therefore produced two sets of corrupted versions of the 1,000 MIDI files we collected, one to measure alignment performance, and one to evaluate confidence reporting. For the first ("easy") set,

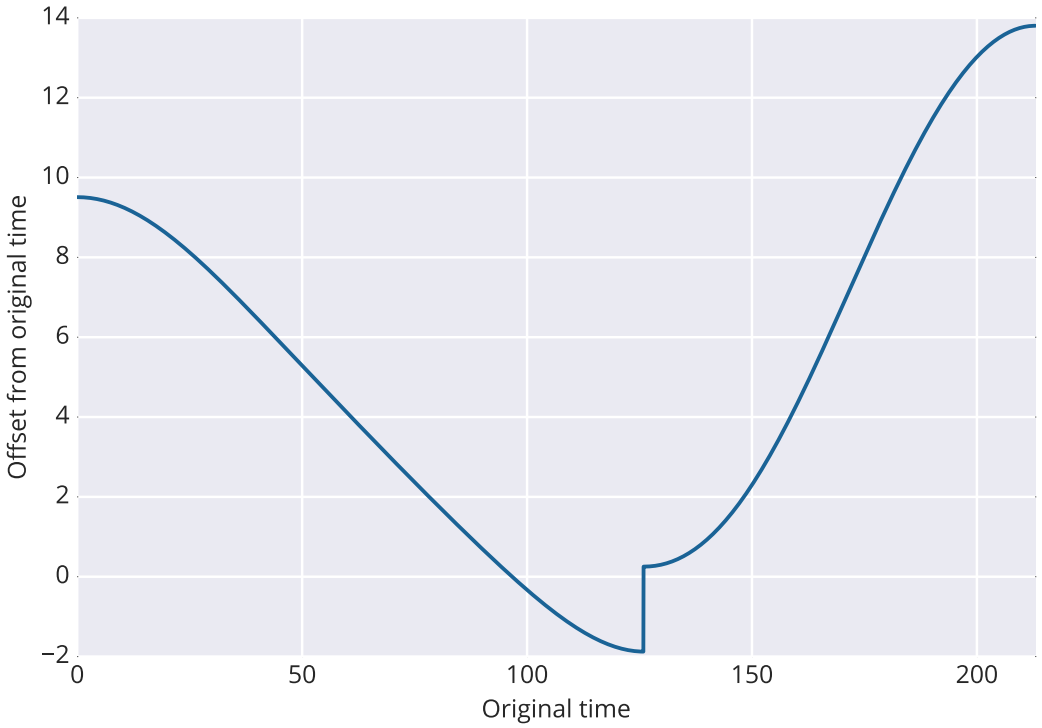

Figure 4.2: Warping offset curve which produced the time distortion in the corrupted MIDI file visualized in figure 4.1. The low-frequency variation is due to the random signal r (equation (4.3)) while the sudden jump around 125 seconds is caused by 10% of the MIDI file being cropped out. In our synthetic alignment task, the goal is to reverse this timing distortion over a large dataset of corrupted MIDI files.

we focused on corruption parameters corresponding to real-world conditions for a high-quality transcription, setting $\sigma_t = 5, p_r = .1, \sigma_v = .2$. For the second ("hard"), we set $\sigma_t = 20, p_r = .5, \sigma_v = 1$ so that the alignment task is sufficiently difficult to result in a significant number of incorrect alignments, which allows us to test whether an alignment system can automatically report failure.

4.3 Optimizing DTW-Based Alignment

Given a dataset of MIDI/audio pairs with known correct alignments, we can evaluate a given alignment scheme via the mean absolute alignment error across the set. The mean error quantifies the extent to which the alignment was able to remove the timing distortions (random warping and cropping) described in section 4.2. When the alignment has failed for a portion of the song, the error between the mapped times and the correct times may be very large, so we clip the absolute error to .5 seconds (which essentially denotes an error threshold above which all local alignment discrepancies are equally incorrect). Thus, our error metric is:

$$\frac{1}{L}\sum_{i=1}^{L}\min(|t_X[p[i]] - \hat{t}_X[q[i]]|, 0.5) \tag{4.4}$$

where $\hat{t}_X$ is the ground truth "warped" time scale. We average this measure across the test set. The resulting error $|t_X[p[i]] - \hat{t}_X[q[i]]|$ from aligning the spectrograms shown in figure 4.1 using our "gold standard" system (described later in section 4.4.1) is shown in figure 4.3.

Rapid calculation of this precise metric (over the "easy" test set) allows us to perform large-scale comparisons of different parameter settings. To decide which settings to try, we use Bayesian optimization, which approximates the relationship between parameter settings and objective values as a Gaussian process. Using

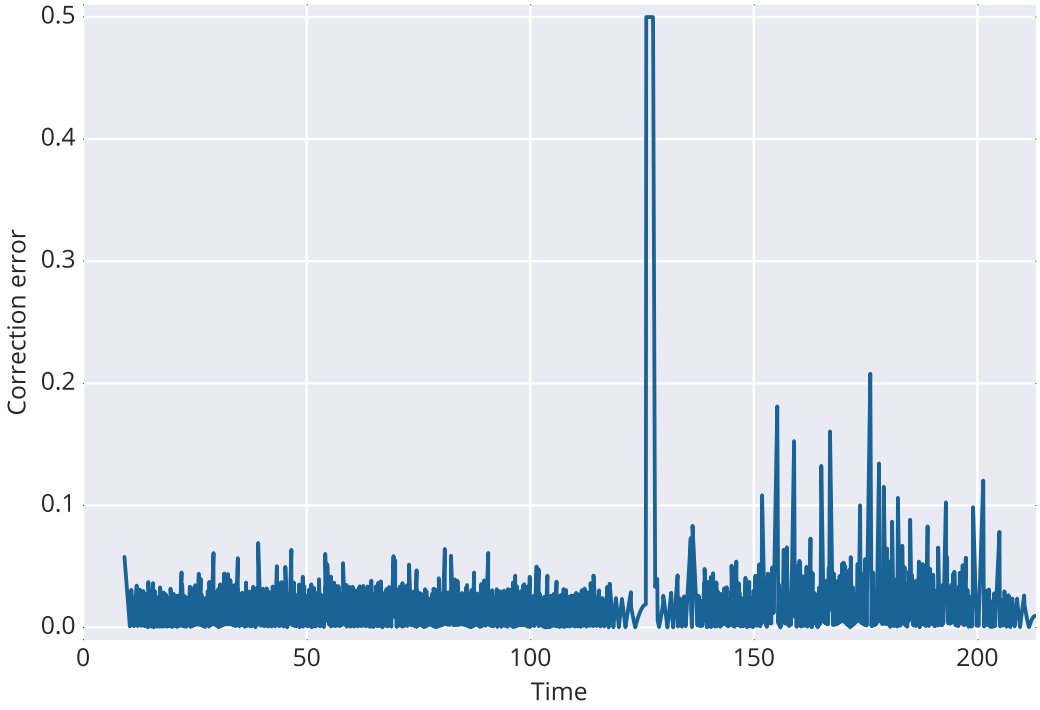

Figure 4.3: Absolute error between the true warping (shown in figure 4.2) and the estimated warping found by aligning the spectrograms in figure 4.1 using the system described in section 4.4.1. The error is clipped at .5 seconds, above which point we consider all discrepancies equally bad. The spike of largest error around 125 seconds is caused by the estimated warping not perfectly correcting the missing 10% of the corrupted MIDI which was randomly cropped out.

this formulation, Bayesian optimization can automatically propose new alignment systems based on the performance of previously-evaluated systems. A more thorough description of Bayesian optimization can be found in section 2.1.5. We utilized the framework and software proposed in (Snoek et al., 2012) in all of our experiments. For an acquisition function, we utilized the Expected Improvement (see section 2.1.5.2).

Based on the design choices outlined in section 4.1, we chose to optimize over the following parameter space:

Feature representation: We used either chroma vectors or constant-Q spectra. The constant-Q spectra spanned 4 octaves, starting from MIDI note C2 (65.4 Hz) with 12 bins per octave. In preliminary experiments, we found that all of the best-performing alignment systems used log-magnitude features regardless of whether chroma vectors or constant-Q spectra were used, so we computed log-magnitude features in all experiments.

Time scale: We either computed feature vectors every 46.4 milliseconds or utilized beat-synchronous features.

Normalization: We optionally z-scored the feature vectors, and normalized them by their L^1, L^2, L^∞ (max) norm, or not at all.

Penalty: For the penalty, we optimized a scale in $[0, 3]$ to apply to the median distance between all pairs of feature vectors in X and Y. Using the median distance made this penalty adaptive to different feature representations and normalization schemes.

Gully and band path constraint: We allowed the gully to take any value in $[0, 1]$ and optionally enforced the band path constraint.

This space subsumes most of the systems discussed in section 4.1. All feature extraction was realized with `librosa` (McFee et al., 2015b,c).

When performing Bayesian optimization, it is helpful to "seed" the optimization with objective values for many randomly-chosen parameter settings to ensure that the optimization thoroughly explores the possible parameter space. We therefore computed the accuracy for 10,000 randomly configured alignment systems and seeded the optimizer with the 100 systems which achieved the lowest mean error. In order to avoid local minima in the parameter space, we carried out 10 differently initialized Bayesian optimization runs with 100 trials each, resulting in 1,000 total trials.

The best-performing alignment system achieved an objective value of 0.0181, meaning that the average absolute error across the entire dataset was about 18 milliseconds. This is both close to the limit of the auditory temporal resolution and less than half of the time-scale resolution used for non-beat-synchronous feature vectors, so attaining a higher accuracy is likely unrealistic. A one-sample t-test was performed to determine which systems gave alignment quality statistically equivalent to the best performance. Testing the per-pair scores rather than the mean error across all pairs gave better robustness to outliers. Using a p-value of 0.05 as the threshold returned 51 configurations of indistinguishable quality.

As a high-level overview of these systems, none used beat-synchronization or a path constraint and all of them used log-magnitude constant-Q spectra as a feature representation and set both the penalty median scale and "gully" close to 1. Almost all of these systems used L^2 normalization (resulting in a cosine distance for local feature comparisons); a few used L^1 normalization. There was no clear trend in the use of z-scoring. Table 4.1 displays the parameter settings and the mean absolute error achieved by the 10 best-performing alignment systems. Interestingly, a relatively wide range (about 0.9 to 1.0) of median scale values for the additive penalty ϕ were

ϕ Median Scale	Standardize?	Gully g	Mean Error
1.035070	Yes	0.967203	0.0180094
0.840218	No	0.970180	0.0180690
0.944247	No	0.967956	0.0181160
0.822652	No	0.974826	0.0181264
0.929215	No	0.971404	0.0181429
0.920111	Yes	0.962121	0.0181909
0.947526	No	0.973988	0.0181951
0.893840	Yes	0.963922	0.0181988
0.906607	Yes	0.965711	0.0182004
0.937798	Yes	0.969747	0.0182236

Table 4.1: Parameters and mean absolute errors of the 10 systems which achieved the best performance on aligning the "easy" synthetic dataset. All of these systems utilized constant-Q transforms, normalized feature vectors by their L^2 norm, did not beat-synchronize, and did not use a band mask constraint.

effective. We propose that this indicates a relative insensitivity to this parameter setting, i.e. as long as ϕ is close to the distance matrix median the alignment system can produce high-quality alignments.

4.4 Optimizing Confidence Reporting

Having found alignment systems which achieve high accuracy on the "easy" dataset, we move on to the question of computing reliable alignment confidence scores. As described in section 2.2.3, DTW provides a "raw" score as the sum of distances between all aligned feature vectors, such that a small distance denotes high confidence. This measure is inappropriate, however, when the number of aligned feature vectors varies from song to song; in this setting, the mean distance is usually used instead of the total distance. Furthermore, it's not clear whether the non-diagonal path penalties ϕ should be included in this distance. Finally, (Raffel and Ellis, 2015b)

advocates further normalizing this distance by

$$\frac{1}{L^2} \sum_{i=\min(p)}^{\max(p)} \sum_{j=\min(q)}^{\max(q)} \|X[i] - Y[j]\|_2 \tag{4.5}$$

i.e. the mean distance between all pairs of frames over the entire aligned portion of both feature sequences. This is intended to help normalize out global differences between different distance matrices; for example, if a MIDI synthesis' timbre does not closely match an audio recording, entries in the resulting distance matrix will tend to be larger which would inflate the DTW distance.

To choose an optimal score normalization scheme, we first aligned all of the MIDI/audio pairs in both the "hard" and "easy" datasets using every alignment system generated during our Bayesian optimization trials. Then, for each system we computed the normalized DTW distance for every MIDI/audio pair using all combinations of the normalization schemes listed above (with and without length normalization, diagonal penalties, and mean distance normalization) resulting in 8 scores per file pair. Finally, we computed the Kendall rank correlation coefficient (Kendall, 1938) between each score and the mean absolute error produced by all alignment systems for every pair in both datasets. A high correlation indicates that the system is able to accurately report the quality of its alignments.

Among our 51 highest-accuracy systems, the highest correlation was 0.710. In general, the high-accuracy systems all produced rank correlations close to this value. All of the systems achieved the highest correlation when including the penalties in the score calculation, normalizing by the path length, and normalizing by the mean distance across the aligned portions. This suggests that these steps are all helpful for producing a reliable score. We visualize the normalized DTW distances and mean errors for each alignment produced by the system which had the best performance

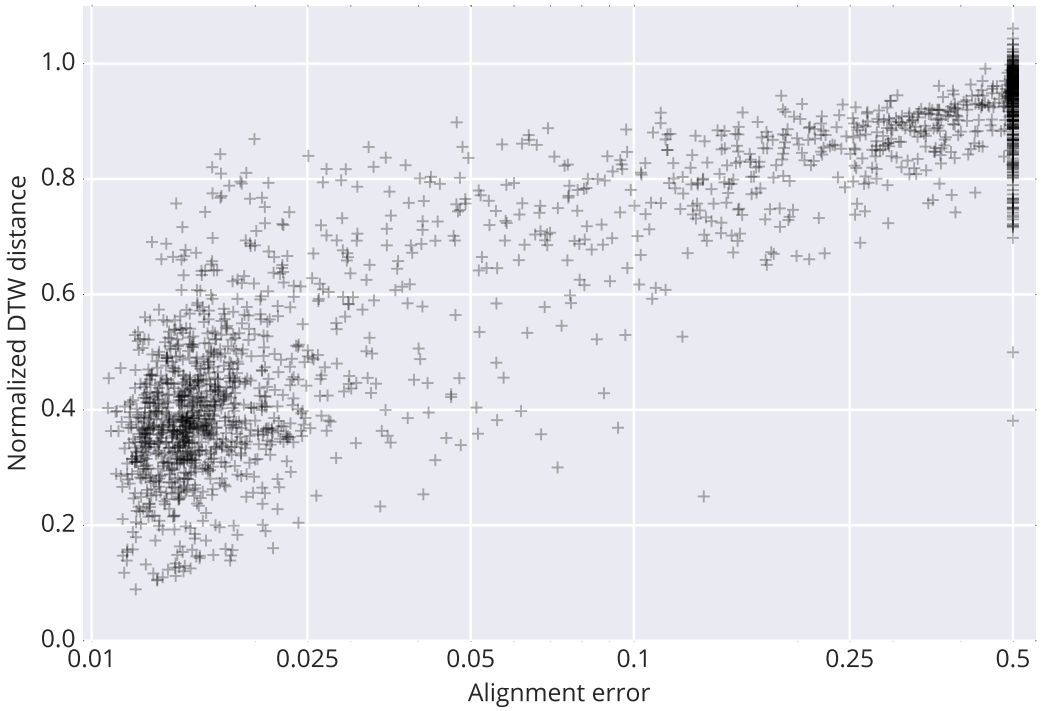

Figure 4.4: Scatter plot of alignment errors (x-axis, log-scaled) and normalized DTW distance (y-axis) produced by the highest-performing alignment system using the DTW distance normalization scheme which produced the highest correlation between the two axes. Ideally, alignments with a small normalized distance will have a small mean error, indicating that the score can reliably predict when alignment was successful. The group of points in the top right correspond to alignments which were globally wrong (resulting in a mean error of .5, the maximum value).

on the "easy" dataset in figure 4.4.

4.4.1 Choosing Easily-Reproducible Parameters

As a final step in producing a "gold standard" alignment system, we decided to find parameter settings that were easy to report and implement but nevertheless produced alignments and confidence scores which were not significantly different than the best systems. We chose the following design: For a feature representation, we

used log-magnitude constant-Q transforms computed on the non-beat-synchronous time scale described in section 4.3. We normalized the spectra by their L^2 norm and did not z-score them. We set the penalty ϕ to the median distance between all pairs of feature vectors in X and Y and used a gully parameter g of 0.96. This system achieved a mean absolute alignment error on the "easy" dataset of 0.0188, with alignment errors which were not significantly different from the best-performing system. By normalizing the DTW distance (including penalties) by the path length and the mean distance across aligned portions, we achieved a rank correlation between confidence and accuracy of 0.700. This "gold standard" system is straightforward to implement, and will be used for the remainder of this thesis.

4.5 Qualitative Evaluation on Real-World Data

In the experiments described above, we have found a system which can provide both high accuracy and a useful confidence measure on synthetic data. To determine its applicability outside of synthetic contexts, we performed a large-scale qualitative evaluation on real-world data. Our dataset consisted of 500 MIDI files which had artist and song title information in their filename, which allowed us to hand-match them to entries in the Million Song Dataset (Bertin-Mahieux et al., 2011). This collection comprises MIDI files with a full range of transcription qualities, and includes some pairs which are incorrectly matched due to metadata errors. By manually evaluating whether each resulting alignment was successful, we can determine how accurately the "gold standard" system performs alignments and how reliable the reported confidence scores are.

Conveniently, the vast majority of the normalized DTW distances for our real-world alignments fell into the range $[0.5, 1.0]$. In order to more intuitively report this

measure as a "confidence score", we decided to re-scale it as follows:

$$c = \max(0, \min(2(1 - s), 1)) \tag{4.6}$$

where s is the normalized DTW distance and c is the resulting confidence score. This maps the range to $[0, 1]$ where, ideally, 0 indicates a failure and 1 indicates a potential perfect alignment. We will use this re-scaled score for the remainder of this thesis.

After aligning all pairs in this dataset, we synthesized the resulting aligned MIDI files and created stereo recordings with the aligned synthesized MIDI audio in one channel and the original audio recording in the other. We then listened to each aligned MIDI pair and annotated a score from 1-5 as follows:

1. MIDI and audio file are incorrectly matched
2. Alignment failed due to major differences
3. Alignment was mostly successful with minor issues
4. Perfect alignment with minor transcription issues
5. Perfect alignment and transcription

For example, if a MIDI transcription was matched and successfully aligned to the correct song but was missing an instrument, a rating of 4 would be given; if the missing instrument caused the alignment to sound "sloppy", the rating would be 3 instead. Ideally, alignments with high confidence scores will be given higher ratings, and vice versa. To prevent biasing the results, we did not have access to the reported confidence score while rating a given aligned pair. For further analysis, we also recorded notes about any interesting qualitative characteristics of each alignment.

Rating	Confidence Score	Note
1	0.520807	Audio sounds like a remix
2	0.497009	Audio is remix
3	0.313761	Remix?
2	0.219259	Audio is remix
2	0.21846	Audio is remix/wrong section
1	0.16089	Audio is remix/live
2	0.146482	Different versions (remix?)
2	0.126939	Audio is remix
2	0.0978644	Very sloppy/audio is remix?
1	0.0739608	Audio is remix
1	0.0687189	Audio may be remix
1	0.0610702	Audio is probably remix
2	0.0484973	Different versions (remix?)
1	0.0431807	Audio is remix?
2	0.0353811	Audio is remix
1	0.0259135	Audio is remix
1	0.015929	Audio is remix
2	0.0117205	Audio is remix? Very sloppy
2	0.0116941	Audio is remix
1	0	Audio is a remix
2	0	Very sloppy/remix?

Table 4.2: Ratings, confidence scores, and annotations for real-world alignments where we thought the audio recording was a remix of the song the MIDI file was a transcription of. Because remixes tend to have substantial differences in structure and instrumentation, it is likely that alignment will fail.

For example, table 4.2 lists all of the ratings and confidence scores of alignments where we annotated that the audio recording was likely a remix (alternate version) of the song that the MIDI file is a transcription of.

Figure 4.5 shows the distributions of confidence scores for pairs assigned each of the five ratings. Apart from encountering some incorrect pairs (rated 1), we also found that various transcription issues prevented successful alignments. A common issue for those pairs rated 2 was that the wrong section of the MIDI transcription was matched

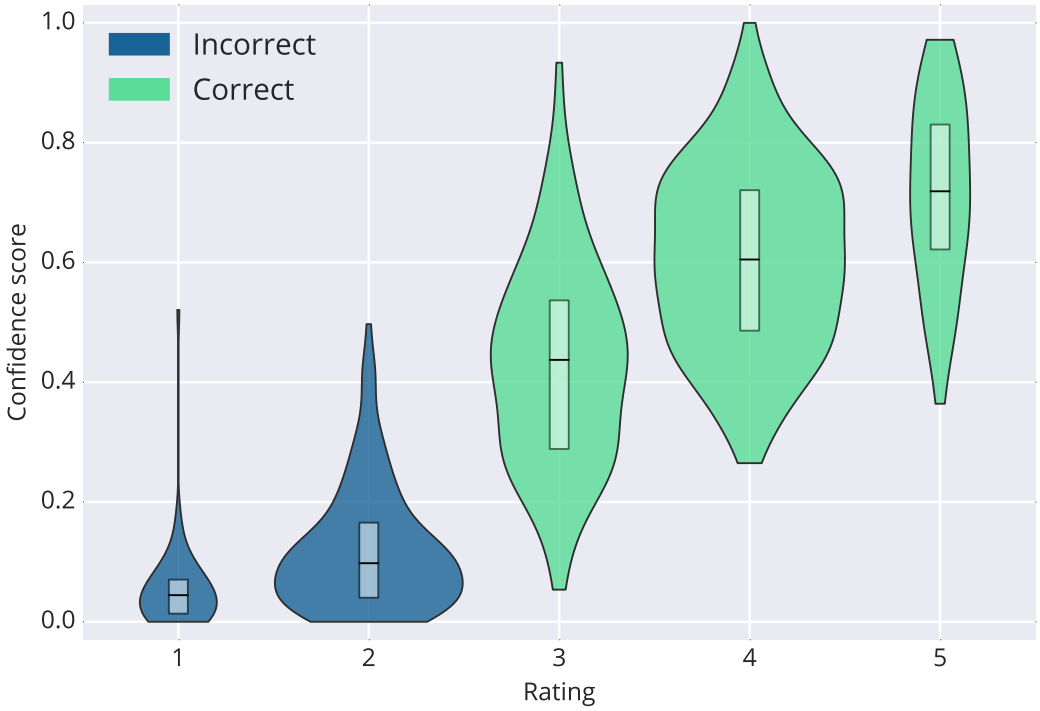

Figure 4.5: Violin plot (box plot with rotated kernel density estimates) showing the distribution of confidence scores for each rating in our qualitative evaluation. A smaller confidence score indicates a more successful alignment. The area of each violin corresponds to the number of pairs which had a given rating. Box plots in each violin show the median and upper and lower quartiles.

to the audio, often due to different instrumentation, keys, or versions (e.g. the audio was a remix). Any pairs rated 3 either had multiple missing instruments, many musical embellishments, or mismatched tempos. In addition, the overlap between the confidence scores for pairs rated 4 and 5 indicates that our confidence score is largely invariant to minor transcription issues. The most common transcription issue for pairs rated 4 was a single missing instrument or minor embellishments, most often on the vocal track.

Overall, our "gold standard" alignment system was able to successfully align most correctly-matched pairs and produced a reliable confidence score. Considering

pairs rated 3-5 as "correct" matches, the resulting confidence scores achieve an area under the receiver-operator characteristic curve of 0.981 (95% confidence interval [.973, .989], calculated by 1000-sample bootstrap), indicating a high quality measure. Unfortunately, there were a few pairs rated 1 or 2 which were not assigned small confidence scores; without these outliers, we could use a wider range of thresholds to obtain high-confidence alignments. The most important remaining flaw is the relative insensitivity to missing instruments and embellishments.

4.6 Discussion

In summary, large-scale optimization over synthetic data has delivered a DTW-based system which is simple to implement and achieves accurate and reliable results for both alignment and matching. We demonstrated that on real-world data, our alignment system produced a confidence score which provided a trustworthy indicator of whether the alignment was successful or not. Nevertheless, there was some overlap between the confidence score distributions for each rating. This implies that our system cannot detect fine-grained issues such as missing instruments or minor alignment errors.

We believe this is caused by the fact that alignment and confidence reporting have somewhat different goals which result in different requirements of the feature representation. For example, alignment benefits from a representation which is invariant to missing instruments so that it can still be successful when a transcription is incomplete. On the other hand, it would be helpful to have a confidence score which was sensitive to whether or not a transcription is missing instruments. Given that the framework we have explored requires that the representation is the same for both tasks, it is unsurprising that there are minor alignment issues which our confidence score is insensitive to. We therefore propose that exploring systems which

utilize different representations for alignment and confidence reporting would be a fruitful research direction.

We also note that there are other design choices and parameters for audio-to-MIDI alignment systems which we have not explored. These include larger selections of possible step sizes such as those described in (Müller, 2007; Sakoe and Chiba, 1978), different feature representations, and even completely different alignment methods such as Smith-Waterman (Ewert et al., 2012) and Needleman-Wunsch (Grachten et al., 2013). In addition, there are additional avenues to be explored for creating realistic synthetic data, such as utilizing different synthesis programs, modifying pitch bend and other control change messages, and applying realistic acoustic noise and nonlinearities (McFee et al., 2015a). Our proposed framework for creating alignment datasets and optimizing system design using Bayesian optimization could be straightforwardly extended to accommodate these new directions.

Given the success of our system at discriminating between correct and incorrect matches, it would appear that it is a viable method for matching MIDI transcriptions to corresponding audio recordings in a database. Specifically, we could compute the alignment and confidence score between a query MIDI file and every recording and assign matches according to those recordings which produced high confidence scores. Unfortunately, the proposed method is much too inefficient for searching all but the smallest databases. We explore this issue and develop methods for mitigating it in the following chapters.

Chapter 5

Learning an Efficient Representation for Dynamic Time Warping

The system developed in chapter 4 produces a confidence score which proved to be extremely reliable at determining whether or not a MIDI file was a transcription of a given audio file. The goal of this thesis is to match a large collection of 178,561 MIDI files to corresponding audio recordings. We cannot rely on any source of metadata from these MIDI files (see section 1.2), so we must instead utilize a *content-based* comparison scheme like the one proposed in chapter 4. To maximize the chances of finding a matching audio recording for a given MIDI file, we need to use a large and comprehensive pool of audio files. We use the 994,960 7digital preview clips (Schindler et al., 2012) corresponding to the Million Song Dataset (MSD) (Bertin-Mahieux et al., 2011), which consist of (typically) 30 second portions of recordings from the largest standard research corpus of popular music. A complete search for matches could thus involve 994,960 alignments for each of our 178,561 MIDI files,

resulting in 177,661,052,560 comparison.

The method proposed in chapter 4 is much too inefficient to be used in this manner. Under this approach, aligning a typical-length MIDI file to a single audio recording from the MSD takes on average about 247 milliseconds on an Intel Core i7-4930k processor when using an extremely efficient LLVM-compiled DTW routine and the optimized distance matrix calculation of `scipy` (Jones et al., 2014). Matching our entire collection of MIDI files using this approach would therefore take approximately

$$\frac{(.247 \text{ seconds})(178{,}561 \text{ MIDI files})(994{,}960 \text{ audio files})}{(60 \text{ seconds})(60 \text{ minutes})(24 \text{ hours})(365 \text{ days})} \approx 1{,}391 \text{ years} \tag{5.1}$$

Clearly, a more efficient approach is warranted.

In fact, the problem of finding the sequence in a large database (here, a collection of audio recordings) which is most similar under DTW distance to a query (here, a MIDI file) is common in the field of data mining (Berndt and Clifford, 1994). Unsurprisingly, a variety of methods have been proposed for speeding up this task to make it feasible for very large databases. A pervasive framework is "pruning", where a computationally efficient operation is used to discard a large portion of the database and the remaining entries are compared using a more expensive, but highly accurate, method. Under certain assumptions, there are methods which can achieve *exact* pruning (i.e. without ever erroneously discarding the correct match) of large-scale nearest-neighbor DTW search. (Rakthanmanon et al., 2012) provides a thorough overview of these techniques and shows that for some problems they can allow exact search of databases with trillions of data points. However, one of the core assumptions of these methods is that the query is always a subsequence of its correct match, and equivalently that the length of the aligned sequence is known a priori, which does not hold in the setting of MIDI-to-audio matching. In fact,

because the 7digital MP3s are truncated preview clips and MIDI files are usually complete transcriptions, for our problem we nearly always have the opposite case where the correct match is a subsequence of the query. We therefore cannot leverage these pruning methods for our problem.

One of the reasons DTW-based search is so expensive is that it has $\mathcal{O}(MN)$ complexity, where M and N are the lengths of the two sequences being compared. Clearly, then, DTW can be made quadratically more efficient by decreasing the number of elements in the sequences being compared. For example, (Keogh et al., 2001; Yi and Faloutsos, 2000) proposed downsampling sequences by computing their average over non-overlapping blocks, which for some tasks provided orders of magnitude speedup without dramatically affecting accuracy. Similarly, (Salvador and Chan, 2007) propose an iterative procedure for making DTW more efficient which starts by finding the alignment path based on a low-resolution estimate of the sequences being compared, and then refines this estimate at higher and higher resolutions. The appropriateness of downsampling depends partially on the extent to which the sequences are oversampled (i.e. having an unnecessarily high correlation between consecutive sequence elements).

Computing the DTW distance between two sequences also potentially involves computing the pairwise Euclidean distance between all of their elements. In data mining tasks, entries in sequences are often one-dimensional, so the cost of this operation is usually not considered independent of the overall cost of DTW. For the time-frequency representation we utilize in chapter 4, our sequences consist of 48-dimensional feature vectors. As a result, in our problem setting computing the pairwise distance matrix is actually more expensive than finding the lowest-cost path through it because each entry in the distance matrix requires at least D operations, where D is the feature dimensionality. Computing the distance matrix similarly

receives quadratic speed gains when the sequences are downsampled, but also can be made faster by using a lower-dimensional or otherwise more efficient feature representation.

In addition, the "raw" representation in which data is initially provided may not produce the most effective measure of similarity under the Euclidean distance utilized by DTW. More concretely, while the audio-to-synthesized MIDI CQT comparison utilized in chapter 4 proved effective, it may be that we are able to transform the individual feature vectors to a space where similarity is better represented so that matching MIDI files and audio recordings have a smaller DTW distance. Recently, the framework of "representation learning" has proven to be effective for this type of problem (Bengio et al., 2013). Representation learning utilizes prior knowledge, such as a collection of pairs of known-similar and dissimilar elements, to learn a mapping to a more effective transformed space. This process is also capable of producing "multimodal" mappings, i.e. transformations which allow the comparison of heterogeneous data.

From the above discussion, it is clear that mapping very long sequences of high-dimensional feature vectors to shorter, lower-dimensional sequences such that similarity is preserved would provide substantial gains when comparing sequences with DTW. Motivated by this possibility, in this chapter we propose a system with the following capabilities:

Maps to a Hamming space: By replacing continuous-valued feature vectors with binary vectors in an embedded Hamming space, computing the distance between a pair of feature vectors simplifies to an exclusive-or followed by a single POPCNT SSE4.2 operation (Intel, 2007), which substantially speeds up distance matrix calculation.

Downsamples sequences: Rather than creating a one-to-one correspondence between the original feature vectors and mapped binary vectors, groups of subsequent feature vectors are mapped to a single binary vector, giving a quadratic increase in efficiency.

Preserves similarity: The system is trained with an objective which seeks to produce a mapping where aligned feature vectors from matching sequences have a small Hamming distance in the embedded space, and non-matched feature vectors have a large distance.

Learns its representation: Our approach is entirely data-driven, which allows it to adapt to different problem settings including multimodal data, as we show in section 5.2.3.

In the following section, we describe our model and training approach in detail. In section 5.2, we test our system's accuracy on the task of matching MIDI files to the Million Song Dataset. Finally, we discuss possibilities for improvement in section 5.3.

5.1 Learning to Downsample and Hash

As a high-level overview, our system will learn a mapping from sequences of feature vectors to downsampled sequences of binary vectors in a Hamming space where similarity is preserved based on a training set of matched and aligned sequences. This training set can be constructed by obtaining a collection of sequences in which matching pairs are known, and then using DTW to find the optimal alignment of feature vectors in matching sequences. From this data, we extract groups of sequentially co-occurring feature vectors to construct $\mathcal{P}$, such that $(x, y) \in \mathcal{P}$ indicates that x is a group of feature vectors which is aligned to the group y from a

matching sequence. For example, in the context of MIDI to audio matching, x will be subsequent constant-Q spectra of a synthesized MIDI which are aligned in time to y, consisting of co-occurring constant-Q spectra from a matching audio recording. We then construct $\mathcal{N}$, a set of "dissimilar" pairs, by repeatedly randomly choosing two pairs $(x_1, y_1), (x_2, y_2) \in \mathcal{P}$ and swapping to obtain $(x_1, y_2), (x_2, y_1) \in \mathcal{N}$.

Given this training data, our goal is to map sequences of feature vectors to sequences of binary vectors in a Hamming space where groups of sequentially co-occurring vectors in $\mathcal{P}$ have a small distance and groups in $\mathcal{N}$ have a large distance. Motivated by the multimodal hashing technique of (Masci et al., 2014), we use the following objective function to measure the quality of the mapping:

$$\mathcal{L} = \frac{1}{|\mathcal{P}|} \sum_{(x,y)\in\mathcal{P}} \|f(x) - g(y)\|_2^2 + \frac{\alpha}{|\mathcal{N}|} \sum_{(a,b)\in\mathcal{N}} \max(0, m - \|f(a) - g(b)\|_2)^2 \quad (5.2)$$

where f and g are learned nonlinear functions, α is a parameter to control the importance of separating dissimilar items, and m is a target separation of dissimilar pairs. The first term in the loss function encourages the embedded distance between similar pairs to be small and the second term encourages dissimilar pairs to have a distance of at least m. More specifically, if $x, y \in \mathbb{R}^{G\times D}$ where G is the number of subsequent feature vectors which are grouped together (and therefore the downsampling ratio) and D is the feature dimensionality, then $f(x), g(y) \in \mathbb{H}^E$ where $\mathbb{H}^E$ is the Hamming space of dimensionality E. This system is then optimized over $\mathcal{P}$ and $\mathcal{N}$ to adjust the parameters of f and g such that equation (5.2) is minimized. The use of two different functions f and g allows for different mappings to be learned for different types of data; we expect this to be beneficial for our problem setting because synthesized MIDI CQTs and real-world audio CQTs may have different characteristics. This also allows us to experiment with multimodal data, i.e. cases where x and y come

from different, and potentially non-comparable, feature spaces. A visualization of this process is shown in figure 5.1.

After minimizing equation (5.2) on a training set, sequences of feature vectors in either modality can then be mapped to downsampled sequences of hash vectors using f and g. Once mapped, we can perform DTW much more efficiently to compute a distance between hash sequences, where the Euclidean distance calculation inherent in DTW has been replaced by Hamming distance. More specifically, we expect a speedup by a factor of up to G^2D, because we have effectively downsampled both sequences by a factor of G and have replaced the D-operation Euclidean distance with a Hamming distance calculation. In practice, the resulting binary vectors can be represented efficiently as unsigned integers, whose Hamming distance can be computed with an exclusive-or operation followed by a call to the POPCNT SSE4.2 instruction (Intel, 2007). The exclusive-or compares bit-by-bit whether the entries of the binary vectors are equal, and the POPCNT implements a "population count" or "hamming weight" calculation, which simply counts the number of nonzero bits in the result. This process is visualized in figure 5.2. Ideally, utilizing DTW to compare sequences which have been mapped to this efficient representation will effectively recover matching and non-matching sequences.

Given our training procedure, we now need to choose appropriate models for the hashing functions f and g. Neural networks are an attractive choice, because they can in principle be optimized with respect to unusual loss functions like the one defined in equation (5.2). In addition, the suite of tricks discussed in section 2.1.3.2 can help ensure effective results.

In (Masci et al., 2014), dense feed-forward neural networks are used for the learnable functions f and g. We could potentially utilize this type of network for our hashing and downsampling procedure by concatenating input $x, y \in \mathbb{R}^{G \times D}$ into

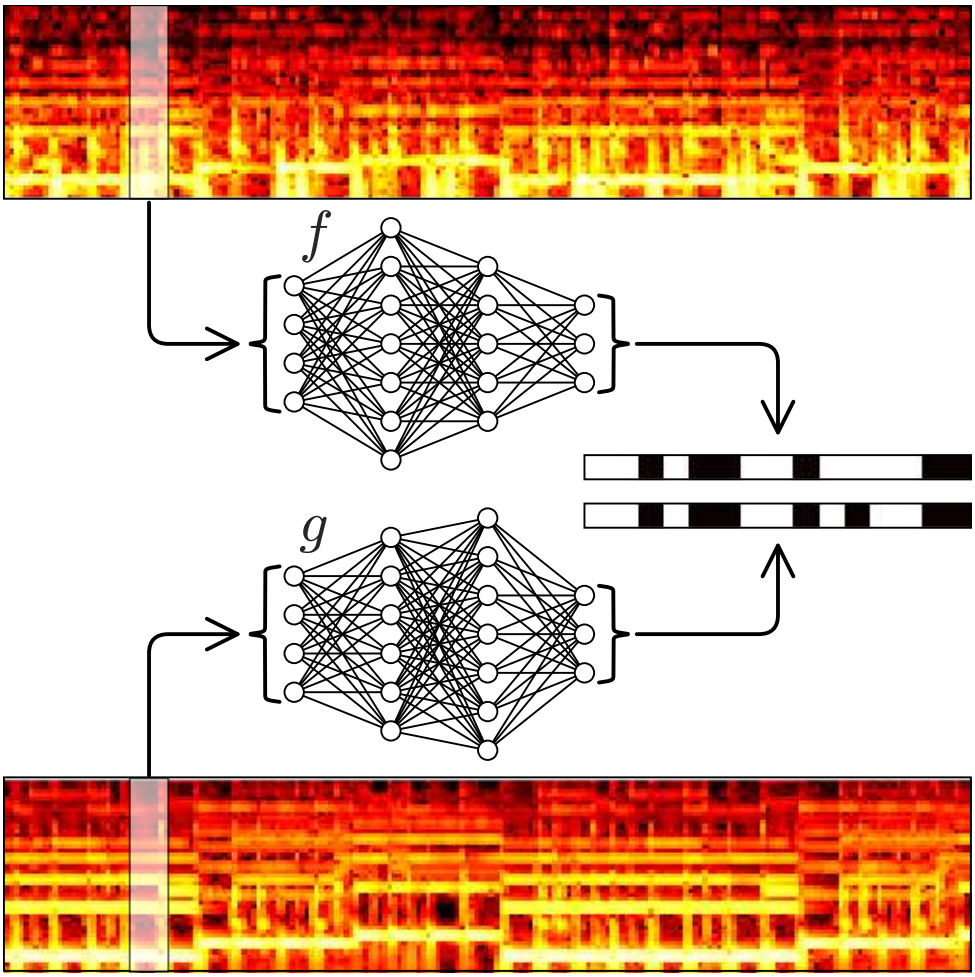

Figure 5.1: Diagram visualizing how subsequent groups of feature vectors are mapped to binary vectors. Matching sequences of feature vectors which have been aligned in time are shown at the top and bottom. Co-occurring groups of feature vectors, highlighted in light boxes, are passed through two learned nonlinear functions f and g (visualized as neural network diagrams). The functions then output a single hash vector each, which are ideally close together in the embedded Hamming space because they represent co-occurring feature vectors from matched and aligned sequences. For groups of feature vectors from non-matching sequences, the networks are trained to output binary vectors with a large Hamming distance.

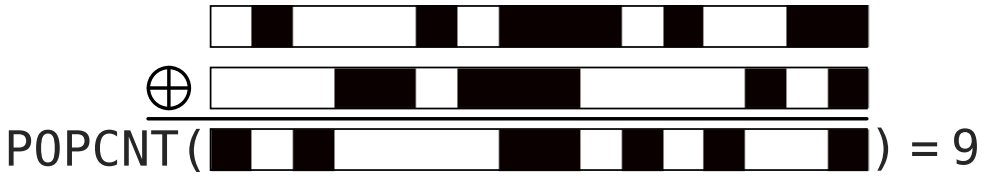

Figure 5.2: Computing the Hamming distance between two binary vectors via an exclusive-or followed by a POPCNT instruction. The exclusive-or of the top two vectors produces the bottom vector, which is 1 where the input vectors differ and 0 where they match. The result of the exclusive-or is passed to the POPCNT SSE4.2 instruction, which simply counts the number of non-zero bits in its input. These two steps therefore can produce the distance between two binary vectors in two machine instructions.

single-dimensional vectors in $\mathbb{R}^{GD}$. However, we argue that convolutional networks (discussed in section 2.1.3.4) are a better choice for sequential data for the following reasons: Sequential data has at least one ordered (temporal) dimension, and in most natural data, the strongest dependencies along the temporal axis are localized. This is exactly the kind of structure convolutional networks are designed to efficiently exploit. Furthermore, the pooling layers which are a common ingredient in convolutional networks provide a natural method of downsampling the sequences. They also allow the network's output a given point in time to depend on a large "receptive field" of the input. In our problem setting, this means that a given hash vector produced by f or g will be affected not only by the corresponding group of G feature vectors but also on a window of feature vectors surrounding it. For the above reasons, we utilize convolutional networks for the rest of this chapter.

5.2 Experiment: MIDI to MSD Matching

To determine the effectiveness of this approach on the task of matching MIDI files to the MSD, we evaluated its effectiveness on a smaller surrogate problem. As an overview, this experiment proceeded as follows: First, we assembled a collection of known-correct MIDI-MSD matches. We then aligned these pairs using the "gold-standard" system described in section 4.4.1 and used a subset of the matched and aligned pairs to train our hashing model. Finally, we utilized the trained model to produce hash sequences for the entire MSD and a held-out subset of MIDI files from our collection of known-correct matches. Our approach was then evaluated based on how accurately it was able to recover the correct match in the MSD by nearest-neighbor DTW search utilizing the downsampled hash sequences. To explore the ability of our approach to handle multimodal data, we repeated this experiment using piano roll matrices to represent MIDI files rather than constant-Q spectrograms. We also compared our technique to a simple baseline, to ensure that our learning-based approach is beneficial.

5.2.1 Preparing Data

For evaluation, we need a collection of ground truth MIDI-audio pairs which are correctly matched. To train our hashing model, we further require a collection of *aligned* MIDI and audio files, to supply the matching groups of feature vectors from each domain that will be used to train our model. To obtain MIDI-audio pairs, we first separated a subset of MIDI files from our 178,561-file collection for which the directory name corresponded to the song's artist and the filename gave the song's title. The resulting metadata needed additional canonicalization; for example, "The Beatles", "Beatles, The", "Beatles", and "The Beatles John Paul Ringo George" all appeared as artists. To normalize these issues, we applied some manual text

processing and resolved the artists and song titles against the Freebase (Bollacker et al., 2008) and Echo Nest[1] databases. This resulted in a collection of 17,256 MIDI files for 10,325 unique songs, which we will refer to as the "clean MIDI subset".

We will leverage the clean MIDI subset in two ways: First, to obtain ground truth pairings of MSD/MIDI matches, and second, to create training data for our hashing scheme. The training data does not need to be restricted to the MSD, and using other sources to increase the training set size will likely improve our hashing performance, so we combined the MSD with three benchmark audio collections: CAL500 (Turnbull et al., 2007), CAL10k (Tingle et al., 2010), and uspop2002 (Berenzweig et al., 2004). To match these datasets to the clean MIDI subset, we used the Python search engine library `whoosh`[2] to perform a fuzzy matching of their metadata. This resulted in 25,707 audio/MIDI file pairs corresponding to 5,249 unique songs.

Fuzzy metadata matching is not enough to ensure that we have MIDI and audio files with matching content; for instance, the metadata could be incorrect, the fuzzy text match could have failed, the MIDI could be a poor transcription (e.g., missing instruments or sections), and/or the MIDI and audio data could correspond to different versions of the song. We therefore utilized the alignment technique developed in section 4.4.1, which was empirically shown to produce a confidence score which reliably reports whether the content of a MIDI file matches a given audio file. Conveniently, utilizing an alignment system to verify matches also provides the aligned sequences we need for training our hashing system. Based on the real-world results presented in section 4.5, we decided to consider a match "correct" when its confidence score was above 0.5; for the 500 alignments annotated in section 4.5, this produced the correct label for all but 1 of the failed matches and for the majority of

[1] `http://developer.echonest.com/docs/v4`
[2] `https://github.com/dokipen/whoosh`

the correct matches. Retaining all alignments with confidence scores higher than this threshold resulted in 10,094 successful matches.

For a fair evaluation, we exclude items used in training from the evaluation of our system, thus we split the successful alignments into three parts: 60% to use as training data, 5% as a validation set used for rapid evaluation of models as they are being trained, 10% as a "development set" to tune the content-based matching system, and the remaining 25% to use for final evaluation of our system. Care was taken to split based on *songs*, rather than by entry (since some songs appear multiple times).

5.2.2 System Specifics

Training the hashing model involves presenting training examples and backpropagating the gradient of equation (5.2) through the model parameters; we therefore need to specify the data format, model architecture, and training process. As a feature representation, we used the same one chosen in section 4.4.1, i.e. log-magnitude constant-Q spectrograms consisting of spectra with 12 bins per octave from MIDI note C2 (65.4 Hz) to B6 (987.8 Hz) computed every 46.4 milliseconds. We utilized `pretty_midi` (Raffel and Ellis, 2014) and `fluidsynth`[3] to synthesize MIDI files and `librosa` (McFee et al., 2015b,c) to compute features. Because utilizing a cosine distance for comparing spectra was consistently beneficial for alignment (see section 4.3), we L^2-normalized each feature vector. In order to improve convergence, we also standardized the feature dimensions of all data utilizing statistics from the training set.

For efficiency, we trained the networks using minibatches of training examples; each minibatch consisted of 50 sequences obtained by choosing a random offset for

[3] `http://fluidsynth.org`

each training sequence pair and cropping out the next 100 spectra. For $\mathcal{N}$, we simply presented the network with length-100 subsequences chosen at random from different songs. Each time the network had iterated over minibatches from the entire training set (one epoch), we repeated the random sampling process. For optimization, we used RMSProp (Tieleman and Hinton, 2012), described in section 2.1.4.5. After each 100 minibatches, we computed the loss over the validation set. If the validation loss was less than 99% of the previous lowest, we trained for 1000 more iterations (minibatches).

While the loss computed over the validation set is a reasonable indicator of network performance, its scale will vary depending on the α and m regularization hyperparameters. To obtain a more consistent metric, we also computed the distribution of distances between the hash vectors produced by the network for the pairs in $\mathcal{P}$ and those in $\mathcal{N}$. We then used the Bhattacharyya coefficient (Bhattacharyya, 1943) to compute the separation of these distributions to obtain a more direct measure of network performance.

In each modality, the hashing networks have the same architecture: A series of alternating convolutional and pooling layers followed by a series of fully-connected layers. While the exact specification of this architecture will affect performance, the most impactful design choice for our problem setting is the extent to which the temporal axis is downsampled by pooling layers. Following the common convention of pooling by 2, each pooling layer will result in sequence lengths being halved and the resulting DTW calculation being faster by up to a factor of 4. However, too much downsampling may result in a loss of precision, particularly if the resulting sequences become extremely short. We therefore chose to utilize a total of three pooling layers, which corresponds to downsampling by a factor of 8. This resulted in the 30-second 7digital preview clips being mapped to length-80 hash sequences,

which provided enough information for high precision while still achieving up to a factor of 64 speedup when computing DTW.

For the convolutional front-end of our networks, we experimented with a variety of architectures and chose the following: Each of the three max-pooling layers were preceded by two convolutional layers, each with 3×3 filters (where the first dimension is time and the second is constant-Q frequency bin). The use of a larger number of smaller convolutional filters is advocated by Simonyan and Zisserman (Simonyan and Zisserman, 2014), who argue that this structure improves the network's modeling capabilities while maintaining receptive field size and requiring fewer parameters. The convolutional layers in each of the three "groups" had 16, 32, and 64 filters, respectively. We padded the input to each convolutional layer so that the output size was the same as the input size. All of the max-pooling layers utilized 2×1 pooling with a stride of 2. The pooling size of 2×1 is advocated for spectrograms in (Humphrey and Bello, 2012) and results in only the temporal axis, not the frequency axis, being downsampled; we found this to work better than the more standard 2×2 pooling commonly used in image processing networks.

After the convolutional layers, we utilized two dense layers with 2048 units each, followed by the output layer. The fully-connected layers were applied independently over the temporal axis to the output of the convolutional portion of the network. In our networks, all layers except the output use rectifier nonlinearities; as in (Masci et al., 2014), the output layer uses a hyperbolic tangent. We can then obtain binary hash vectors by testing whether each output unit is greater or less than zero. We chose 32 bits as the dimensionality of our embedded Hamming space, since 32 bit values are efficiently manipulated as unsigned integers. For initial weight parameter values, as suggested in (He et al., 2015), we used normally-distributed random variables with a mean of zero and a standard deviation of $\sqrt{2/n_{in}}$, where n_{in} is the

number of inputs to each layer. All bias values were initialized to 0. Our model was implemented using `theano` (Bergstra et al., 2010; Bastien et al., 2012; Al-Rfou et al., 2016) and `lasagne` (Dieleman et al., 2015).

For regularization, we found it beneficial to add a squared L^2 penalty to the network's output prior to thresholding. This effectively encourages the network to produce values near 0, which we believe improves convergence due to the fact that the gradient of the hyperbolic tangent nonlinearity vanishes away from 0. This is in contrast to the intuition provided in (Masci et al., 2014) for using a hyperbolic tangent, where it is argued that saturating the nonlinearity will more closely simulate a binary embedding. To control the amount of regularization, we multiplied this L^2 penalty by a scalar hyperparameter β before adding it to equation (5.2).

To ensure good performance, we optimized all model hyperparameters using the Bayesian optimization framework proposed in (Snoek et al., 2012). We optimized over the convolutional/pooling layer architecture, the RMSProp learning rate and decay parameters, and the α, β and m regularization parameters of the loss function and the output L^2 penalty. As a hyperparameter optimization objective, we used the Bhattacharyya coefficient as described above. The best performing network had the structure we described above, a learning rate of 10^{-5} with an RMSProp decay parameter of 0.26, $\alpha = 5.33$, $m = 4$, and $\beta = 0.05$. This hyperparameter configuration achieved a Bhattacharyya coefficient of 0.385.

Beyond those described above, we also experimented with a number of other regularizers, model architectures, and optimization schemes. For posterity, we report those here:

- We experimented with a larger 5×12 filter size in the first layer as used in (Raffel and Ellis, 2015b) with single 3×3 convolutional in the second and third

layers, but found this structure resulted in slightly worse performance despite having a similar receptive field.

- We tested the use of dropout (Hinton et al., 2012) in the fully-connected layers, but it slightly decreased performance.
- We tried adding a "stress" (Kruskal, 1964) term to the loss function to encourage the networks' outputs to preserve the distances of their input, but found it hurt performance.
- We also tried the hash entropy-encouraging loss term proposed in (Yang et al., 2015, equation (3)) but it did not significantly affect the resulting hash distribution.
- In early experiments, we utilized fully-connected networks with no convolutional layers, but quickly achieved better results after including the convolutional front-end.
- We experimented with a variety of architectures for the fully-connected portion of the network, including using 1, 2, or 3 hidden layers, with 256, 512, 1024, 2048, or 4096 layers; 2 layers with 2048 units was the largest configuration which could fit in memory and incidentally produced the best performance.
- Apart from RMSProp, we also experimented with Nesterov's accelerated gradient (Nesterov, 1983) and Adam (Kingma and Ba, 2015) (see section 2.1.4.3 and section 2.1.4.6) but found that RMSProp produced the best results.

5.2.3 Adapting to Multimodal Data

Our proposed system is structured to learn mappings from two different modalities thanks to the fact that two separate networks f and g are trained. Utilizing the

shared constant-Q spectrogram representation described in the previous section does not test the multimodal capabilities of our model because it allows MIDI files and audio recordings to be directly compared (as demonstrated in chapter 4). We therefore also evaluated the use of a "piano roll" representation for the MIDI data. A piano roll matrix P is constructed such that $P[i, j] = v$, where v is the total velocity of all notes on all instruments playing with pitch j at the time corresponding to the i^{th} feature vector. This representation can be readily extracted from the MIDI file itself (i.e. without synthesis), and contains substantially less information than the CQT. It is, however, still a "time-frequency" matrix, so it is not truly multimodal in the strictest sense.

For ease of comparison, we utilized the exact same system as outlined in section 5.2.2 except that we replaced the constant-Q spectrograms of synthesized MIDI files with piano roll matrices, computed over the same timebase and range of semitones. Specifically, after matching and aligning files from the clean MIDI subset, we computed piano roll matrices using the `get_piano_roll` method of the `pretty_midi` library (Raffel and Ellis, 2014). As with the constant-Q spectrograms, we also L^2 normalized each feature vector in the piano roll matrices and standardized the feature dimensions using statistics from the training set. We then carried out a separate hyperparameter optimization, because we don't necessarily expect the same system design to perform best using both feature representations. The best performance was attained with a learning rate of 0.00011, an RMSProp decay parameter of 0.90, $\alpha = 5.72$, and $m = 5$, and $\beta = 0.036$, producing a Bhattacharyya coefficient of 0.428.

5.2.4 Baseline

In order to verify whether our hashing system is beneficial, we also evaluated a simple learning-free baseline method. This approach is inspired by the "piecewise

aggregate approximation" approach proposed in (Keogh et al., 2001; Yi and Faloutsos, 2000), which downsamples a sequence by computing the average of feature vectors within non-overlapping blocks of fixed length. Given a constant-Q spectrogram, our baseline method first computes the average spectrum over non-overlapping blocks of 8 spectra, which achieves the same downsampling ratio as our convolutional network models. In order to produce binary vectors, we first computed the mean value in each feature dimension among all sequences in our training set, and then thresholded the downsampled sequences about this value. More explicitly, given a sequence $X \in \mathbb{R}^{M \times D}$ which consists of M D-dimensional feature vectors, we compute

$$\hat{X}[n,d] = \begin{cases} 1, \left(\frac{1}{G} \sum_{m=Gn}^{G(n+1)} X[m,d] \right) \geq \bar{X}[d] \\ 0, \text{otherwise} \end{cases} \tag{5.3}$$

where G is the downsampling factor (8 in our experiments), $n \in \{1, \ldots, \lfloor \frac{M}{G} \rfloor\}$, $\hat{X} \in \mathbb{H}^{\lfloor \frac{M}{G} \rfloor \times D}$ is the resulting downsampled hash sequence, and

$$\bar{X}[d] = \frac{1}{|\mathcal{S}|} \sum_{k=1}^{|\mathcal{S}|} \frac{1}{M_k} \sum_{m=1}^{M_k} X_k[m,d] \tag{5.4}$$

where $X_1, \ldots, X_{|\mathcal{S}|} \in \mathcal{S}$ is the training set of sequences with $X_k \in \mathbb{R}^{M_k \times D}$. After thresholding our constant-Q spectrograms, this resulted in binary vectors in $\mathbb{H}^{48}$. For more direct comparison, and to enable the use of exclusive-or and POPCNT to measure Hamming distance, we simply discarded the bottom and top eight entries in each of the resulting binary vectors so that they instead were embedded in $\mathbb{H}^{32}$. Intuitively, the resulting downsampled hash sequences represent whether a given semitone had more or less energy than average over subsequent blocks of 8 frames (corresponding to 371.5 milliseconds).

5.2.5 Matching MIDI Files to the MSD

After training our hashing system as described above, the process of evaluating the effectiveness of matching MIDI files to the MSD proceeds as follows: First, we precompute downsampled hash sequences for every 7digital preview clip and every MIDI file in the test and development sets for which we know a priori the correct match(es) in the MSD. Then, we compute the DTW distance as described in section 2.2.3 between every audio and MIDI hash sequence, except that the Euclidean distance of equation (2.81) is replaced with Hamming distance. Ideally, for a given MIDI file the correct match in the MSD will have the smallest DTW distance. For brevity, we will refer to the CQT-to-CQT hashing model of section 5.2.2 as "DHS" (for **d**ownsampled **h**ash **s**equence), the piano roll-to-CQT model of section 5.2.3 as "DHS-piano", and the baseline system of section 5.2.4 as "TPAA" (for **t**hresholded **p**iecewise **a**ggregate **a**pproximation).

We tuned the parameters of the DTW cost calculation to optimize results over our "development" set of successfully aligned MIDI/MSD pairs. We found it beneficial to use a smaller value of $g = 0.9$. In chapter 4, we utilized the median pairwise distance for the "non-diagonal move" penalty for CQT-based distance matrices, but we found that this median was consistent across Hamming-distance matrices. Using a fixed value for the additive non-diagonal move penalty ϕ avoids the median calculation, so we chose a fixed $\phi = 15$, the median distance we observed. Similarly, we found that normalizing by the average distance value as defined in equation (4.5) did not help, so we skipped this step.

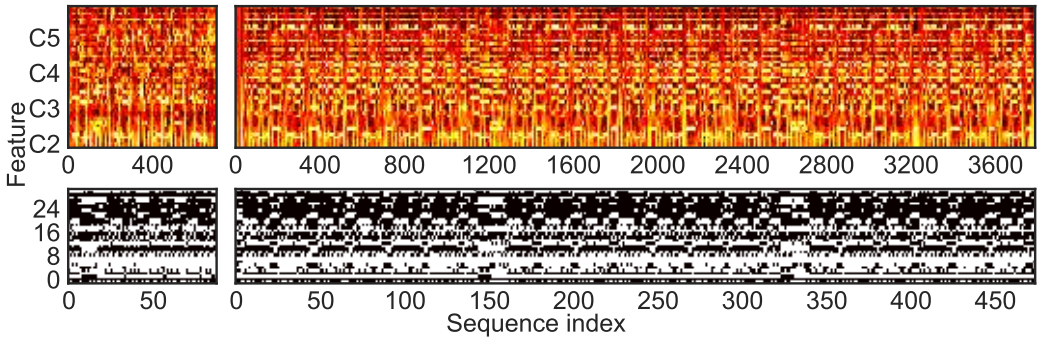

Figure 5.3: Example constant-Q spectrograms (top) and downsampled hash sequences (bottom) for a matched MSD entry (left) and MIDI transcription (right) of the song "I Wanna Be Your Lover" by Prince, which appeared in our development set. The downsampled hash sequences were created using the trained "DHS" model described in section 5.2.2. As utilized in our experiments, the constant-Q spectrograms are log-magnitude and L^2-normalized. For the spectrograms, the feature dimension corresponds to semitones; for the hash sequence, the Hamming space dimensions have no explicit meaning. The spectrogram utilizes a timebase of 46.4 milliseconds per sequence step, which the hash sequence downsamples by a factor of 8 to 371.5 milliseconds.

5.2.6 Results

5.2.6.1 Qualitative

Before diving into the empirical performance of our system, it is useful to get a qualitative idea of its behavior. By way of example, we show our model's output for an example pair of matching sequences from our development set in figure 5.3. While the hash sequences are not directly interpretable, they do appear to match the overall structure of the constant-Q spectrograms. The pairwise distance matrices and DTW path for these constant-Q spectrogram and downsampled hash sequence pairs can be seen in the first and second row of figure 5.4 respectively. Visualized in this manner, the hash sequence distance matrix clearly shows the structure of the song, including local repetitions as well as the two choruses at around 1/3 and 2/3 of the

way through the song. Furthermore, the hash sequence DTW operation recovered the same alignment path. The distance matrix produced by the "multimodal" system, shown in the third row, has similar structure although the DTW alignment chose the first rather than second chorus. At the bottom is the baseline TPAA distance matrix, which has less obvious structure than the other hash sequence representations.

An intuitive surrogate for performance is the distribution of distances between matching and non-matching pairs. Ideally, the distance between aligned feature vectors or embedded binary vectors in matched MIDI and audio files will be small, and will be large for non-matching pairs. The extent to which the distributions of matching and non-matching distances overlap is measured by our objective, the Bhattacharyya coefficient, but it is also useful to inspect the distributions themselves before and after undergoing downsampling and hashing for the various techniques we are evaluating. These distributions, for the raw CQT-to-CQT, piano roll-to-CQT, and each of the Hamming-space representations produced by the DHS, DHS-piano, and TPAA approaches are shown in figure 5.5.

The overlap between the matching and non-matching distributions of CQT-to-CQT distances in the top left shows that there is certainly some room for improvement in this representation. Below this, the distributions for the piano roll-to-CQT distances show that comparing these two representations is substantially less effective, although there is some trend towards smaller distances for matching feature vectors. At the bottom left, the distributions for raw (pre-thresholding) distances are shown. This plot gives a clear picture that representation learning is helpful, because there is virtually no overlap between matching and non-matching distances, and furthermore the Euclidean distance in the embedded space for matching sequence elements is generally very small. The effect of thresholding this representation is shown in the bottom right; thresholding clearly limits some of the discriminative power of the

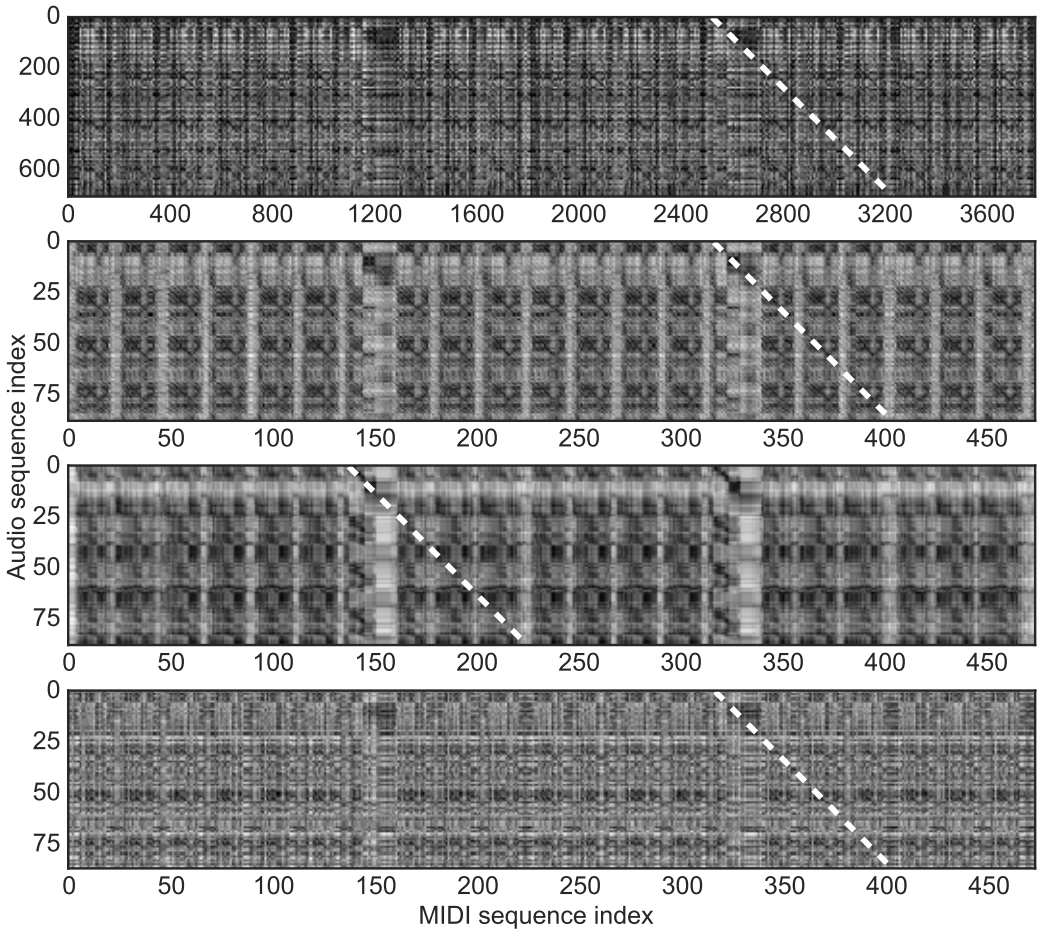

Figure 5.4: Distance matrices and DTW alignment paths (shown as dashed white lines) between the MSD entry and MIDI transcription shown in figure 5.3 utilizing different representations. At the top is the standard synthesized MIDI CQT-to-audio CQT distance matrix, as utilized in chapter 4. The second row shows the distance matrix for the downsampled hash sequences shown at the bottom of figure 5.3, produced by the DHS system. Below that is the distance matrix for hash sequences created utilizing the DHS-piano system. Finally, the bottom distance matrix was computed using the TPAA method.

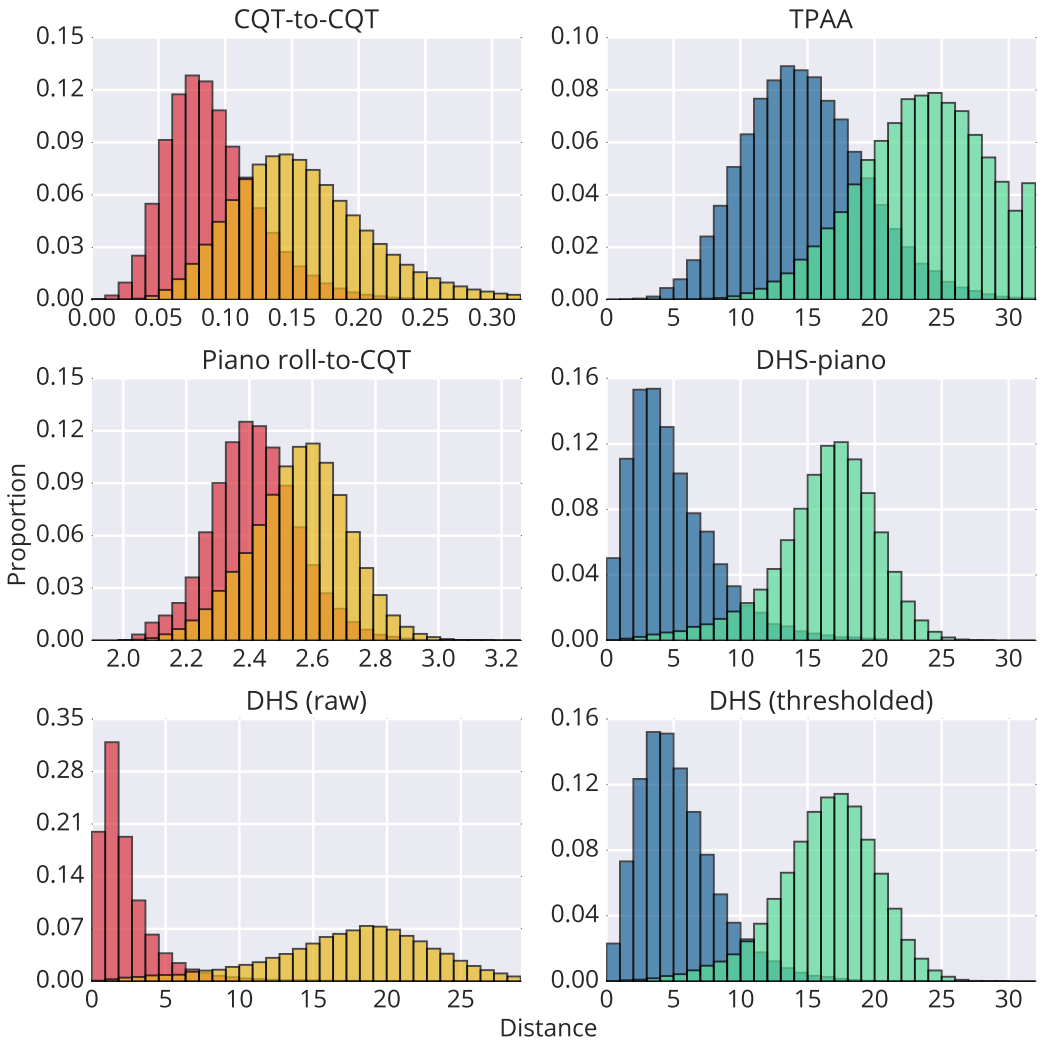

Figure 5.5: Distributions of distances for matching and non-matching sequence elements from our validation set for a variety of representations. Real-valued (squared Euclidean) distance distributions for matching and non-matching pairs are shown in red and orange respectively; Hamming distance distributions are shown in blue and green. From top left to bottom right are distance distributions for vectors produced by synthesized MIDI CQT-to-audio recording CQT, the TPAA method, MIDI piano roll-to-audio CQT, the thresholded output of the DHS-piano model using this "multimodal" data, the DHS model prior to thresholding, and the DHS model after thresholding.

representation, but nevertheless there is much less overlap in the distributions than there was for the "raw" CQT feature representation. Above, the Hamming distance distributions are shown for the model which learned to embed piano roll vectors; the separation of the distributions almost matches the CQT-to-CQT-based model, showing that our system is adaptable to differing input representations. Finally, in the top right, we show the distributions for our baseline TPAA method. Clearly, there is substantially move overlap between matching and non-matching embeddings in this representation.

5.2.6.2 Ranking Metrics

To evaluate our system using the known MIDI-audio pairs of our test set, we ranked MSD entries according to their hash sequence DTW distance to a given MIDI file, and determined the rank of the correct match. Because some MIDI files were matched to multiple MSD entries due to duplicates in the MSD, we made the forgiving choice of utilizing the best (i.e. smallest) rank among all correct matches for a given MIDI file. We also ignored all resulting hash sequences with less than 30 sequence elements; these generally corresponded to corrupt MP3 files and occasionally confounded the results. The mean reciprocal rank (MRR) across the test set for the three methods we evaluated is shown in the first row of table 5.1. This metric makes it clear that our learning-based system outperforms our baseline, producing a MRR of 0.704 compared to 0.270. We also observe a modest, but significant, drop in performance to an MRR of 0.580 when using piano rolls instead of constant-Q spectrograms of MIDI syntheses.

While the MRR provides a compact measure of performance, we believe a more intuitive empirical measure is the proportion of MIDI files in the test set whose correct match ranked under a certain threshold. We plot this proportion against the

Metric	**DHS**	**DHS-piano**	**TPAA**
Mean Reciprocal Rank	0.704	0.580	0.270
95% Rank Threshold	98	518	25,944

Table 5.1: Metrics measuring the performance of the different approaches we tried for creating downsampled hash sequences. The first row shows the mean reciprocal rank of the correct match across all entries in our test set. The second shows the threshold below which 95% of the test set had a smaller rank.

rank threshold in figure 5.6. A high-performance metric will yield a curve which approaches 100% very quickly, meaning that the majority of the correct matches in the test set were ranked highly. From a high level, then, we can see that our proposed method substantially outperforms the baseline, due to the fact a much larger proportion of the test set was highly ranked.

Nevertheless, using our learning-based approach for creating downsampled hash sequences resulted in only about 59.2% of the matches in our test set having the smallest DTW distance (i.e. having rank 1). This implies that we can't rely on the correct entry appearing as the top match among all the MSD tracks in practice. Some degree of inaccuracy can be attributed to the fact that our hashing model is not perfect (as shown in figure 5.5) and that the MSD is very large, containing many possible decoys. In a relatively small proportion of cases, the MIDI hash sequence ended up being very similar to many MSD hash sequences, pushing down the rank of the correct entry.

Given that we cannot reliably assume the top hit is the correct MSD entry, it is more realistic to look at our system as a pruning technique; that is, it can be used to discard MSD entries which we can be reasonably confident do not match a given MIDI file. We therefore propose that a useful metric is to find the rank threshold below which we can be highly confident that the correct match appears, which we

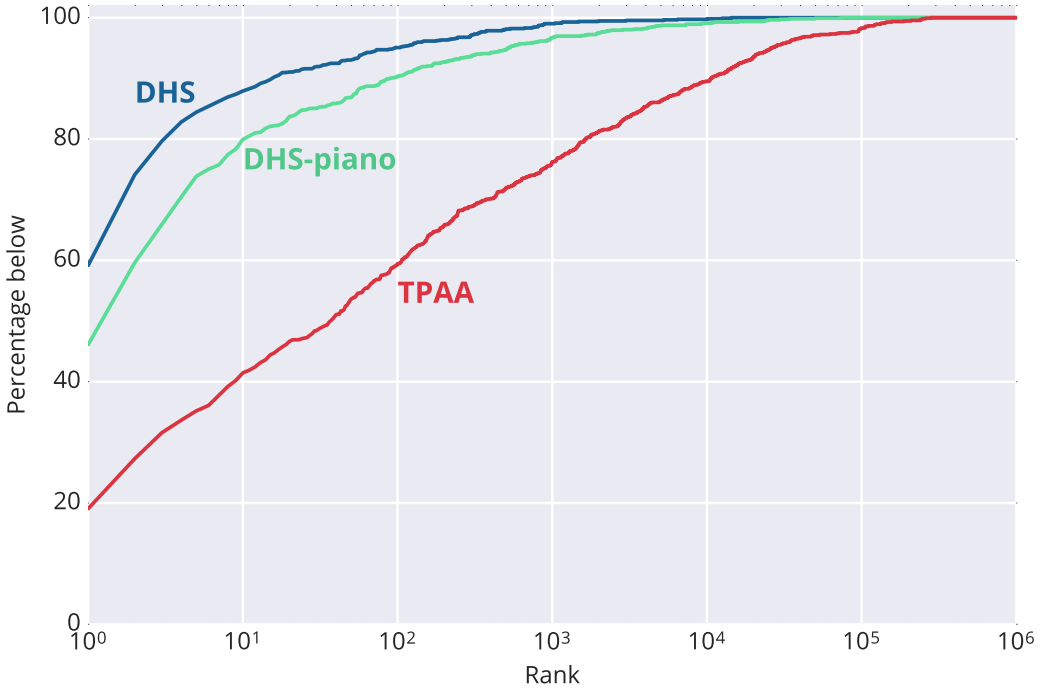

Figure 5.6: Percentage of MIDI files in the test set whose correct match in the MSD ranked better than a given threshold, using each of the methods we evaluated.

report for our different methods in the second row of table 5.1. Specifically, we computed the threshold N at which 95% of the correct matches in the test set had a rank better (i.e. lower) than N. Given this high-confidence rank threshold N, we can compute the hash sequence DTW distance between a query MIDI and all MSD entries, discard all entries in the MSD except those with the N-smallest distances, and then perform the more precise and more expensive method of chapter 4 to find the best match in the remaining candidates. Inspecting the resulting values of N for the methods we evaluated, we see that we can discard the vast majority (all but 98 entries, or over 99.99%) of the MSD with high confidence using the DHS model. Once again, utilizing piano roll matrices incurs a modest decrease in precision, and the baseline TPAA method has substantially worse performance.

5.3 Discussion

Pruning methods are valuable only when they are substantially faster than performing the original computation; fortunately, our approach is orders of magnitude faster: On an same Intel Core i7-4930k processor, calculating a Hamming distance matrix between downsampled hash sequences is on average about 1,500 times faster than computing the CQT cosine distance matrix (142 microseconds vs. 225 milliseconds) and computing the DTW score is about 62 times faster (371 microseconds vs. 23 milliseconds). These speedups can be attributed to the fact that computing an exclusive-or and a POPCNT is much more efficient than computing the cosine distance between two vectors and that, thanks to downsampling, our hash-based distance matrices have $\frac{1}{64}$ of the entries of the CQT-based ones. In summary, a straightforward way to describe the success of our system is to observe that we can, with high confidence, discard over 99.99% of the entries of the MSD by performing a calculation that takes about as much time as matching against 0.2% of the MSD. In addition, our learning-based approach provides a huge improvement over simply thresholding the raw feature values.

We identify a variety of potential avenues for improving our approach. First, our simple method of randomly swapping matching pairs to construct our dissimilar pair set $\mathcal{N}$ may not be the most effective approach. Preferentially choosing pairs on which our model is having difficulty, as done in (Schroff et al., 2015), or using a curriculum learning approach which presents more and more difficult examples to the network over time (Bengio et al., 2009) could prove beneficial. Furthermore, we are interested in trying the various other loss functions which have been proposed for similarity-preserving embedding, such as the "coscos" loss from (Kamper et al., 2016) or the k-means style loss from (Hershey et al., 2016).

Despite our system's efficiency, we estimate that comparing each of the MIDI files

in our 178,561 file collection to the entire the MSD would still take about 1,000 days. We therefore still require an even more efficient technique. The main bottleneck is that it still requires DTW to compare hash sequences; if we could avoid this quadratic-complexity operation we could potentially develop an even faster pruning method. We explore this possibility in the following chapter.

Chapter 6

Pruning Subsequence Search by Embedding Sequences

The DTW-based approaches for computing sequence similarity we have discussed so far have quadratic complexity in the sequence length because they involve multiple pairwise operations between each element in the sequences being compared. While the downsampling and binary embedding approach proposed in chapter 5 can make DTW much more efficient by lowering the number of sequence elements and making the pairwise comparisons faster, it is nevertheless still quadratic in complexity. When datapoints can be represented as single fixed-length vectors, rather than as sequences of vectors, comparing two datapoints is effectively linear-time in the vector dimensionality. This makes the task of matching a query to its most similar entry in a database extremely efficient. An obvious question, then, is whether we can avoid the operation of comparing sequences for the majority of the database by comparing fixed-length vector surrogates for the sequences whose distance approximates their similarity.

Motivated by this question, we propose a learning-based system for mapping

variable-length sequences of feature vectors to a fixed-length embedded space where sequence similarity is approximated by Euclidean distance. This allows us to pre-compute embeddings for all entries in a large database of sequences, and rapidly estimate similarity by embedding a query sequence and computing its Euclidean distance to all of the database embeddings. As a result, this operation does not become less efficient for longer or higher-dimensional sequences, i.e. it is effectively invariant to sequence length and feature vector dimensionality. Our approach utilizes a novel attention-based neural network model which can adapt to any problem setting according to the training data provided.

In the next section we discuss embedding and attention-based neural networks, which we use to motivate our proposed model. In section 6.2, we evaluate the effectiveness of our approach on our exemplary task of matching MIDI files to entries in the Million Song Dataset. Finally, we discuss results and possibilities for improvement in section 6.3.

6.1 Embedding Sequences with Attention

6.1.1 Embedding

The idea behind "embedding" is attractive: Given data in a representation which is either inefficient or does not readily reveal factors of interest, can we produce a mapping to fixed-length vectors that possess these desired properties? Because many simple machine learning methods are particularly effective given data where Euclidean distance corresponds to some notion of similarity, embedding can make data more amenable to learning and comparison.

As a concrete example, in (Schroff et al., 2015), a neural network architecture is used to embed images of faces such that images of the same person have small

pairwise distances in the embedded space and images of different people have large distances. The authors were able to achieve state-of-the-art results on the "labeled faces the wild" dataset by simply thresholding the resulting distances to denote "same person" or "different person". In addition, the embedded representation produced qualitatively useful results when performing unsupervised clustering using k-means. Alternatively, (Yang et al., 2015) proposed a supervised method for learning Hamming-space embeddings for images which reflect salient characteristics of the images being embedded. This allows for rapid "semantic" nearest neighbor queries to be made over large image databases.

For text, a striking application is word2vec, which is a family of models for mapping words to a Euclidean space with desirable properties (Mikolov et al., 2013). For example, the difference between "China" and "Beijing" in the embedded space is roughly equivalent to the difference between "Rome" and "Italy". These embeddings are typically learned in a unsupervised manner based on a word's context.

Most relevant to this thesis, embedding has been applied to the task of large-scale DTW search: (Papapetrou et al., 2011) proposes a technique which maps variable-length sequences of feature vectors to a fixed-length embedded space where DTW distance is approximated by Euclidean distance. The embedding is constructed by pre-computing the DTW distance between each sequence in the database and a small collection of "reference" sequences that are chosen to optimize retrieval accuracy. Then, to match a query sequence to the database, its DTW distance is computed against the reference sequences, and the resulting vector of DTW distances is matched against the database of embeddings using a standard Euclidean distance-based nearest-neighbor search. The resulting algorithm is approximate, but provided state-of-the-art results both in terms of accuracy and speed. This technique relies on the assumption that the query is a subsequence of its correct match, and the training

procedure involves repeatedly aligning a training set of sequences to the database. These restrictions make it invalid in some cases, such as our MIDI-to-MSD matching problem where the correct database entry may be a subsequence of a given query and where it is computationally unreasonable to repeatedly align sequences to the entire database in the training process.

The idea of embedding has also been utilized in other sequential data problems. For fixed-length sequences, the same type of feed-forward model used for fixed-size input can be used. For example, (Kamper et al., 2016; Bengio and Heigold, 2014) learn to embed spoken utterances of a fixed length in a Euclidean space where utterances of the same word are close together.

For variable-length sequences, it is common to use a model with some form of temporal awareness. A natural example are recurrent networks (see section 2.1.3.3) whose hidden state can be treated as a learned embedding of the input sequence up to a given point in time. Notably, in (Sutskever et al., 2014), a recurrent network was trained to map a sequence of words in a source language to a fixed-length vector, which was then used to generate a sequence of words in a target language, resulting in a quality translation of the original sentence. The resulting source-sentence embeddings encoded high-level linguistic characteristics, such as invariance to word order in sentences with the same meaning. In the audio domain, (Hershey et al., 2016) propose training a recurrent network to embed time-frequency "pixels" in spectrograms of acoustic mixtures so that sources can be separated using clustering techniques.

6.1.2 Attention

While recurrent networks have proven to be an effective method for embedding variable-length sequences, they have difficulty in modeling long-term dependencies

when trained with backpropagation through time (Bengio et al., 1994; Hochreiter and Schmidhuber, 1997; Pascanu et al., 2013) (see section 2.1.3.3 for a discussion). In the embedding setting, this may result in the end of the sequence having a greater impact on the embedding than the beginning. A recent technique for mitigating this issue has been dubbed "attention" (Bahdanau et al., 2014). Conceptually, an attention-based model includes a mechanism which learns to assign a weight to each sequence step based on the current and previous states. When used with recurrent networks, the addition of attention has proven effective in a wide variety of domains, including machine translation (Bahdanau et al., 2014), image captioning (Xu et al., 2015), and speech recognition (Chan et al., 2015; Bahdanau et al., 2015); see (Cho et al., 2015) for an overview. Attention can also be seen as analogous to the "soft addressing" mechanisms of the recently proposed Neural Turing Machine (Graves et al., 2014) and End-To-End Memory Network (Sukhbaatar et al., 2015) models.

More specifically, following the definition from (Bahdanau et al., 2014), given a length-T sequence h, attention-based models compute a series of "context" vectors c_t as the weighted mean of the sequence h by

$$c_t = \sum_{j=1}^{T} \omega_t[j] h[j] \tag{6.1}$$

where $\omega_t \in \mathbb{R}^T$ is a vector of weights computed at each time step t, with an entry for each sequence step in h. These context vectors are then used to compute a new sequence s, where s_t depends on s_{t-1}, c_t and the model's output at $t-1$. The weightings $\omega_t[j]$ are computed by

$$e_t[j] = a(s_{t-1}, h[j]) \tag{6.2}$$

$$\omega_t[j] = \frac{\exp(e_t[j])}{\sum_{k=1}^{T} \exp(e_t[k])} \tag{6.3}$$

where a is a learned function which can be thought of as computing a scalar importance value for $h[j]$ given the value of $h[j]$ and the previous state s_{t-1}. This formulation allows the new sequence s to have more direct access to the entire sequence h.

6.1.3 Feed-Forward Attention

While previous work on attention-based models has focused on recurrent networks, in the present work we will use feed-forward networks for the following reasons: First, even with attention, recurrent networks have difficulty modeling very long-term dependencies. The constant-Q spectrogram representation we have been using can result in sequences with thousands of timesteps, which is prohibitively long even for sophisticated models such as long short-term memory networks (Hochreiter and Schmidhuber, 1997). Second, feed-forward networks are much more efficient to train and evaluate than recurrent networks because their operations can be completely parallelized, whereas recurrent networks must evaluate each time step sequentially. Finally, attention provides its own form of temporal modeling because it integrates over the entire sequence.

We therefore propose a "feed-forward" variant of attention, which computes a weighting for each sequence step independent of all other steps as follows:

$$e[t] = a(h[t]) \tag{6.4}$$

$$\omega[t] = \frac{\exp(e[t])}{\sum_{k=1}^{T} \exp(e[k])} \tag{6.5}$$

$$c = \sum_{t=1}^{T} \omega[t] h[t] \tag{6.6}$$

Conceptually, this attention mechanism can be thought of as using the learned nonlinear function a to compute scalar "importance" values for each sequence step

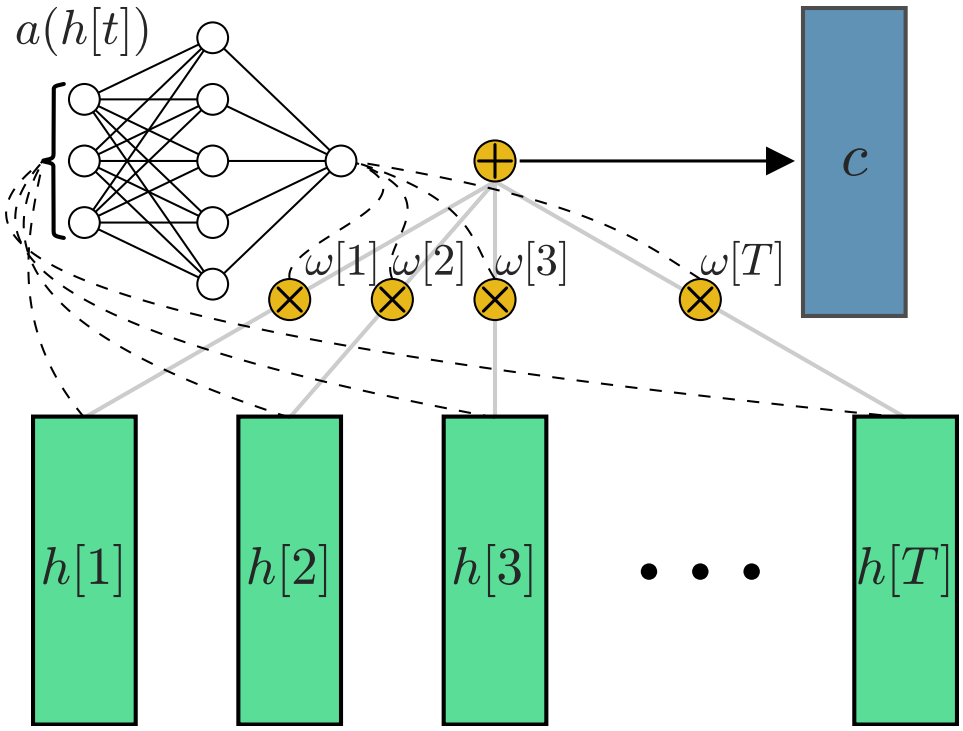

Figure 6.1: Diagram visualizing our feed-forward attention mechanism. Given a length-T input sequence h, the learned nonlinear function a computes a weighting $\omega[t]$ for each sequence element $h[t]$. The resulting probability vector $\omega \in \mathbb{R}^T$ is used to compute a weighted average c of h.

in h, which are then normalized using the softmax function and used to compute a weighted mean c over all sequence elements in h. Clearly, this mechanism will produce a single fixed-length vector regardless of the length of the sequence, which is what is necessary in sequence embedding. A schematic of our feed-forward attention mechanism is shown in figure 6.1.

The main difference between this formulation and the one proposed in (Bahdanau et al., 2014; Cho et al., 2015) (stated in the previous section) is that here we are only producing a single weighting ω and context vector c rather than a different ω_t and context vector c_t at each time step. This is primarily because our goal in

embedding is to produce a single vector for the entire sequence whereas previous applications of attention have mainly focused on sequence-to-sequence transduction. As a result, equation (6.6) does not contain any terms for a previous attention vector or a previous hidden state, because in the present contexts these quantities do not exist. The main disadvantage to using attention in this way is that it ignores temporal order by computing an average over the entire sequence. However, computing an order-agnostic integration of sequence has in fact been shown to be highly effective in the domain of music information retrieval. A common technique is to summarize a sequence of feature vectors by the mean, variance, and occasionally covariance of feature dimensions, which has been used for genre classification (Tzanetakis and Cook, 2002), instrument classification (Deng et al., 2008), artist recognition (Mandel and Ellis, 2005), and similarity rating prediction (Foster et al., 2014).

6.2 Experiment: MIDI to MSD Matching

Recall that the goal of this chapter is to prune large-scale sequence similarity searches by embedding sequences in a fixed-dimensional space where Euclidean distance approximates sequence similarity. Towards this end, we will utilize embeddings as follows: First, given a training set of matching sequences, we will train a feed-forward network with attention to embed sequences in a Euclidean space where matching pairs have a small distance and non-matching pairs have a large distance. Then, we will utilize this network to pre-compute embeddings for all sequences in a database. In order to match a query sequence, we will compute the distance between its embedding and all pre-computed embeddings from the database. Before proceeding, then, we need to specify the structure of our embedding model, our loss function, and our evaluation technique. In fact, this problem setting is remarkably similar to the previous chapter, and we will evaluate it on the same task and data. The primary

difference is that rather than embedding groups of aligned feature vectors in a shared Hamming space, we are embedding entire unaligned sequences as fixed-length vectors.

For our loss function, we can actually re-use equation (5.2) exactly, which we re-print here for convenience:

$$\mathcal{L} = \frac{1}{|\mathcal{P}|} \sum_{(x,y)\in\mathcal{P}} \|f(x) - g(y)\|_2^2 + \frac{\alpha}{|\mathcal{N}|} \sum_{(a,b)\in\mathcal{N}} \max(0, m - \|f(a) - g(b)\|_2)^2 \quad (6.7)$$

where $\mathcal{P}$ and $\mathcal{N}$ are collections of matching and non-matching sequences respectively, f and g are learned functions, and m and α are hyperparameters controlling the importance of non-matching pairs having a large embedded distance. In the present context, the learned embedding functions f and g will produce a single vector, and their input will be entire unaligned sequences rather than short groups of aligned feature vectors. In order to produce a single vector from an entire sequence, we will utilize the feed-forward attention mechanism described in the previous section. Specifically, we will utilize two different embedding networks for f and g, each of which will include a feed-forward attention mechanism. For the attention mechanisms' learned adaptive weighting function a, we use the standard architecture of a small neural network with one hidden layer:

$$a(x) = v^\top \tanh(Wx + b) \quad (6.8)$$

where $x \in \mathbb{R}^D$ is a single feature vector from the input sequence and $W \in \mathbb{R}^{D\times D}$, $b \in \mathbb{R}^D$, and $v \in \mathbb{R}^D$ are parameters.

6.2.1 Experiment Specifics

As training data, we use the same collection assembled in section 5.2.1. However, we will solely use the matches as ground truth, *not* the alignments, because our

system needs to be able to produce similarity-preserving embeddings regardless of whether the input sequences are aligned. We use the same feature representation, i.e. log-magnitude constant-Q spectrograms consisting of spectra with 12 bins per octave from MIDI note C2 (65.4 Hz) to B6 (987.8 Hz) computed every 46.4 milliseconds. We also use the same exact training, validation, development, and test splits, and utilize the statistics from the training split to standardize feature vectors.

For our networks, we fed the sequences first into a convolutional layer with 32 3×3 filters followed by a 2×2 max-pooling layer, followed by the attention mechanism of equation (6.6) with the weighting function of equation (6.8), followed by two fully-connected layers with 2048 units each, followed by a 128-dimensional output layer. We utilized rectified linear nonlinearities throughout, except on the final output layer and the weighting function which both used a hyperbolic tangent nonlinearity. All weights were initialized using He et. al.'s method (He et al., 2015) and all biases were initialized to zero. The networks were optimized with respect to equation (6.7) using RMSProp with mini-batches of 20 similar and dissimilar pairs. As in chapter 5, we assembled $\mathcal{N}$ by randomly choosing non-matching pairs which were re-sampled every time we had iterated over the training set. Minibatches were constructed by randomly choosing matching and non-matching pairs of sequences and cropping them to the minimum length within the minibatch. Every 200 minibatches, we computed the mean loss on the held-out validation set; when it was less than .99 times the previous lowest validation loss, we trained for 1000 more minibatches. We implemented our model using `lasagne` (Dieleman et al., 2015), which is built on `theano` (Bergstra et al., 2010; Bastien et al., 2012; Al-Rfou et al., 2016). A visualization of our model structure is shown in figure 6.2.

To choose hyperparameter settings, we once again used the Bayesian optimization framework proposed in (Snoek et al., 2012). We optimized the learning rate and

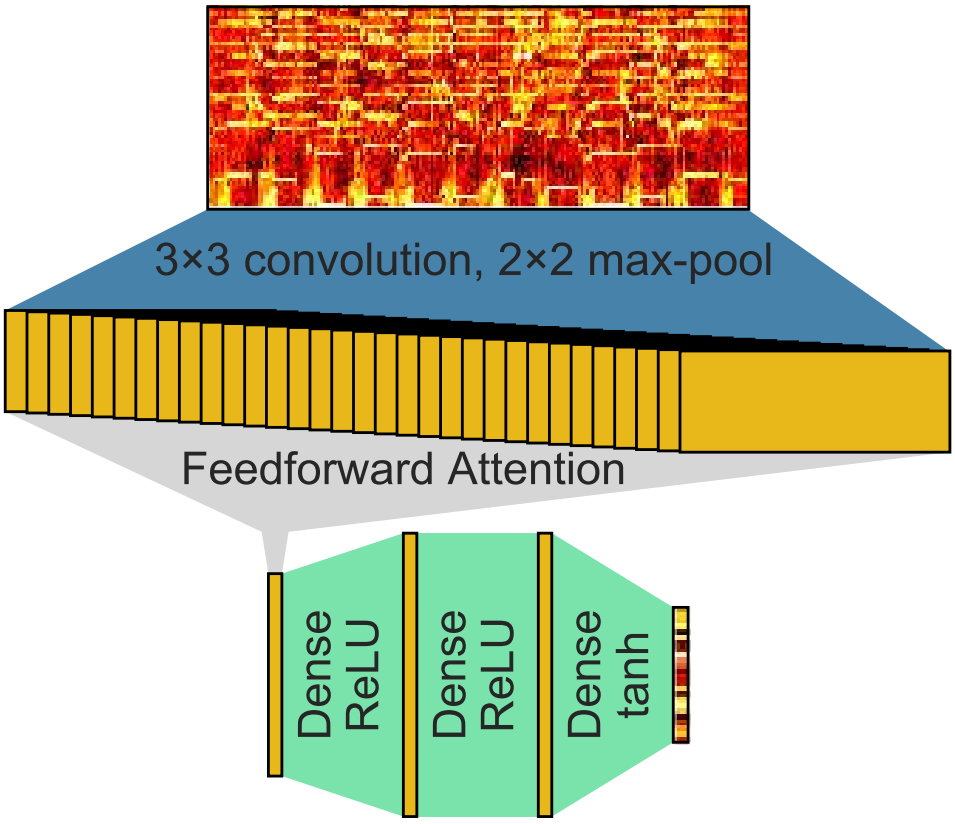

Figure 6.2: Visualization of the structure of our feed-forward network with attention. The network first processed the input with a convolutional and pooling layer, followed by our proposed feed-forward attention mechanism which integrates the entire sequence to a single fixed-length vector, followed by a series of fully-connected layers which produce the embedding.

RMSProp decay parameter, and the loss parameters α and m. As an objective, we computed the Bhattacharyya coefficient (Bhattacharyya, 1943) between the distributions of embedded distances between matching and non-matching pairs in the validation set. The best-performing network configuration achieved a Bhattacharyya coefficient of 0.393 and used a learning rate of $5 \cdot 10^{-6}$, an RMSProp decay of 0.999, $\alpha = 3.342$ and $m = 1.055$.

Before arriving at this system design, we also tried:

- Using more or fewer convolutional/pooling layers at the input of the network, including using no convolutional and pooling layers at all, a 5×12 convolutional layer followed by a variable number of 2×2 max-pooling layers and 3×3 convolutional layers, or using the same front-end as in chapter 5.
- Using 2×1 pooling layers as in chapter 5 and (Humphrey and Bello, 2012); we found 2×2 max-pooling worked better for embedding networks.
- Using up to four "parallel" attention mechanisms, whose output was concatenated before being processed by the fully-connected layer; this produced no performance gains.
- The output L^2 penalty from chapter 5 as well as the hash entropy-encouraging loss term proposed in (Yang et al., 2015, equation (3)), but neither were beneficial.
- Using dropout in the fully-connected layers, which hurt performance.
- Using more or fewer fully-connected layers of different sizes; two 2048-unit layers produced the best results.

- For optimization, Nesterov's accelerated gradient (Nesterov, 1983) and Adam (Kingma and Ba, 2015) (see section 2.1.4.3 and section 2.1.4.6) but found that RMSProp was most effective.

6.2.2 Baseline Method

As in chapter 5, we also evaluated a simple learning-free baseline method to make sure that our approach was beneficial. For our baseline, we followed the tradition of (Tzanetakis and Cook, 2002; Deng et al., 2008; Mandel and Ellis, 2005; Foster et al., 2014) and others by summarizing spectrograms by their mean and standard deviation across feature dimensions and concatenating the resulting statistics into a fixed-length embedding. Specifically, given a sequence $X \in \mathbb{R}^{M \times D}$ which consists of M D-dimensional feature vectors, we compute

$$\mu_X[d] = \frac{1}{M} \sum_{m=1}^{M} X[m, d] \tag{6.9}$$

$$\sigma_X[d] = \sqrt{\frac{1}{M} \sum_{m=1}^{M} (X[m, d] - \mu_X[d])^2} \tag{6.10}$$

$$e = \begin{bmatrix} \mu_X \\ \sigma_X \end{bmatrix} \tag{6.11}$$

where $e \in \mathbb{R}^{2D}$ is the resulting embedding. We compute this statistics-based embedding over the same log-magnitude, L^2 normalized spectrograms used throughout this thesis. This resulted in a 96-entry embedded vector for each spectrogram.

6.2.3 Results

Some qualitative intuition about our system's behavior can be gained by inspecting the embeddings resulting from matching and non-matching pairs of sequences.

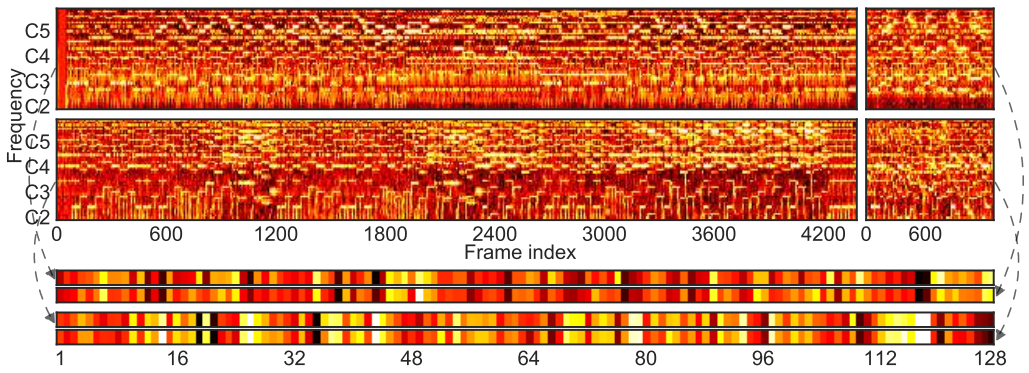

Figure 6.3: Example embeddings for two pairs of matching sequences produced by our feed-forward attention model after training. The first two rows display pairs of matching constant-Q spectrograms, with synthesized MIDI on the left and MSD entry on the right. For both pairs, the right sequence matches a subsequence of the left. The last two rows show their resulting embeddings. Dashed arrows denote which embedding corresponds to which sequence.

Embeddings for two different matching sequence pairs from the validation set can be seen in figure 6.3. Among the four embeddings displayed, the pairs of matched embeddings resulted in squared Euclidean distances of 0.0189 and 0.00738, while non-matching pairs had distances of 0.300 and 0.228. While these distances indicate the appropriate behavior, i.e. matching pairs result in a much smaller embedded distance, a more complete picture can be obtain by looking at the distributions of distances between matching and non-matching pairs in our validation set. We visualize these distributions for our proposed feed-forward attention system and for the baseline feature statistics-based method in figure 6.4. From this figure, we can see that our proposed approach produces embeddings which better differentiate between matching and non-matching pairs.

Because our proposed feed-forward attention system is a novel architecture, we are interested in getting some insight into its learned behavior. One way to accomplish this is to look at the attention weighting values ω over the course of an

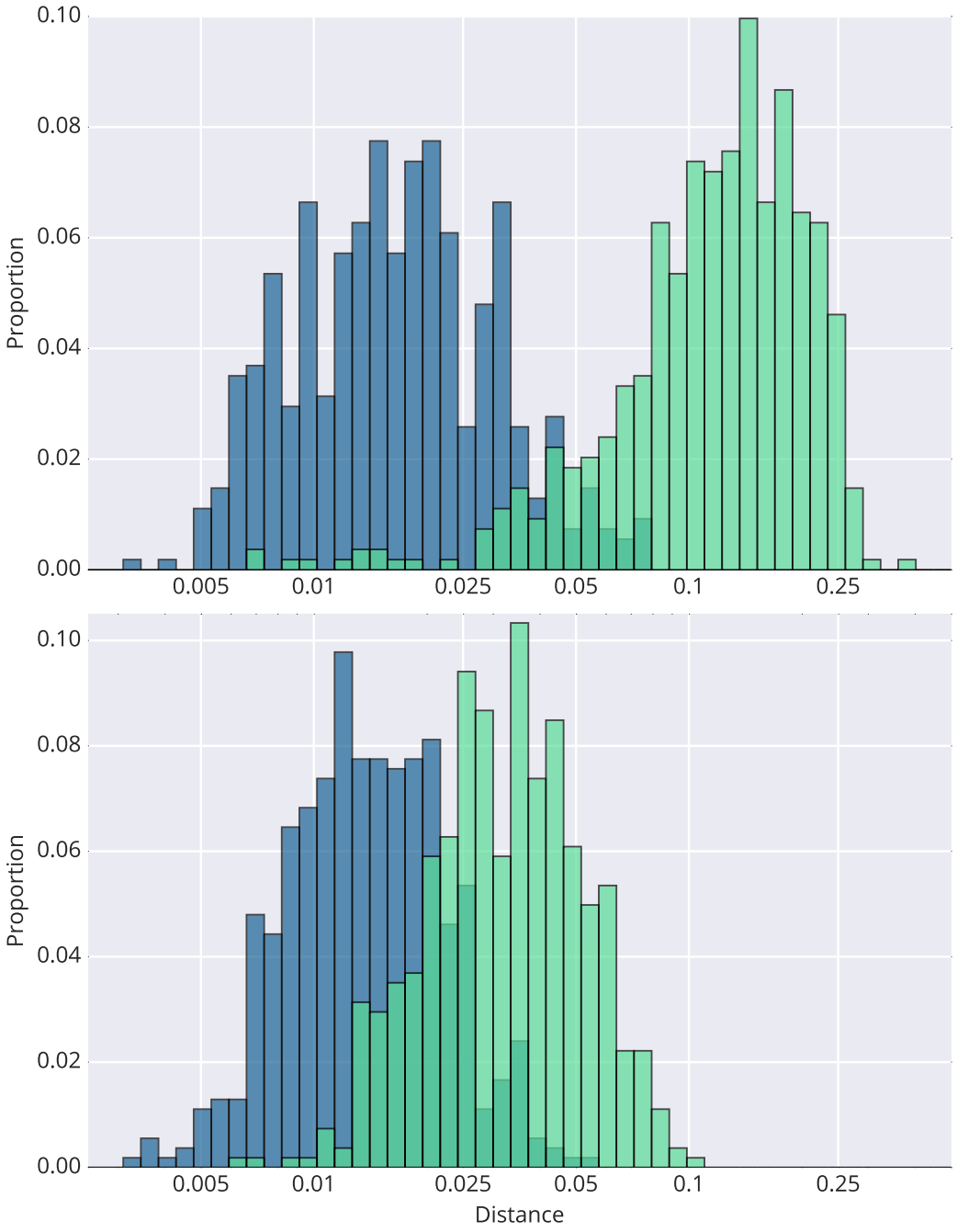

Figure 6.4: Distributions of embedded distances for matching (blue) and non-matching (green) sequences from our validation set. At the top are the distances produced by our proposed feed-forward attention model after training; at the bottom are distances based on the baseline proposed in section 6.2.2. Note that the x-axes are log-scaled.

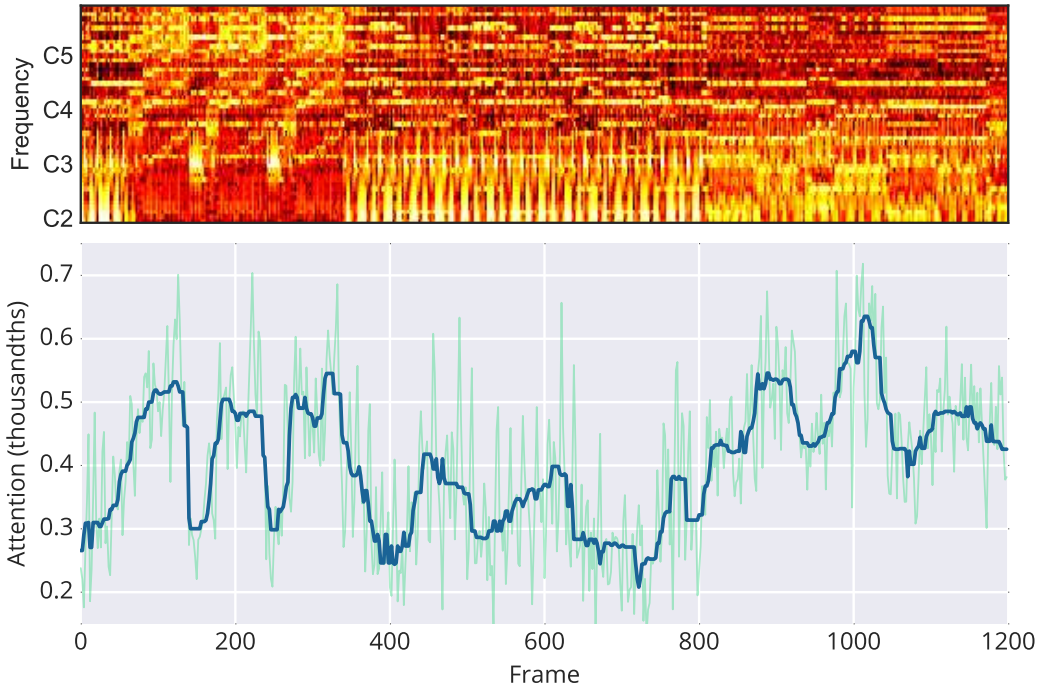

Figure 6.5: Example attention weighting values ω (bottom) produced by our MIDI embedding network for a synthesized MIDI spectrogram (top) from our validation set. The raw attention values are shown in light green; because they are noisy, we also display a median filtered version (with a window size of 15 frames) in blue.

input sequence. We visualize this for our MIDI spectrogram embedding network in figure 6.5. One clear pattern from this visualization is that the attention mechanism tends to downweight the portion of the song with strong low-frequency transients (frames 350 to 800). In addition, in one region of mostly high attention (frames 100 to 350) there are two regions where the attention drops; these regions also have a sudden burst of energy in a low-frequency bin.

Another generally applicable visualization technique is DeepDream (Mordvintsev et al., 2015), which refers to the technique of optimizing the *input* of a pre-trained model, rather than the parameters, in order to minimize an objective. We utilized this technique to see what kind of inputs the attention mechanism in each of our

networks weights very highly. To achieve this, we optimized a randomly-initialized input to minimize the entropy of the attention weights ω. This effectively tells the network to generate an input for which a small region is given the maximum attention and the rest of the input is ignored. As advocated in (Mordvintsev et al., 2015), we initialized with zero-mean, unit-variance Gaussian noise and utilized standard gradient descent with gradients normalized to have unit norm. In addition, we enforced the constraint that the input feature vectors stay L^2-normalized. We plot 40 frames of the resulting input surrounding the maximal-attention region for both of our networks in figure 6.6. Most apparently, there is a clear lack of high-frequency content in the high-attention regions. In addition, the regions of low attention appear to be basically noise. This suggests that our attention mechanism may be most interested in steady-state regions with little high-frequency content.

For empirical evaluation, we follow the procedure of section 5.2.5 exactly, except that rather than comparing sequences using downsampled hash sequences, we will be embedding them and computing the distance between their embedded representations. More specifically, we first computed embeddings of constant-Q spectrograms for all of the short (30 to 60 second) preview recordings of each entry in the MSD provided by 7digital (Schindler et al., 2012). Then, we computed a constant-Q spectrogram and its embedding for each MIDI file in the test set. To determine performance, we computed the Euclidean distance between the embedding of each MIDI file and the embeddings of every entry in the MSD and determined how many entries had a distance that was smaller than the correct match. We carried out this procedure both for our proposed attention-based embedding technique (referred to as "PSE", for **p**airwise **s**equence **e**mbedding) and for the feature statistics-based baseline approach described in section 6.2.2 (referred to as "statistics").

We plot the resulting proportion of MIDI files in the test set whose correct

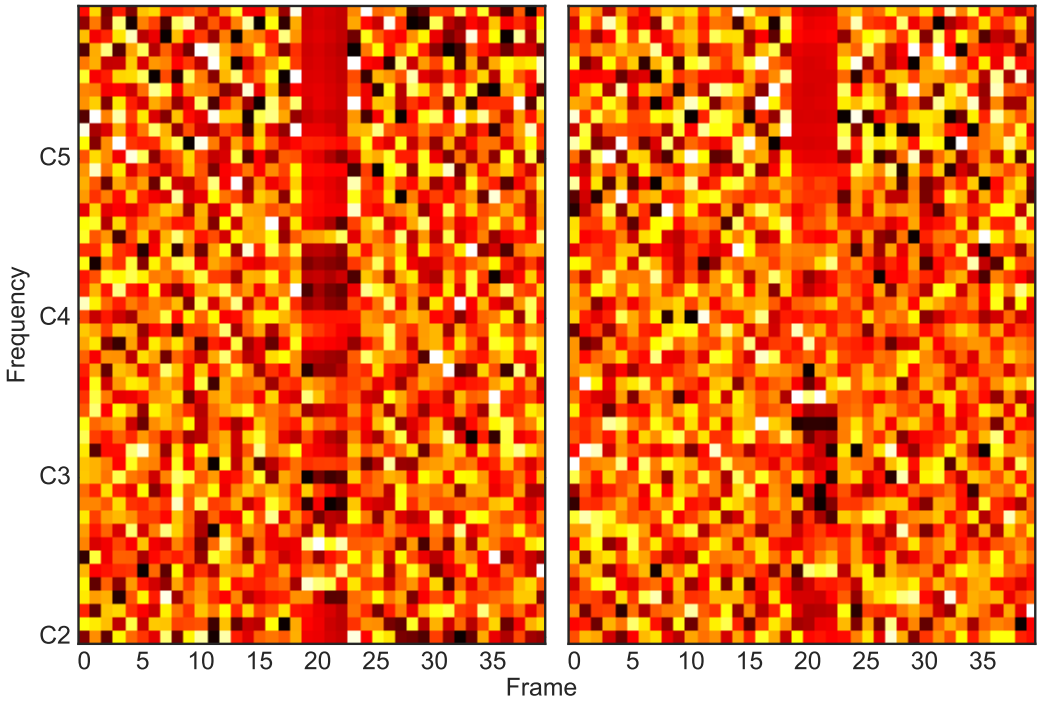

Figure 6.6: Inputs generated by minimizing the entropy of the resulting attention weights ω produced by our trained MIDI embedding (left) and audio embedding (right) networks. Shown are 40 frames of each generated input, centered around the region of maximal attention.

match ranked under a certain threshold in figure 6.7. To borrow the evaluation from chapter 5, we also computed the threshold N at which 95% of the correct matches in the test set had a rank better than N. For PSE, N can be set to 70,445, while the baseline would require 242,573. This means that we can used our proposed embedding method to prune over 90% of the database with high confidence, whereas the simple learning-free baseline would only allow us to discard about 75%.

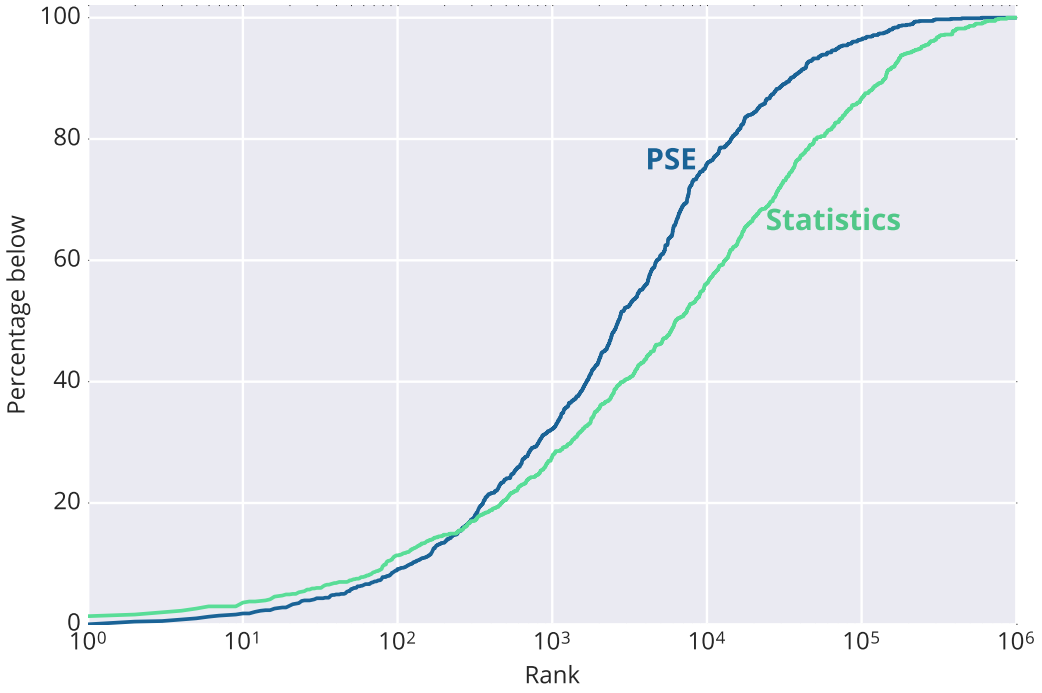

Figure 6.7: Percentage MIDI files in the test set whose correct match in the MSD fell below a given rank, using the attention-based PSE method and our baseline statistics method described in section 6.2.2.

6.3 Discussion

Comparing to figure 5.6, our embedding approach is clearly much less precise than any of the hash sequence-based methods. This is to be expected due to the fact that we are representing sequences of potentially thousands of 48-dimensional vectors as a single 128-dimensional vector. Fortunately, it is also much faster: Computing the million embedding distance calculations required to compare a MIDI file to the entire MSD only takes about 400 milliseconds (i.e. about 400 nanoseconds per MIDI-to-MSD entry comparison) on our Intel Core i7-4930k CPU. This means that we can discard over 90% of the MSD with high confidence with an operation which is over 1,000 times faster than the downsampled hash sequence-based methods of

chapter 5 and is over 600,000 times faster than a brute-force approach using the DTW system from chapter 4.

To improve performance of our proposed method we are interested in investigating the same ideas discussed in section 5.3, i.e. different loss functions and methods of sampling negative examples. In addition, as more and more methods are developed for making recurrent networks able to handle very long-term dependencies (some of which are mentioned in section 2.1.3.3), we are interested in testing whether recurrence could become applicable to our problem.

In summary, we have proposed a novel feed-forward attention mechanism which is capable of integrating arbitrary-length sequences to individual vectors in a fixed-dimensional embedded space where sequence similarity is approximated as Euclidean distance. We showed that using this model for pruning large-scale similarity search produces a system which can extremely efficiently rule out a large proportion of the database. Utilizing this technique alongside the systems proposed in chapter 5 and chapter 4 provides a "cascade" of techniques where we are able to discard more and more of the database with more and more expensive operations until we obtain a high-confidence match. In the following chapter, we put this approach to use on the task of identifying matches in the MSD for our entire 178,561 MIDI file collection.

Chapter 7

Assembling a Collection of MIDI Files Matched to the Million Song Dataset

Combining the results of chapters 4 to 6, we now have all the ingredients required to efficiently match MIDI files to the Million Song Dataset. Taken together, these three techniques present a cascade of matching methods, each of which achieves more precision at higher computational cost than the last. This allows us to quickly discard the vast majority of the dataset before utilizing the expensive, but accurate, DTW operation of chapter 4. While our main application in this thesis is MIDI-to-audio matching, all of these techniques are learning-based and data-adaptive, so we are optimistic they will be effective in other domains.

In this chapter, we finally utilize these techniques to match our collection of 178,561 MIDI files to the MSD. In the following section, we discuss the practical implementation of this process, perform a small experiment to verify its performance, and give an overview of the results. Then, in section 7.2, we quantify the reliability

of annotations for content-based MIR extracted from this data by comparing them to human-annotated ground truth.

7.1 Matching MIDI Files

As an overview, our method for matching MIDI files to the MSD will proceed as follows: Given a query MIDI file, we first synthesize it using `pretty_midi`'s `fluidsynth` method. Then, we compute the log-magnitude, L^2-normalized, constant-Q spectrogram representation utilized throughout this thesis. Using the pre-trained DHS and PSE models from chapters 4 and 6, we compute the MIDI file's downsampled hash sequence and embedding. We then compute the pairwise distance between the resulting embedding and all pre-computed embeddings of entries in the MSD, rank the entries according to their distance, and discard all but the top N_{PSE}. We perform a similar pruning with the downsampled hash sequences, computing the DTW distance between the query and the MSD entries which remain after the embedding-based pruning step, ranking by hash sequence DTW distance, and discarding all but the top N_{DHS}. Finally, we align the MIDI file to the remaining N_{DHS} entries utilizing the "gold-standard" system of section 4.4.1.

Before proceeding, we must first decide the pruning thresholds N_{PSE} and N_{DHS}. Choosing these thresholds directly trades off precision and computation, because setting larger thresholds will decrease the likelihood of discarding a valid match but will result in more comparisons being made downstream. Based on the results of section 5.2.6.2 and section 6.2.3, we chose $N_{PSE} = 100,000$ and $N_{DHS} = 250$. This corresponds to pruning 90% of the MSD in the first step, then pruning 99.75% of the remaining entries, so that in the end the expensive DTW comparison is only performed on 0.025% of the MSD. According to our test-set results, this results in a false-reject rate of approximately 3.5% for the embedded distance pruning and 3.2%

for the downsampled hash sequences. If false rejects occur in each step independently, we can expect a combined false reject rate of up to 6.6%.

7.1.1 Shortcuts

Overall, we will follow the matching processes of sections 4.4.1, 5.2.2 and 6.2.1, except for the following two changes made for efficiency:

First, we utilize an exact pruning method when comparing downsampled hash sequences. Assuming we are computing the DTW distance as defined in section 2.2.3 between two sequences, one of length M and one of length N with $M \leq N$, then the normalized DTW distance will never be smaller than T if there are not at least M pairwise distances less than T. Given this, while computing the pairwise distance matrix for downsampled hash sequences, we can (with a minor performance penalty) keep track of the number of entries which are smaller than T, where T is set to the smallest normalized DTW distance which has been encountered so far. If fewer than M entries are less than T, we can skip computing the DTW alignment path because we can guarantee that the resulting normalized DTW distance will not be smaller than T. In practice, this avoids the expensive operation of computing the DTW distance for many candidates whose distance matrices dictate that they cannot be the correct match.

Second, instead of utilizing the median pairwise distance for the CQT-to-CQT DTW additive penalty as suggested in section 4.4.1, we will use the mean distance because in practice it is cheaper to compute than the median. We justify this decision based on the discussion in section 4.3 which suggests that small changes to the additive penalty will not affect alignment quality. In practice, we found the mean and median pairwise distance to be extremely close, so this change likely has a negligible effect on the results aside from a small speed-up.

7.1.2 Measuring Performance

Prior to performing the matching of our full collection of MIDI files to the MSD, it is useful to get an idea of the expected accuracy of the process described above. We therefore utilized this approach on the same evaluation task used in chapters 5 and 6, i.e. measuring the accuracy of matching our test set of known-correct MIDI-MSD pairs. Specifically, we utilized the DHS and PSE methods of chapters 5 and 6 to prune all but 100,000 and 250 entries respectively and then computed the match confidence score for the remaining 250 entries using the gold-standard alignment system of section 4.4.1. For evaluation, we ranked the remaining 250 entries according to their resulting confidence scores. If the correct match in the MSD was pruned incorrectly, we assigned a rank 1,000,000, because this effectively results in the correct matching being unrecoverable by our system. For full details of this evaluation task, refer to section 5.2.

The proportion of pairs for which the correct entry was ranked under a given threshold is shown in figure 7.1. For comparison, we also plot the curves obtained using the DHS and PSE approaches from chapters 5 and 6. Using the combined method, the correct match in the MSD had the highest confidence score 78.2% of the time, and for 94.6% of the test set the correct match ranked in the top 5. Beyond this point, the performance is effectively constant - this is because for 94.7% of the test set, the pruning steps discarded the correct match. This is within our expected upper limit for the false reject rate of 6.6%. We could in theory further verify the source of these false rejects by also evaluating the final slow-but-accurate DTW step alone, but running this evaluation over our entire test set would take about nine years of CPU time, so we skipped this experiment. That our high-precision approach only ranked the correct match first 78.2% of the time is also expected because the MSD contains many duplicates and alternate versions of songs (Bertin-Mahieux and

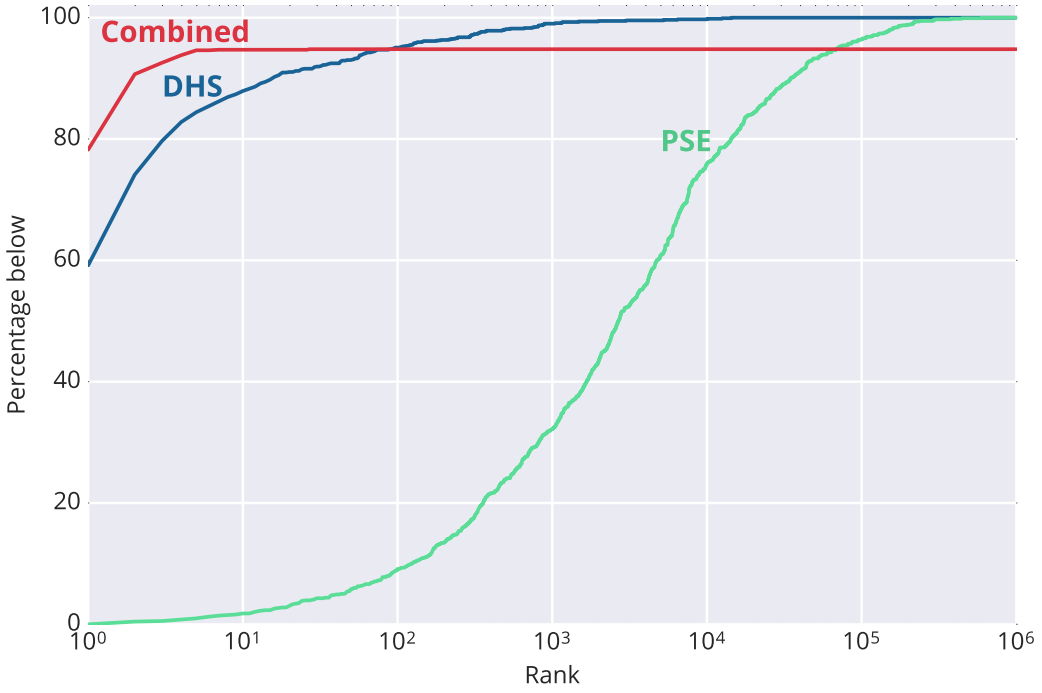

Figure 7.1: Percentage of MIDI files in the test set whose correct match in the MSD ranked better than a given threshold, using the DHS model of chapter 5, PSE model of chapter 6, and their combination with our best-performing DTW system described section 4.4.1.

Ellis, 2012). It's also important to note that these figures do not necessarily reflect real-world performance because, due to the way it was constructed, our test set consists only of ground truth matches which could successfully be recovered by our DTW-based system of section 4.4.1. Nevertheless, it does demonstrate the extent to which our combination of pruning approaches causes false rejects.

Of course, our approach's precision is only beneficial if it is also efficient. On average, matching a MIDI file to the MSD using this cascade of techniques takes about 109 seconds utilizing a single core of an Intel Core i7-4930k CPU. Of this, on average roughly 500 milliseconds is spent pruning all but 100,000 entries using the embedding approach, 46 seconds is spent pruning down to 250 entries using the

downsampled hash sequence approach, and 62 seconds is spent computing confidence scores for the remaining 250 entries. Most of the remaining 500 milliseconds is the time required to compute all of the necessary representations for the MIDI file prior to matching. Compared to the approximately 2.8 days it would take to match a single MIDI file to the MSD using the DTW approach from section 4.4.1 on its own, we consider the 5.4% false reject rate a small price to pay for this increase in efficiency. Of course, we could increase precision substantially by raising the pruning thresholds in exchange for a drop in efficiency, but we consider this to be a good trade-off.

7.1.3 The Lakh MIDI Dataset

Now that we have empirically validated our approach, we proceed to matching our collection of 178,561 MIDI files to the MSD. To do so, we simply followed the approach of the previous section exactly, except that we matched our entire collection against the MSD instead of just our test set. Based on the fact that matching a single file takes roughly 109 seconds on average, performing this matching for all 178,561 MIDI files would take about 225 days of compute time. To speed things up, we ran this process in parallel across 32 processors, resulting in a total run time of about 7 days. While performing the match, we discovered 5,602 MIDI files which were corrupt or otherwise unparsable by `pretty_midi`. In addition, in order to avoid spurious matches and incomplete transcriptions, we discarded all MIDI files which were less than 30s in length. In the end, we therefore searched for matches in the MSD for a total of 160,603 MIDI files.

An informative way to report the extent to which we successfully found matches in the MSD is to plot the number of MIDI files in our collection which had at least one MSD entry against which the DTW confidence score was below a threshold. We

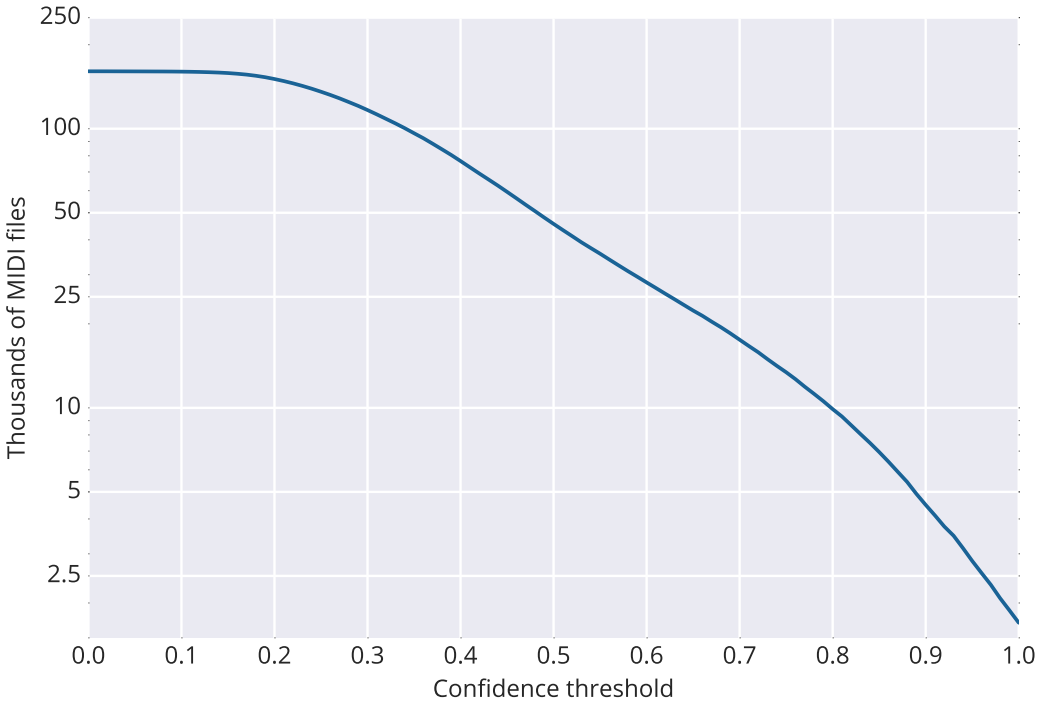

Figure 7.2: Number of MIDI files which had at least one entry in the MSD whose DTW confidence score was above a given threshold. For example, just under 50,000 MIDI files had at least one MSD entry whose match confidence score was above 0.5.

plot this curve in figure 7.2. Choosing 0.5 as the threshold above which a confidence score denotes a match (as we did in section 5.2.1 based on the results of section 4.5), we found 45,612 MIDI files which had at least one match in the MSD, and a total of 118,661 MIDI-to-MSD pairings. Many MIDI files were matched to multiple MSD entries due to duplicates, covers, and alternate versions; for example, one MIDI transcription of "Imagine" by John Lennon was matched to six original renditions, ten covers, one hip-hop song which sampled it ("Image" by Nas), and one song which used an identical chord progression ("Bringin' Down Dinner" by Neil Young).

Of course, the main purpose of creating this dataset is to facilitate MIR research by sharing it. Following the example of the Million Song Dataset, we name this

collection the "Lakh MIDI Dataset" (LMD), because depending on how you count we are providing about 100,000 MIDI files. Towards this end, we make it available online[1] in the following formats: First, the entire collection of 178,561 unique MIDI files, unmodified, along with all potential metadata available from their filenames (referred to as "LMD-full"). Second, a mapping between each of these MIDI files to MSD entries they were successfully matched to, along with the resulting DTW confidence score (referred to as "LMD-matched"). Third, for each successful MIDI-MSD pair, a pre-aligned MIDI file utilizing our "gold standard" system from section 4.4.1 (referred to as "LMD-aligned"). By distributing it in these varying levels of granularity, we also allow future researchers to improve upon our matching and alignment processes. In addition, we note that should a new collection of MIDI files become available, our process could easily be re-run and the resulting datasets expanded. Alternatively, we could also easily leverage other digital score formats for which many transcriptions are available such as MusicXML (Good, 2001) or GuitarPro.[2]

7.2 Measuring a Baseline of Reliability for MIDI-Derived Information

Based on the discussion in section 3.1, our dataset can potentially be used to produce ground truth annotations for audio content-based MIR. For example, beat annotations for a given 7digital preview recording could be obtained by utilizing the time signature and tempo change events in the corresponding aligned MIDI file. However, the validity of a given annotation will depend on both the transcription quality as well as the success of the alignment process. Ideally, our alignment confidence score will reflect whether or not a given annotation is to be trusted. To

[1] http://colinraffel.com/projects/lmd
[2] http://www.guitar-pro.com

investigate further, we now focus on measuring the reliability of ground truth data extracted from aligned MIDI transcriptions.

A straightforward way to evaluate the quality of MIDI-derived annotations is to compare them with hand-made annotations for the same songs. Given a MIDI transcription and human-generated ground truth data, we can extract corresponding information from the MIDI file and compare using the evaluation metrics employed in the Music Information Retrieval Evaluation eXchange (MIREX) (Downie, 2008). We therefore leveraged the Isophonics Beatles annotations (Mauch et al., 2009) as a source of ground truth to compare against MIDI-derived information. MIDI transcriptions of these songs are readily available due to The Beatles' popularity.

Our choice in tasks depends on the overlap in sources of information in the Isophonics annotations and MIDI files. Isophonics includes beat times, song-level key information, chord changes, and structural segmentation. As noted in Section 3.1, beat times and key changes may be included in MIDI files but there is no standard way to include chord change or structural segmentation information. We therefore performed experiments to evaluate the quality of key labels and beat times available in MIDI files. Fortuitously, these two experiments give us an insight into both song-level timing-agnostic information (key) and alignment-dependent timing-critical information (beats). To carry out these experiments, we first manually identified 545 MIDI files from LMD-full[3] which had filenames indicating that they were transcriptions of one of the 179 songs in the Isophonics Beatles collection; we found MIDI transcriptions for all but 11. The median number of MIDI transcriptions per song was 2; the song "Eleanor Rigby" had the most, with 14 unique transcriptions.

[3]The Million Song Dataset does not contain any Beatles songs, so we were unable to source these files from LMD-matched or LMD-aligned.

7.2.1 Key Experiment

In our first experiment, we evaluated the reliability of key change events in MIDI files. We followed the MIREX methodology for comparing keys (Ehmann et al., 2016), which proceeds as follows: Each song may only have a single key. All keys must be either major or minor, e.g. "C# Major" and "E minor" are allowed but "D Mixolydian" is not. An estimated key is given a score of 1.0 when it exactly matches a ground truth key, 0.5 when it is a perfect fifth above the ground truth key, 0.3 when it is a relative major or minor, 0.2 when it is a parallel major or minor, and 0.0 otherwise (Ehmann et al., 2016). We utilized the evaluation library `mir_eval` (Raffel et al., 2014) to compute this score.

The Isophonics annotations mostly follow this format, except that 21 songs contained multiple key annotations and 7 others contained non-major/minor keys. To simplify evaluation, we discarded these songs, leaving 151 ground truth key annotations. Of our 545 Beatles MIDIs, 221 had no key change event and 5 had more than one, which we also omitted from evaluation. This left 223 MIDI files for which we extracted key annotations and compared them to valid Isophonics annotations. Because of the preponderance of C major key change events noted in Section 3.1.4, we also evaluated MIDI-derived C Major and non-C major instances separately to see whether they were less reliable.

As a baseline, we also extracted keys using the QM Vamp Key Detector plugin (Cannam et al., 2015) whose underlying algorithm is based on (Noland and Sandler, 2007) which finds the key profile best correlated with the chromagram of a given song. This plugin achieved the highest score in MIREX 2013, and has been the only key detection algorithm submitted in 2014 and 2015. This gives us a reasonable expectation for a good audio content-based key estimator. To determine the extent to which human annotators agree on key labels, we also collected key annotations for

Source	Score	Comparisons
MIDI, all keys	0.400	223
MIDI, C major only	0.167	146
MIDI, non-C major	0.842	77
QM Key Detector	0.687	151
`whatkeyisitin.com`	0.857	145

Table 7.1: Mean scores achieved, and the number of comparisons made, by different datasets and algorithms compared to Isophonics Beatles key annotations.

Beatles' songs from `whatkeyisitin.com`. As with the Isophonics key annotations, some songs had multiple and/or modal key labels; we discarded these and ended up with 145 labels for songs in the Isophonics dataset.

The mean scores resulting from comparing each dataset to the Isophonics annotations can be seen in Table 7.1. At first glance, the mean score of 0.4 achieved by MIDI key change messages is discouraging. However, by omitting all MIDI files with C major key events (which achieved a mean score of 0.167), the mean score jumps to 0.842. This is comparable to the human baseline, and is substantially higher than the algorithmically estimated score. We therefore propose that *non-C major* MIDI key change events are as reliable as hand-annotated labels, but that C major key annotations in MIDI files are effectively useless.

7.2.2 Beat Experiment

Utilizing many of the sources of information in MIDI files depends on the precise alignment of a given MIDI file to an audio recording of a performance of the same song. We therefore performed an additional experiment to evaluate the quality of MIDI-derived beat annotations, which are evaluated on the scale of tens of milliseconds. Producing valid beat annotations from a MIDI file requires not only

that the file's meter information is correct, but also that it has been aligned with high precision. To align our Beatles MIDI files to corresponding audio recordings, we used our best-performing DTW system from section 4.4.1 exactly, except that we computed spectrograms with a hop size of 23.2 milliseconds rather than 46.4 milliseconds. This finer timescale is more appropriate for the beat evaluation metrics we will use, which have tolerances measured in tens of milliseconds.

We used `pretty_midi`'s `get_beats` method to extract beat times from our 545 Beatles MIDI files, and adjusted each beat's timing according to the MIDI file's alignment to corresponding audio recordings. For evaluation, we used the *F-measure*, *Any Metric Level Total*, and *Information Gain* metrics described in (Davies et al., 2009), as implemented in `mir_eval`. As a baseline, we also computed beat locations using the `DBNBeatTracker` from the `madmom` software library,[4] which is based on the algorithm from (Böck et al., 2014). This represents a state-of-the-art general-purpose beat tracker which, on the Beatles data, can reliably produce high-quality annotations. If MIDI-derived beat annotations are to be taken as ground truth, they must achieve scores similar to or higher than the `DBNBeatTracker`.

We visualize the resulting scores in Figure 7.3. Because we don't expect beats to be extracted accurately from MIDI files that are poor transcriptions or when alignment failed, we plotted each MIDI file as a single point whose x coordinate corresponds to the alignment confidence score and whose y coordinate is the resulting evaluation metric score achieved. Ideally, all points in these plots would be clustered in the bottom left (corresponding to failed alignments with low confidence scores) or top right (corresponding to a successful alignment and beat annotation extraction with a high confidence score). For reference, we plot the mean score achieved by the `DBNBeatTracker` as dotted lines for each metric. From these plots, we can see that

[4] `https://github.com/CPJKU/madmom`

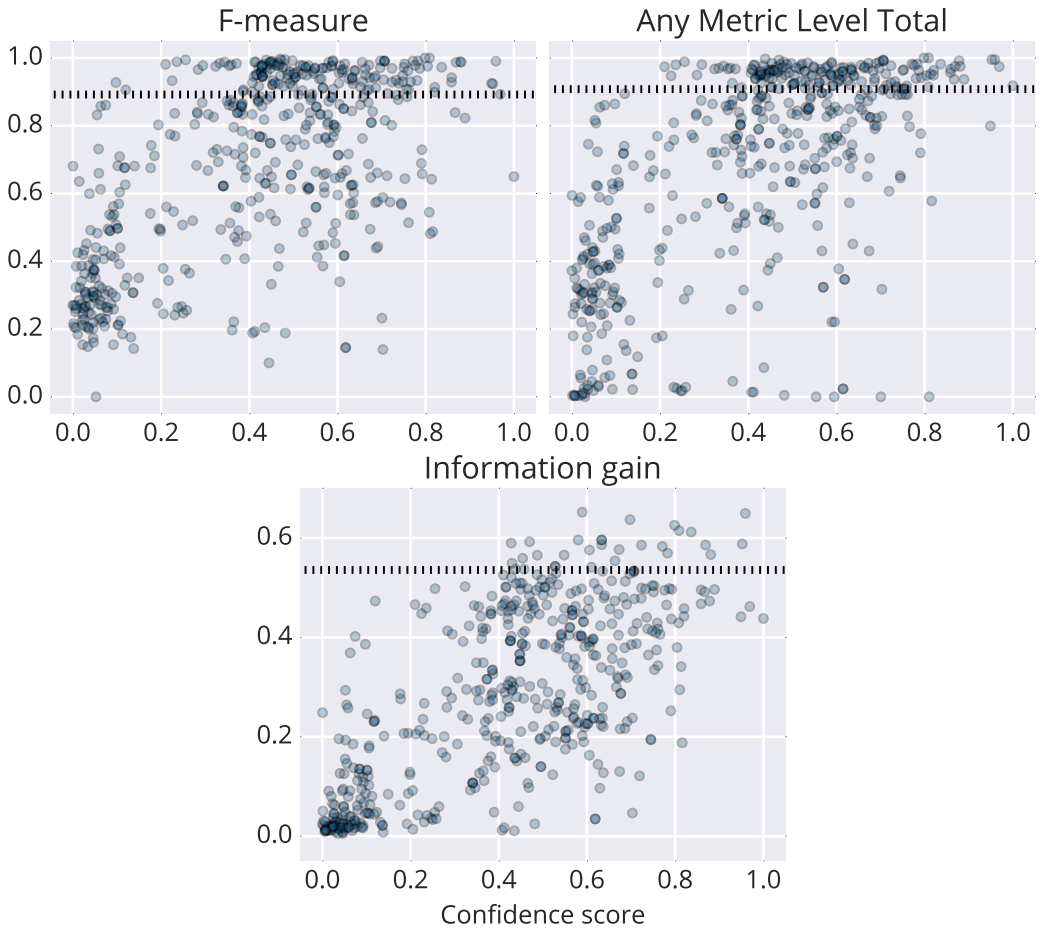

Figure 7.3: Beat evaluation metric scores (compared to Isophonics beat annotations) and alignment confidence scores achieved by different audio-to-MIDI alignments of Beatles MIDI files, with each shown as a blue dot. Mean scores for each metric achieved by the `DBNBeatTracker` (Böck et al., 2014) are shown as dashed lines.

in many cases, MIDI-derived annotations achieve near-perfect scores, particularly for the *F-Measure* and *Any Metric Level Total* metrics. However, there is no reliable correspondence between high confidence scores and high evaluation metric scores. For example, while it appears that a prerequisite for an accurate MIDI-derived beat annotation is a confidence score above .5, there are many MIDI files which had high confidence scores but low metric scores (appearing in the bottom-right corner of the plots in Figure 7.3).

We found that this undesirable behavior was primarily caused by a few issues: First, it is common that the alignment system would produce alignments which were slightly "sloppy", i.e. were off by one or two frames (corresponding to 23.2 milliseconds each) in places. This had less of an effect on the *F-measure* and *Any Metric Level Total* metrics, which are invariant to small temporal errors up to a certain threshold, but deflated the *Information Gain* scores because this metric rewards consistency even for fine-timing errors. Second, many MIDI files had tempos which were at a different metric level than the annotations (e.g. double, half, or a third of the tempo). This affected the *Any Metric Level Total* scores the least because it is invariant to these issues, except for the handful of files which were transcribed at a third of the tempo. Finally, we found that the confidence score produced by the alignment system is most reliable at producing a low score in the event of a total failure (indicated by points in the bottom left of the plots in Figure 7.3), but was otherwise insensitive to the more minor issues that can cause beat evaluation metrics to produce low scores.

7.2.3 Motivationg Future Work

These results suggest that the use of MIDI-derived annotations may come with a few caveats. We therefore believe that developing better methods to leverage all of the

information present in them is a tantalizing avenue for obtaining reliable ground truth data for music information retrieval. For example, while C major key annotations cannot be trusted, developing a highly reliable C major vs. non-C major classification algorithm for symbolic data (which would ostensibly be much more tractable than creating a perfect general-purpose audio content-based key estimation algorithm) would enable the reliable usage of all key change messages in MIDI files. Further work into robust audio-to-MIDI alignment is also warranted in order to leverage timing-critical information, as is the neglected problem of alignment confidence score reporting. Novel questions such as determining whether all instruments have been transcribed in a given MIDI file would also facilitate their use as ground truth transcriptions. Fortunately, all of these tasks are made easier by the fact that it is straightforward to extract musical information from MIDI files.

Chapter 8

Conclusion

To summarize, we have studied learning-based methods for large-scale sequence retrieval, with the specific application of matching a collection of MIDI files to the Million Song Dataset. We began by enumerating the sources of information available in MIDI files as they pertain to content-based music information retrieval in chapter 3. Then, in chapter 5, we optimized DTW-based audio-to-MIDI alignment system design over a novel synthetic task to produce an alignment scheme which was both accurate and able to report a reliable confidence score. This system was too slow for large-scale applications, so in chapter 5 we developed a method for learning a similarity-preserving mapping from sequences of continuously-valued feature vectors to downsampled sequences of binary vectors. For further speed-up, we also designed an attention mechanism which can learn to map entire sequences of feature vectors to single vectors in an embedded space where similarity is preserved, described in chapter 6. The combination of these methods allowed us to efficiently assemble the largest collection of MIDI files which have been paired and aligned to corresponding audio recordings, which we discuss in chapter 7.

Our specific contributions are as follows:

Chapter 2 We define and give background information for all of the tools and techniques used in this thesis, including neural networks, stochastic optimization, Bayesian optimization, time-frequency analysis, and dynamic time warping.

Section 3.1 We present a unified overview of the events in MIDI files which are relevant to music information retrieval and measure their availability in MIDI files found "in the wild".

Section 3.2 We itemize the steps necessary for leveraging MIDI files in music information retrieval tasks.

Section 4.1 We give a thorough review of system design choices and parameter settings used in DTW-based audio-to-MIDI alignment systems.

Section 4.2 We design a method for generating synthetic ground truth alignments for evaluating the accuracy of a given alignment system.

Section 4.3 We optimize DTW system design over synthetically-generated data to obtain a simple-to-implement system which is accurate and produces a score which reliably indicates whether an alignment has failed. We also verified that this system performs well on real-world data by manually auditioning alignments.

Section 5.1 We propose the novel approach of using a pair of convolutional networks to map groups of subsequent feature vectors to binary vectors. The implicit downsampling provides quadratic speedup to DTW and the use of binary vectors allows us to compute pairwise distances by a single exclusive-or followed by a POPCNT instruction.

Section 5.2 We show that, on the task of matching MIDI files to the Million Song Dataset, this approach can discard over 99.99% of the database with

95% confidence and is about 500 times faster than naive application of DTW. We also show that our model is applicable in settings where the raw data representation is not directly comparable using Euclidean distance.

Section 6.1 We use the paradigms of embedding and attention to motivate a novel feed-forward network structure which can learn to embed sequences in a fixed-dimensional space where similarity is approximated by Euclidean distance.

Section 6.2 On the same MIDI-to-MSD matching experiment, we show that our embedding technique can discard over 90% of the database with 95% confidence, and is about 600,000 times faster than naive DTW.

Section 7.1 We combine the above techniques to enable the full-scale matching of our collection of 178,561 MIDI files to the MSD, which yields about 50,000 MIDI files which have been successfully matched and aligned to entries in the MSD, the largest collection of its type. We make the collection available online.[1]

Section 7.2 We perform a quantitative evaluation of whether information extracted from MIDI files can be used as ground truth information for content-based music information retrieval by comparing MIDI-derived annotations to human-annotated ground truth.

Chapter B We make available all of the code used in this thesis.

8.1 Next Steps

The main product of this thesis is the Lakh MIDI Dataset, described in section 7.1.3. On its own, the collection of 178,561 unique MIDI files ("LMD-full") itself is most

[1] http://colinraffel.com/projects/lmd

applicable to corpus studies of music structure and patterns which are "metadata-agnostic". For example, (Mauch and Dixon, 2012) analyzed a smaller corpus of MIDI files to determine common drum patterns. Similar analyses could be performed on other aspects of the transcriptions, such as common chord progressions and melodic patterns. While LMD-full cannot be used on its own to obtain ground truth for audio content-based MIR tasks, MIDI files themselves provide a useful mechanism of obtaining synthetic data, as demonstrated in chapter 4 and utilized for onset detection in (Bello et al., 2005). In addition, there are ways in which non-matched, non-aligned transcriptions can be used to benefit content-based MIR tasks. For example, in automatic transcription systems it is common to smooth predictions with a "musical language model" which can predict the probability of a given candidate transcription (Poliner and Ellis, 2007; Sigtia et al., 2015). Such language models only require a large amount of transcriptions, not transcriptions matched to corresponding audio recordings. Of course, there is no guarantee that all of the MIDI files in LMD-full are actually full transcriptions because MIDI files are also sometimes used to store short snippets of music or even example riffs and drum beats for composition.

In contrast, we can say with high confidence that files in LMD-matched are transcriptions of songs. This greatly augments the information available for those MSD entries which have been matched to transcriptions. For example, in the past large-scale musicological studies have been undertaken using estimated pitch features (Serrà et al., 2012; Bertin-Mahieux et al., 2010). These results are potentially confounded by the fact that these features only represent a noisy and potentially error-prone approximation of the actual transcription. We expect that similar corpus studies utilizing the matched transcriptions will produce more reliable results. The transcriptions can also provide a stronger signal for query-by-humming retrieval

systems. On the other hand, by being matched to the MSD, the MIDI files themselves are augmented with a great deal of additional metadata, including the song's release year, lyrics, user listening preferences (Jansson et al., 2015), and human-annotated tags and genre labels (Schreiber, 2015).

Finally, we hope that LMD-aligned becomes a source for annotations for training audio content-based music information retrieval systems. However, as we discussed in section 7.2.3, some novel problems must be tackled before annotations derived from these aligned MIDI files can be fully leveraged. We hope that the release of the Lakh MIDI Dataset prompts further research into these problems, so that all of the information sources in MIDI files may be wholly utilized.

Beyond these potential uses for our new dataset, we also are interested in extending the techniques and applications we proposed in this thesis. We are most interested in fusing the different approaches we explored. For example, we believe a representation learning approach like the one used in chapter 5 could make DTW more effective in the final audio-to-MIDI alignment step. This is prompted by our observation in section 4.6 that the best representation for alignment is not necessarily the best for confidence reporting. The success of representation learning in chapter 5 in creating a more efficient and similarity-preserving representation for DTW suggests this approach may also be beneficial in the final high-precision alignment.

An issue with the training process utilized in chapter 5 is that it requires training data in the form of pre-aligned sequences. More broadly, this method of training does not optimize DTW performance directly, but instead optimizes the representation without considering the effect on the DTW alignment or distance. A more principled approach, then, would be to learn both the representation and the alignment procedure together as part of a larger system. Recently, a variety of approaches have been proposed to learn alignment as part of either the model's loss

function (e.g. the Connectionist Temporal Classification loss (Graves et al., 2006)) or model architecture (e.g. attention models (Bahdanau et al., 2014) or pointer networks (Vinyals et al., 2015)). We are interested in applying these methods to jointly learning a representation and a procedure for alignment.

In this thesis we developed a cascade of methods of varying precision and efficiency for large-scale sequence similarity searches. Each step was developed independently, which may be suboptimal. A better approach may be to simultaneously learn more and more efficient representations by repeatedly downsampling to coarser timescales, as is done in FastDTW (Salvador and Chan, 2007). However, rather than downsampling by simply computing the average of non-overlapping groups of subsequent feature elements as in (Salvador and Chan, 2007), we could utilize the same learning procedure of chapter 5 to produce efficient data-driven approximations at each timescale. We expect that this hierarchical approach would allow for even more efficient pruning.

Finally, we are interested in testing these techniques in domains outside of music. We suspect they will be beneficial in large-scale sequence comparison problems where sequences are oversampled, consist of high-dimensional feature vectors, and/or have elements which are not directly comparable by Euclidean distance. We hope the results and methods proposed in this thesis prompt novel research into learning-based methods for efficiently and effectively comparing sequences.

Bibliography

Ossama Abdel-Hamid, Abdel-rahman Mohamed, Hui Jiang, and Gerald Penn. Applying convolutional neural networks concepts to hybrid NN-HMM model for speech recognition. In *Proceedings of the 37th IEEE International Conference on Acoustics, Speech and Signal Processing*, pages 4277–4280, 2012. (Cited on page 34.)

Rami Al-Rfou, Guillaume Alain, Amjad Almahairi, Christof Angermueller, Dzmitry Bahdanau, Nicolas Ballas, Frédéric Bastien, Justin Bayer, Anatoly Belikov, Alexander Belopolsky, Yoshua Bengio, Arnaud Bergeron, James Bergstra, Valentin Bisson, Josh Bleecher Snyder, Nicolas Bouchard, Nicolas Boulanger-Lewandowski, Xavier Bouthillier, Alexandre de Brébisson, Olivier Breuleux, Pierre-Luc Carrier, Kyunghyun Cho, Jan Chorowski, Paul Christiano, Tim Cooijmans, Marc-Alexandre Côté, Myriam Côté, Aaron Courville, Yann N. Dauphin, Olivier Delalleau, Julien Demouth, Guillaume Desjardins, Sander Dieleman, Laurent Dinh, Mélanie Ducoffe, Vincent Dumoulin, Samira Ebrahimi Kahou, Dumitru Erhan, Ziye Fan, Orhan Firat, Mathieu Germain, Xavier Glorot, Ian Goodfellow, Matt Graham, Caglar Gulcehre, Philippe Hamel, Iban Harlouchet, Jean-Philippe Heng, Balázs Hidasi, Sina Honari, Arjun Jain, Sébastien Jean, Kai Jia, Mikhail Korobov, Vivek Kulkarni, Alex Lamb, Pascal Lamblin, Eric Larsen, César Laurent, Sean Lee, Simon Lefrancois, Simon Lemieux, Nicholas Léonard, Zhouhan Lin, Jesse A. Livezey, Cory Lorenz, Jeremiah Lowin, Qianli Ma, Pierre-Antoine Manzagol, Olivier Mastropietro, Robert T. McGibbon, Roland Memisevic, Bart van Merriënboer, Vincent Michalski, Mehdi Mirza, Alberto Orlandi, Christopher Pal, Razvan Pascanu, Mohammad Pezeshki, Colin Raffel, Daniel Renshaw, Matthew Rocklin, Adriana Romero, Markus Roth, Peter Sadowski, John Salvatier, François Savard, Jan Schlüter, John Schulman, Gabriel Schwartz, Iulian Vlad Serban, Dmitriy Serdyuk, Samira Shabanian, Étienne Simon, Sigurd Spieckermann, S. Ramana Subramanyam, Jakub Sygnowski, Jérémie Tanguay, Gijs van Tulder, Joseph Turian, Sebastian Urban, Pascal Vincent, Francesco Visin, Harm de Vries, David Warde-Farley, Dustin J. Webb, Matthew Willson, Kelvin Xu, Lijun Xue, Li Yao, Saizheng Zhang, and Ying Zhang. Theano: A python framework for fast computation of mathematical expressions. *arXiv preprint arXiv:1605.02688*, 2016. (Cited on pages 122, 145, and 200.)

Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*, 2014. (Cited on pages 140, 142, and 176.)

Dzmitry Bahdanau, Jan Chorowski, Dmitriy Serdyuk, Philemon Brakel, and Yoshua Bengio. End-to-end attention-based large vocabulary speech recognition. *arXiv preprint arXiv:1508.04395*, 2015. (Cited on page 140.)

Frédéric Bastien, Pascal Lamblin, Razvan Pascanu, James Bergstra, Ian J. Goodfellow, Arnaud Bergeron, Nicolas Bouchard, and Yoshua Bengio. Theano: new features and speed improvements. Deep Learning and Unsupervised Feature Learning NIPS Workshop, 2012. (Cited on pages 122, 145, and 200.)

Justin Bayer, Christian Osendorfer, Thomas Rückstiess, Sarah Diot-Girard, and Sebastian Urban. climin - a pythonic framework for gradient-based function optimization. Technical report, Technical University of Munich, 2015. (Cited on page 39.)

Juan Pablo Bello, Laurent Daudet, Samer Abdallah, Chris Duxbury, Mike Davies, and Mark B. Sandler. A tutorial on onset detection in music signals. *IEEE Transactions on Speech and Audio Processing*, 13(5):1035–1047, 2005. (Cited on pages 75 and 174.)

Samy Bengio and Georg Heigold. Word embeddings for speech recognition. In *Proceedings of Interspeech*, pages 1053–1057, 2014. (Cited on page 139.)

Yoshua Bengio. Learning deep architectures for AI. *Foundations and trends in Machine Learning*, 2(1):1–127, 2009. (Cited on page 28.)

Yoshua Bengio. Practical recommendations for gradient-based training of deep architectures. In *Neural Networks: Tricks of the Trade*, pages 437–478. Springer, 2012. (Cited on pages 28, 29, 30, and 40.)

Yoshua Bengio and Olivier Delalleau. On the expressive power of deep architectures. In *International Conference on Algorithmic Learning Theory*, pages 18–36, 2011. (Cited on page 28.)

Yoshua Bengio, Patrice Simard, and Paolo Frasconi. Learning long-term dependencies with gradient descent is difficult. *IEEE Transactions on Neural Networks*, 5(2): 157–166, 1994. (Cited on pages 33 and 140.)

Yoshua Bengio, Pascal Lamblin, Dan Popovici, and Hugo Larochelle. Greedy layer-wise training of deep networks. *Advances in Neural Information Processing Systems*, 19:153, 2007. (Cited on pages 28 and 31.)

Yoshua Bengio, Jérôme Louradour, Ronan Collobert, and Jason Weston. Curriculum learning. In *Proceedings of the 26th International Conference on Machine Learning*, pages 41–48, 2009. (Cited on page 134.)

Yoshua Bengio, Aaron Courville, and Pierre Vincent. Representation learning: A review and new perspectives. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 35(8):1798–1828, 2013. (Cited on page 111.)

Adam Berenzweig, Beth Logan, Daniel P. W. Ellis, and Brian Whitman. A large-scale evaluation of acoustic and subjective music-similarity measures. *Computer Music Journal*, 28(2):63–76, 2004. (Cited on page 118.)

James Bergstra and Yoshua Bengio. Random search for hyper-parameter optimization. *The Journal of Machine Learning Research*, 13(1):281–305, 2012. (Cited on page 46.)

James Bergstra, Olivier Breuleux, Frédéric Bastien, Pascal Lamblin, Razvan Pascanu, Guillaume Desjardins, Joseph Turian, David Warde-Farley, and Yoshua Bengio. Theano: a CPU and GPU math expression compiler. In *Proceedings of the 9th Python in Science Conference*, pages 1–7, 2010. (Cited on pages 122, 145, and 200.)

Donald J. Berndt and James Clifford. Using dynamic time warping to find patterns in time series. In *AAAI Workshop on Knowledge Discovery in Databases*, volume 10, pages 359–370, 1994. (Cited on pages 88 and 109.)

Thierry Bertin-Mahieux and Daniel P. W. Ellis. Large-scale cover song recognition using the 2D fourier transform magnitude. In *Proceedings of the 13th International Society for Music Information Retrieval Conference*, pages 241–246, 2012. (Cited on page 159.)

Thierry Bertin-Mahieux, Ron J. Weiss, and Daniel P. W. Ellis. Clustering beat-chroma patterns in a large music database. In *Proceedings of the 11th International Society for Music Information Retrieval Conference*, pages 111–116, 2010. (Cited on page 174.)

Thierry Bertin-Mahieux, Daniel P. W. Ellis, Brian Whitman, and Paul Lamere. The million song dataset. In *Proceedings of the 12th International Society for Music Information Retrieval Conference*, pages 591–596, 2011. (Cited on pages 85, 86, 102, 108, and 198.)

Anil Kumar Bhattacharyya. On a measure of divergence between two statistical populations defined by their probability distributions. *Bulletin of the Calcutta Mathematical Society*, 35:99–109, 1943. (Cited on pages 120 and 147.)

Jens Blauert. *Spatial hearing: the psychophysics of human sound localization*. MIT press, 1997. (Cited on page 90.)

Sebastian Böck, Florian Krebs, and Gerhard Widmer. A multi-model approach to beat tracking considering heterogeneous music styles. In *Proceedings of the 15th International Society for Music Information Retrieval Conference*, pages 603–608, 2014. (Cited on pages 167 and 168.)

Kurt Bollacker, Colin Evans, Praveen Paritosh, Tim Sturge, and Jamie Taylor. Freebase: a collaboratively created graph database for structuring human knowledge. In *Proceedings of the 2008 ACM SIGMOD International Conference on Management of Data*, pages 1247–1250. ACM, 2008. (Cited on page 118.)

Marina Bosi and Richard E. Goldberg. *Introduction to digital audio coding and standards*, volume 721. Springer Science & Business Media, 2012. (Cited on page 53.)

Eric Brochu, Vlad M. Cora, and Nando De Freitas. A tutorial on bayesian optimization of expensive cost functions, with application to active user modeling and hierarchical reinforcement learning. *arXiv preprint arXiv:1012.2599*, 2010. (Cited on page 47.)

Judith C. Brown. Calculation of a constant Q spectral transform. *The Journal of the Acoustical Society of America*, 89(1):425–434, 1991. (Cited on pages 64 and 89.)

Judith C. Brown and Miller S. Puckette. An efficient algorithm for the calculation of a constant Q transform. *The Journal of the Acoustical Society of America*, 92 (5):2698–2701, 1992. (Cited on page 64.)

Chris Cannam, Emmanouil Benetos, Matthias Mauch, Matthew E. P. Davies, Simon Dixon, Christian Landone, Katy Noland, and Dan Stowell. MIREX 2015 entry: Vamp plugins from the centre for digital music. In *11th Music Information Retrieval Evaluation eXchange*, 2015. (Cited on page 165.)

Dave Carlton. I analyzed the chords of 1300 popular songs for patterns. This is what I found. `http://www.hooktheory.com/blog/`, June 2012. (Cited on page 81.)

William Chan, Navdeep Jaitly, Quoc V. Le, and Oriol Vinyals. Listen, attend and spell. *arXiv preprint arXiv:1508.01211*, 2015. (Cited on page 140.)

Kyunghyun Cho, Bart Van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using RNN encoder-decoder for statistical machine translation. *arXiv preprint arXiv:1406.1078*, 2014. (Cited on page 33.)

Kyunghyun Cho, Aaron C. Courville, and Yoshua Bengio. Describing multimedia content using attention-based encoder-decoder networks. *arXiv preprint arXiv:1507.01053*, 2015. (Cited on pages 140 and 142.)

Dan Claudiu Ciresan, Ueli Meier, Luca Maria Gambardella, and Jürgen Schmidhuber. Deep, big, simple neural nets for handwritten digit recognition. *Neural Computation*, 22(12):3207–3220, 2010. (Cited on pages 28 and 31.)

Perry R. Cook. *Music, cognition, and computerized sound: an introduction to psychoacoustics*. MIT Press, 1999. (Cited on page 53.)

Thomas M. Cover and Peter E. Hart. Nearest neighbor pattern classification. *IEEE Transactions on Information Theory*, 13(1):21–27, 1967. (Cited on page 15.)

Michael Scott Cuthbert and Christopher Ariza. `music21`: A toolkit for computer-aided musicology and symbolic music data. In *Proceedings of the 11th International Society for Music Information Retrieval Conference*, pages 637–642, 2010. (Cited on pages 76 and 83.)

Michael Scott Cuthbert, Christopher Ariza, and Lisa Friedland. Feature extraction and machine learning on symbolic music using the `music21` toolkit. In *Proceedings of the 12th International Society for Music Information Retrieval Conference*, pages 387–392, 2011. (Cited on page 76.)

George Cybenko. Approximation by superpositions of a sigmoidal function. *Mathematics of Control, Signals and Systems*, 2(4):303–314, 1989. (Cited on pages 18 and 28.)

Yann N. Dauphin, Harm de Vries, Junyoung Chung, and Yoshua Bengio. RMSProp and equilibrated adaptive learning rates for non-convex optimization. *arXiv preprint arXiv:1502.04390*, 2015. (Cited on page 43.)

Matthew E. P. Davies, Norberto Degara, and Mark D. Plumbley. Evaluation methods for musical audio beat tracking algorithms. Technical Report C4DM-TR-09-06, Queen Mary University of London, 2009. (Cited on page 167.)

Jeremiah D. Deng, Christian Simmermacher, and Stephen Cranefield. A study on feature analysis for musical instrument classification. *IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics*, 38(2):429–438, 2008. (Cited on pages 143 and 148.)

Sander Dieleman, Jan Schlüter, Colin Raffel, Eben Olson, Søren Kaae Sønderby, Daniel Nouri, Daniel Maturana, Martin Thoma, Eric Battenberg, Jack Kelly, Jeffrey De Fauw, Michael Heilman, diogo149, Brian McFee, Hendrik Weideman, takacsg84, peterderivaz, Jon, instagibbs, Dr. Kashif Rasul, CongLiu, Britefury, and Jonas Degrave. Lasagne: First release. `https://github.com/Lasagne/Lasagne`, 2015. (Cited on pages 122, 145, and 200.)

Simon Dixon. Automatic extraction of tempo and beat from expressive performances. *Journal of New Music Research*, 30(1):39–58, 2001. (Cited on page 203.)

Simon Dixon and Gerhard Widmer. MATCH: A music alignment tool chest. In *Proceedings of the 6th International Society for Music Information Retrieval Conference*, pages 492–497, 2005. (Cited on page 89.)

J. Stephen Downie. The music information retrieval evaluation exchange (2005-2007): A window into music information retrieval research. *Acoustical Science and Technology*, 29(4):247–255, 2008. (Cited on page 164.)

John Duchi, Elad Hazan, and Yoram Singer. Adaptive subgradient methods for online learning and stochastic optimization. *The Journal of Machine Learning Research*, 12:2121–2159, 2011. (Cited on page 42.)

Vincent Dumoulin and Francesco Visin. A guide to convolution arithmetic for deep learning. *arXiv preprint arXiv:1603.07285*, 2016. (Cited on page 35.)

Andreas Ehmann, Kris West, Mert Bay, Kahyun Choi, and Yun Hao. MIREX task: Audio key detection. http://music-ir.org/mirex/wiki/2016:Audio_Key_Detection, 2016. (Cited on page 165.)

Daniel P. W. Ellis. Aligning MIDI files to music audio. http://www.ee.columbia.edu/~dpwe/resources/matlab/alignmidi/, 2013. (Cited on pages 89, 90, and 91.)

Dumitru Erhan, Pierre-Antoine Manzagol, Yoshua Bengio, Samy Bengio, and Pascal Vincent. The difficulty of training deep architectures and the effect of unsupervised pre-training. In *International Conference on Artificial Intelligence and Statistics*, pages 153–160, 2009. (Cited on pages 28 and 31.)

Sebastian Ewert, Meinard Müller, Verena Konz, Daniel Müllensiefen, and Geraint A. Wiggins. Towards cross-version harmonic analysis of music. *IEEE Transactions on Multimedia*, 14(3):770–782, 2012. (Cited on pages 83, 85, 87, 89, 90, and 107.)

Sebastian Ewert, Bryan Pardo, Meinard Müller, and Mark Plumbley. Score-informed source separation for musical audio recordings: An overview. *IEEE Signal Processing Magazine*, 31(3):116–124, 2014. (Cited on page 75.)

Peter Foster, Matthias Mauch, and Simon Dixon. Sequential complexity as a descriptor for musical similarity. *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, 22(12):1965–1977, 2014. (Cited on pages 143 and 148.)

Takuya Fujishima. Realtime chord recognition of musical sound: A system using common lisp music. In *Proceedings of the 1999 International Computer Music Conference*, pages 464–467, 1999. (Cited on pages 89 and 204.)

Kunihiko Fukushima. Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position. *Biological Cybernetics*, 36(4):193–202, 1980. (Cited on page 34.)

Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In *International Conference on Artificial Intelligence and Statistics*, pages 249–256, 2010. (Cited on pages 28 and 32.)

Xavier Glorot, Antoine Bordes, and Yoshua Bengio. Deep sparse rectifier neural networks. In *International Conference on Artificial Intelligence and Statistics*, pages 315–323, 2011. (Cited on pages 28 and 29.)

Michael Good. MusicXML for notation and analysis. In Walter B. Hewelett and Eleanor Selfridge-Field, editors, *The virtual score: representation, retrieval, restoration*, volume 12, pages 113–124. MIT Press, 2001. (Cited on page 163.)

Maarten Grachten, Martin Gasser, Andreas Arzt, and Gerhard Widmer. Automatic alignment of music performances with structural differences. In *Proceedings of the 14th International Society for Music Information Retrieval Conference*, pages 607–612, 2013. (Cited on pages 87 and 107.)

Alex Graves. *Supervised sequence labelling with recurrent neural networks*. Springer, 2012. (Cited on page 33.)

Alex Graves. Generating sequences with recurrent neural networks. *arXiv preprint arXiv:1308.0850*, 2013. (Cited on pages 33 and 44.)

Alex Graves, Santiago Fernández, Faustino Gomez, and Jürgen Schmidhuber. Connectionist temporal classification: Labelling unsegmented sequence data with recurrent neural networks. In *Proceedings of the 23rd International Conference on Machine learning*, pages 369–376, 2006. (Cited on page 176.)

Alex Graves, Greg Wayne, and Ivo Danihelka. Neural turing machines. *arXiv preprint arXiv:1410.5401*, 2014. (Cited on page 140.)

Stephen José Hanson and Lorien Y. Pratt. Comparing biases for minimal network construction with back-propagation. In *Advances in Neural Information Processing Systems*, pages 177–185, 1989. (Cited on page 30.)

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 1026–1034, 2015. (Cited on pages 28, 29, 32, 34, 121, and 145.)

Donald Olding Hebb. *The organization of behavior: A neuropsychological theory*. Psychology Press, 1949. (Cited on page 17.)

John R. Hershey, Zhuo Chen, Jonathan Le Roux, and Shinji Watanabe. Deep clustering: Discriminative embeddings for segmentation and separation. In *Proceedings of the 41st IEEE International Conference on Acoustics, Speech, and Signal Processing*, 2016. (Cited on pages 134 and 139.)

Geoffrey E. Hinton and Ruslan R. Salakhutdinov. Reducing the dimensionality of data with neural networks. *Science*, 313(5786):504–507, 2006. (Cited on pages 28 and 31.)

Geoffrey E. Hinton, Simon Osindero, and Yee-Whye Teh. A fast learning algorithm for deep belief nets. *Neural Computation*, 18(7):1527–1554, 2006. (Cited on pages 28 and 31.)

Geoffrey E. Hinton, Nitish Srivastava, Alex Krizhevsky, Ilya Sutskever, and Ruslan R. Salakhutdinov. Improving neural networks by preventing co-adaptation of feature detectors. *arXiv preprint arXiv:1207.0580*, 2012. (Cited on pages 28, 30, and 123.)

Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Computation*, 9(8):1735–1780, 1997. (Cited on pages 33, 140, and 141.)

Kurt Hornik, Maxwell Stinchcombe, and Halbert White. Multilayer feedforward networks are universal approximators. *Neural Networks*, 2(5):359–366, 1989. (Cited on pages 18 and 28.)

Ning Hu, Roger B. Dannenberg, and George Tzanetakis. Polyphonic audio matching and alignment for music retrieval. In *IEEE Workshop on Applications of Signal Processing to Audio and Acoustics*, pages 185–188, 2003. (Cited on pages 85, 89, and 90.)

Eric J. Humphrey and Juan P. Bello. Rethinking automatic chord recognition with convolutional neural networks. In *Proceedings of the 11th International Conference on Machine Learning and Applications (ICMLA)*, volume 2, pages 357–362, 2012. (Cited on pages 35, 121, and 147.)

Intel. SSE4 programming reference. Technical Report D91561-003, 2007. (Cited on pages 111 and 114.)

The International MIDI Association. MIDI musical instrument digital interface specification 1.0. 1983. (Cited on page 71.)

The International MIDI Association. Standard MIDI files. 1988. (Cited on page 71.)

The International MIDI Association. General MIDI level 1 specification. 1991. (Cited on page 71.)

Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *Proceedings of the 32nd International Conference on Machine Learning*, pages 448–456, 2015. (Cited on page 28.)

Herbert Jaeger. Long short-term memory in echo state networks: Details of a simulation study. Technical Report 27, Jacobs University, 2012. (Cited on page 33.)

Andreas Jansson, Colin Raffel, and Tillman Weyde. This is my jam - data dump. In *16th International Society for Music Information Retrieval Conference Late Breaking and Demo Papers*, 2015. (Cited on page 175.)

Kevin Jarrett, Koray Kavukcuoglu, Marc'Aurelio Ranzato, and Yann LeCun. What is the best multi-stage architecture for object recognition? In *Proceedings of the 12th IEEE International Conference on Computer Vision*, pages 2146–2153, 2009. (Cited on page 29.)

Eric Jones, Travis Oliphant, and Pearu Peterson. SciPy: Open source scientific tools for Python. 2014. (Cited on page 109.)

Herman Kamper, Weiran Wang, and Karen Livescu. Deep convolutional acoustic word embeddings using word-pair side information. In *Proceedings of the 41st IEEE International Conference on Acoustics, Speech, and Signal Processing*, 2016. (Cited on pages 134 and 139.)

Maurice G. Kendall. A new measure of rank correlation. *Biometrika*, pages 81–93, 1938. (Cited on page 100.)

Eamonn Keogh, Kaushik Chakrabarti, Michael Pazzani, and Sharad Mehrotra. Dimensionality reduction for fast similarity search in large time series databases. *Knowledge and Information Systems*, 3(3):263–286, 2001. (Cited on pages 110 and 125.)

Yoon Kim. Convolutional neural networks for sentence classification. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing*, pages 1746–1751, 2014. (Cited on page 35.)

Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *Proceedings of the 3rd International Conference on Learning Representations*, 2015. (Cited on pages 44, 123, and 148.)

Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. ImageNet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems*, pages 1097–1105, 2012. (Cited on page 34.)

Joseph B Kruskal. Multidimensional scaling by optimizing goodness of fit to a nonmetric hypothesis. *Psychometrika*, 29(1):1–27, 1964. (Cited on page 123.)

Quoc V. Le, Navdeep Jaitly, and Geoffrey E. Hinton. A simple way to initialize recurrent networks of rectified linear units. *arXiv preprint arXiv:1504.00941*, 2015. (Cited on page 33.)

Yann LeCun, Bernhard Boser, John S. Denker, Donnie Henderson, Richard E. Howard, Wayne Hubbard, and Lawrence D. Jackel. Backpropagation applied to handwritten zip code recognition. *Neural Computation*, 1(4):541–551, 1989. (Cited on page 34.)

Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *Nature*, 521 (7553):436–444, 2015. (Cited on page 28.)

Yann A. LeCun, Léon Bottou, Genevieve B. Orr, and Klaus-Robert Müller. Efficient backprop. In *Neural Networks: Tricks of the trade*, pages 9–48. Springer, 2012. (Cited on pages 28, 32, 38, 40, and 42.)

Andrew L. Maas, Awni Y. Hannun, and Andrew Y. Ng. Rectifier nonlinearities improve neural network acoustic models. In *Proceedings of the 30th International Conference on Machine Learning*, volume 30, page 1, 2013. (Cited on page 29.)

Michael I. Mandel and Daniel P. W. Ellis. Song-level features and support vector machines for music classification. In *Proceedings of the 6th International Conference on Music Information Retrieval*, pages 594–599, 2005. (Cited on pages 143 and 148.)

James Martens and Ilya Sutskever. Learning recurrent neural networks with hessian-free optimization. In *Proceedings of the 28th International Conference on Machine Learning*, pages 1033–1040, 2011. (Cited on page 33.)

Jonathan Masci, Michael M. Bronstein, Alexander M. Bronstein, and Jürgen Schmidhuber. Multimodal similarity-preserving hashing. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 36(4):824–830, 2014. (Cited on pages 113, 114, 121, and 122.)

Matthias Mauch and Simon Dixon. A corpus-based study of rhythm patterns. In *Proceedings of the 13th International Society for Music Information Retrieval Conference*, pages 163–168, 2012. (Cited on pages 80, 84, and 174.)

Matthias Mauch, Chris Cannam, Matthew Davies, Simon Dixon, Christopher Harte, Sefki Kolozali, Dan Tidhar, and Mark Sandler. OMRAS2 metadata project 2009. In *10th International Society for Music Information Retrieval Conference Late Breaking and Demo Papers*, 2009. (Cited on page 164.)

Warren S. McCulloch and Walter Pitts. A logical calculus of the ideas immanent in nervous activity. *The Bulletin of Mathematical Biophysics*, 5(4):115–133, 1943. (Cited on page 17.)

Brian McFee, Eric J. Humphrey, and Juan P. Bello. A software framework for musical data augmentation. In *Proeedings of the 16th International Society for Music Information Retrieval Conference*, 2015a. (Cited on pages 31, 35, and 107.)

Brian McFee, Matt McVicar, Colin Raffel, Dawen Liang, Oriol Nieto, Josh Moore, Dan Ellis, Douglas Repetto, Petr Viktorin, and João Felipe Santos. `librosa`: v0.4.0. `https://github.com/bmcfee/librosa`, 2015b. (Cited on pages 98 and 119.)

Brian McFee, Colin Raffel, Dawen Liang, Daniel P. W. Ellis, Matt McVicar, Eric Battenberg, and Oriol Nieto. librosa: Audio and music signal analysis in python. In *Proceedings of the 14th Python in Science Conference*, 2015c. (Cited on pages 98 and 119.)

Cory McKay and Ichiro Fujinaga. jSymbolic: A feature extractor for MIDI files. In *Proceedings of the International Computer Music Conference*, pages 302–305, 2006. (Cited on page 75.)

Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S. Corrado, and Jeff Dean. Distributed representations of words and phrases and their compositionality. In *Advances in Neural Information Processing Systems*, pages 3111–3119, 2013. (Cited on page 138.)

Tomas Mikolov, Armand Joulin, Sumit Chopra, Michael Mathieu, and Marc'Aurelio Ranzato. Learning longer memory in recurrent neural networks. *arXiv preprint arXiv:1412.7753*, 2014. (Cited on page 33.)

Dmytro Mishkin and Jiri Matas. All you need is a good init. *arXiv preprint arXiv:1511.06422*, 2015. (Cited on page 32.)

F. Richard Moore. The dysfunctions of MIDI. *Computer music journal*, 12(1):19–28, 1988. (Cited on page 71.)

Alexander Mordvintsev, Michael Tyka, and Christopher Olah. Inceptionism: Going deeper into neural networks. `http://googleresearch.blogspot.com/2015/06/inceptionism-going-deeper-into-neural.html`, June 2015. (Cited on pages 151 and 152.)

Meinard Müller. Dynamic time warping. *Information Retrieval for Music and Motion*, pages 69–84, 2007. (Cited on pages 66, 67, and 107.)

Vinod Nair and Geoffrey E. Hinton. Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th International Conference on Machine Learning*, pages 807–814, 2010. (Cited on page 29.)

Yurii Nesterov. A method of solving a convex programming problem with convergence rate $O(1/\sqrt{2})$. In *Soviet Mathematics Doklady*, volume 27, pages 372–376, 1983. (Cited on pages 42, 123, and 148.)

Michael A. Nielsen. *Neural Networks and Deep Learning*. `http://neuralnetworksanddeeplearning.com`, 2015. (Cited on page 20.)

Katy Noland and Mark Sandler. Signal processing parameters for tonality estimation. In *Proceedings of the 122nd Audio Engineering Society Convention*, 2007. (Cited on page 165.)

Panagiotis Papapetrou, Vassilis Athitsos, Michalis Potamias, George Kollios, and Dimitrios Gunopulos. Embedding-based subsequence matching in time-series databases. *ACM Transactions on Database Systems*, 36(3):17, 2011. (Cited on page 138.)

Razvan Pascanu, Tomas Mikolov, and Yoshua Bengio. On the difficulty of training recurrent neural networks. In *Proceedings of the 30th International Conference on Machine Learning*, pages 1310–1318, 2013. (Cited on pages 33 and 140.)

Graham E. Poliner and Daniel P. W. Ellis. A discriminative model for polyphonic piano transcription. *EURASIP Journal on Applied Signal Processing*, 2007(1): 154–154, 2007. (Cited on page 174.)

Boris T. Polyak. Some methods of speeding up the convergence of iteration methods. *USSR Computational Mathematics and Mathematical Physics*, 4(5):1–17, 1964. (Cited on page 41.)

Pedro José Ponce de León Amador, José Manuel Iñesta Quereda, and David Rizo Valero. Mining digital music score collections: melody extraction and genre recognition. In Peng-Yeng Yin, editor, *Pattern Recognition Techniques, Technology and Applications*, chapter 25, pages 559–590. InTech, 2008. (Cited on page 83.)

Lutz Prechelt. Early stopping - but when? In *Neural Networks: Tricks of the Trade*, pages 53–67. Springer, 2012. (Cited on page 30.)

Colin Raffel and Daniel P. W. Ellis. Intuitive analysis, creation and manipulation of MIDI data with `pretty_midi`. In *15th International Society for Music Information Retrieval Conference Late Breaking and Demo Papers*, 2014. (Cited on pages 84, 93, 119, and 124.)

Colin Raffel and Daniel P. W. Ellis. Accelerating multimodal sequence retrieval with convolutional networks. In *NIPS Multimodal Machine Learning Workshop*, 2015a. (Cited on page 196.)

Colin Raffel and Daniel P. W. Ellis. Large-scale content-based matching of MIDI and audio files. In *Proceedings of the 16th International Society for Music Information Retrieval Conference*, pages 234–240, 2015b. (Cited on pages 89, 90, 91, 99, 122, 195, and 196.)

Colin Raffel and Daniel P. W. Ellis. Extracting ground-truth information from MIDI files: A MIDIfesto. In *Proceedings of the 17th International Society for Music Information Retrieval Conference (to appear)*, 2016a. (Cited on page 194.)

Colin Raffel and Daniel P. W. Ellis. Optimizing DTW-based audio-to-MIDI alignment and matching. In *Proceedings of the 41st IEEE International Conference on Acoustics, Speech, and Signal Processing*, pages 81–85, 2016b. (Cited on pages 85 and 194.)

Colin Raffel and Daniel P. W. Ellis. Pruning subsequence search with attention-based embedding. In *Proceedings of the 41st IEEE International Conference on Acoustics, Speech, and Signal Processing*, pages 554–558, 2016c. (Cited on page 196.)

Colin Raffel, Brian McFee, Eric J. Humphrey, Justin Salamon, Oriol Nieto, Dawen Liang, and Daniel P. W. Ellis. `mir_eval`: A transparent implementation of common MIR metrics. In *Proceedings of the 15th International Society for Music Information Retrieval Conference*, pages 367–372, 2014. (Cited on page 165.)

Thanawin Rakthanmanon, Bilson Campana, Abdullah Mueen, Gustavo Batista, Brandon Westover, Qiang Zhu, Jesin Zakaria, and Eamonn Keogh. Searching and mining trillions of time series subsequences under dynamic time warping. In *Proceedings of the 18th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 262–270, 2012. (Cited on pages 90 and 109.)

Carl Edward Rasmussen. *Gaussian Processes for Machine Learning*. MIT Press, 2006. (Cited on page 49.)

Chotirat Ann Ratanamahatana and Eamonn Keogh. Everything you know about dynamic time warping is wrong. In *3rd Workshop on Mining Temporal and Sequential Data*, 2004. (Cited on page 91.)

Christophe Rhodes, David Lewis, and Daniel Müllensiefen. Bayesian model selection for harmonic labelling. In *Mathematics and Computation in Music*, pages 107–116. Springer, 2007. (Cited on page 83.)

Frank Rosenblatt. The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological Review*, 65(6):386, 1958. (Cited on page 17.)

Sebastian Ruder. An overview of gradient descent optimization algorithms. `http://sebastianruder.com/optimizing-gradient-descent`, 2016. (Cited on page 39.)

David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. Learning representations by back-propagating errors. *Nature*, 323:533–536, 1986. (Cited on pages 19 and 20.)

Tara N. Sainath, Abdel-rahman Mohamed, Brian Kingsbury, and Bhuvana Ramabhadran. Deep convolutional neural networks for LVCSR. In *Proceedings of the 38th IEEE International Conference on Acoustics, Speech and Signal Processing*, pages 8614–8618, 2013. (Cited on page 34.)

Hiroaki Sakoe and Seibi Chiba. Dynamic programming algorithm optimization for spoken word recognition. *IEEE Transactions on Acoustics, Speech and Signal Processing*, 26(1):43–49, 1978. (Cited on pages 66, 67, and 107.)

Stan Salvador and Philip Chan. Toward accurate dynamic time warping in linear time and space. *Intelligent Data Analysis*, 11(5):561–580, 2007. (Cited on pages 110 and 176.)

Andrew M. Saxe, James L. McClelland, and Surya Ganguli. Exact solutions to the nonlinear dynamics of learning in deep linear neural networks. *arXiv preprint arXiv:1312.6120*, 2013. (Cited on page 32.)

Tom Schaul, Ioannis Antonoglou, and David Silver. Unit tests for stochastic optimization. *arXiv preprint arXiv:1312.6055*, 2013. (Cited on page 39.)

Alexander Schindler, Rudolf Mayer, and Andreas Rauber. Facilitating comprehensive benchmarking experiments on the million song dataset. In *Proceedings of the 13th International Society for Music Information Retrieval Conference*, pages 469–474, 2012. (Cited on pages 108, 152, and 197.)

Jan Schlüter and Sebastian Böck. Improved musical onset detection with convolutional neural networks. In *Proceedings of the 39th IEEE International Conference on Acoustics, Speech and Signal Processing*, pages 6979–6983, 2014. (Cited on page 35.)

Jan Schlüter and Thomas Grill. Exploring data augmentation for improved singing voice detection with neural networks. In *Proceedings of the 16th International Society for Music Information Retrieval Conference*, 2015. (Cited on pages 31 and 35.)

Christian Schörkhuber and Anssi Klapuri. Constant-Q transform toolbox for music processing. In *7th Sound and Music Computing Conference*, pages 3–64, 2010. (Cited on pages 64 and 65.)

Hendrik Schreiber. Improving genre annotations for the million song dataset. In *Proceedings of the 16th International Society for Music Information Retrieval Conference*, 2015. (Cited on page 175.)

Florian Schroff, Dmitry Kalenichenko, and James Philbin. Facenet: A unified embedding for face recognition and clustering. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 815–823, 2015. (Cited on pages 134 and 137.)

Joan Serrà, Álvaro Corral, Marián Boguñá, Martín Haro, and Josep Ll Arcos. Measuring the evolution of contemporary western popular music. *Scientific Reports*, 2, 2012. (Cited on page 174.)

Siddharth Sigtia, Emmanouil Benetos, and Simon Dixon. An end-to-end neural network for polyphonic music transcription. *arXiv preprint arXiv:1508.01774*, 2015. (Cited on page 174.)

Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014. (Cited on page 121.)

Daniel Sleator and David Temperley. The `melisma` music analyzer. `http://www.link.cs.cmu.edu/melisma`, 2001. (Cited on page 83.)

Julius O. Smith. *Mathematics of the Discrete Fourier Transform (DFT with Audio Applications)*. `https://ccrma.stanford.edu/~jos/st/`, 2007. (Cited on page 58.)

Julius O. Smith. *Spectral Audio Signal Processing*. `https://ccrma.stanford.edu/~jos/sasp/`, 2011. (Cited on page 58.)

Jasper Snoek, Hugo Larochelle, and Ryan P. Adams. Practical bayesian optimization of machine learning algorithms. In *Advances in Neural Information Processing Systems*, pages 2951–2959, 2012. (Cited on pages 47, 48, 52, 97, 122, and 145.)

Sainbayar Sukhbaatar, Arthur Szlam, Jason Weston, and Rob Fergus. End-to-end memory networks. *arXiv preprint arXiv:1503.08895*, 2015. (Cited on page 140.)

Ilya Sutskever, James Martens, George Dahl, and Geoffrey Hinton. On the importance of initialization and momentum in deep learning. In *Proceedings of the 30th International Conference on Machine Learning*, pages 1139–1147, 2013. (Cited on pages 28, 31, 32, 33, 41, and 42.)

Ilya Sutskever, Oriol Vinyals, and Quoc V. Le. Sequence to sequence learning with neural networks. In *Advances in Neural Information Processing Systems*, pages 3104–3112, 2014. (Cited on page 139.)

Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1–9, 2015. (Cited on page 34.)

Michael Tang, Yip Chi Lap, and Ben Kao. Selection of melody lines for music databases. In *Proceedings of the 24th International Computer Software and Applications Conference*, pages 243–248, 2000. (Cited on page 83.)

Tijmen Tieleman and Geoffrey Hinton. Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude. *COURSERA: Neural Networks for Machine Learning*, 4:2, 2012. (Cited on pages 43 and 120.)

Derek Tingle, Youngmoo E. Kim, and Douglas Turnbull. Exploring automatic music annotation with "acoustically-objective" tags. In *Proceedings of the International Conference on Multimedia Information Retrieval*, pages 55–62. ACM, 2010. (Cited on page 118.)

Robert J. Turetsky and Daniel P. W. Ellis. Ground-truth transcriptions of real music from force-aligned MIDI syntheses. In *Proceedings of the 4th International Society for Music Information Retrieval Conference*, pages 135–141, 2003. (Cited on pages 75, 89, and 90.)

Douglas Turnbull, Luke Barrington, David Torres, and Gert Lanckriet. Towards musical query-by-semantic-description using the CAL500 data set. In *Proceedings of the 30th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 439–446. ACM, 2007. (Cited on page 118.)

George Tzanetakis and Perry Cook. Musical genre classification of audio signals. *IEEE Transactions on Speech and Audio Processing*, 10(5):293–302, 2002. (Cited on pages 143 and 148.)

Karen Ullrich, Jan Schlüter, and Thomas Grill. Boundary detection in music structure analysis using convolutional neural networks. In *Proceedings of the 15th International Society for Music Information Retrieval Conference*, 2014. (Cited on page 35.)

Aaron Van den Oord, Sander Dieleman, and Benjamin Schrauwen. Deep content-based music recommendation. In *Advances in Neural Information Processing Systems*, pages 2643–2651, 2013. (Cited on page 35.)

Oriol Vinyals, Meire Fortunato, and Navdeep Jaitly. Pointer networks. In *Advances in Neural Information Processing Systems*, pages 2674–2682, 2015. (Cited on page 176.)

Stefan Wager, Sida Wang, and Percy S. Liang. Dropout training as adaptive regularization. In *Advances in Neural Information Processing Systems*, pages 351–359, 2013. (Cited on page 30.)

Paul J. Werbos. Backpropagation through time: what it does and how to do it. *Proceedings of the IEEE*, 78(10):1550–1560, 1990. (Cited on page 33.)

Kelvin Xu, Jimmy Ba, Ryan Kiros, Aaron Courville, Ruslan Salakhutdinov, Richard Zemel, and Yoshua Bengio. Show, attend and tell: Neural image caption generation with visual attention. *arXiv preprint arXiv:1502.03044*, 2015. (Cited on page 140.)

Huei-Fang Yang, Kevin Lin, and Chu-Song Chen. Supervised learning of semantics-preserving hashing via deep neural networks for large-scale image search. *arXiv preprint arXiv:1507.00101*, 2015. (Cited on pages 123, 138, and 147.)

Byoung-Kee Yi and Christos Faloutsos. Fast time sequence indexing for arbitrary $\mathcal{L}_p$ norms. In *Proceedings of the 26th International Conference on Very Large Data Bases*, pages 385–394, 2000. (Cited on pages 110 and 125.)

Matthew D. Zeiler and Rob Fergus. Visualizing and understanding convolutional networks. In *13th European Conference on Computer Vision*, pages 818–833, 2014. (Cited on page 37.)

Xiang Zhang and Yann LeCun. Text understanding from scratch. *arXiv preprint arXiv:1502.01710*, 2015. (Cited on page 35.)

Appendix A

Relevant Publications

Much of the work presented in this thesis has been previously published in peer-reviewed venues. However, in various places the experiments have been improved and results have been subsequently updated. In this appendix, we reference these publications in approximate order of appearance in this thesis and outline how the work presented here differs.

Extracting Ground-Truth Information from MIDI Files

Chapter 3 and section 7.2 appear together in a paper to appear at the 17th International Society for Music Information Retrieval conference (Raffel and Ellis, 2016a). They are presented in this thesis largely unchanged, except for some editing to make them better fit into the storyline.

Optimizing DTW-Based Audio-to-MIDI Alignment and Matching

Much of chapter 4 was published in a paper at the 41st International Conference on Acoustics, Speech, and Signal Processing (Raffel and Ellis, 2016b). For better explanation of the results, we added figures 4.1 to 4.4 and tables 4.1 and 4.2. We also added the normalized DTW distance re-scaling step defined

in equation (4.6), which makes it more readily interpretable as a confidence score.

Large-Scale Content-Based Matching of MIDI and Audio Files

The approach of chapter 5 was originally proposed in a paper at the 16th International Society for Music Information Retrieval Conference (Raffel and Ellis, 2015b). However, it is presented in this thesis with many improvements. First, in (Raffel and Ellis, 2015b), beat-synchronous spectrograms are used (i.e. the beats of the music signal are estimated and frames in the spectrogram within each beat are aggregated). This caused issues when the beats of a given MIDI file were estimated differently than its correct match. We therefore forewent beat synchronization and instead used a finer, fixed timescale. This prompted the use of three maxpooling-by-2 layers, resulting in downsampling in time by 8, rather than the two maxpooling layers used in (Raffel and Ellis, 2015b). Second, we utilized the "gold standard' system of section 4.4.1 to create the training data rather than the alignment scheme proposed in (Raffel and Ellis, 2015b) for consistency and due to its better performance. Third, we used 32-bit binary vectors rather than 16-bit, which potentially gives them more expressive power. It also resulted in the use of the POPCNT instruction being more efficient than the pre-computed table look-up used in (Raffel and Ellis, 2015b). Finally, we experimented with a wider variety of system designs, all of which are documented in section 5.2.2. This included the output L^2 penalty, 2×1 pooling, and the larger number of small 3×3 convolutional layers, rather than the 5×12 input filter size. Taken together, these changes resulted in dramatic increases in precision; specifically, we were improved our high-confidence pruning threshold from 10,000 to 250. In addition, for better framing of our results, we added the baseline learning-free approach described

in section 5.2.4.

Accelerating Multimodal Sequence Retrieval with Convolutional Networks

In chapter 5, we also include the "multimodal" variant of the experiment which was first explored in a paper at the 2015 Workshop on Multi-Modal Machine Learning (Raffel and Ellis, 2015a). We improved upon this approach in the same way as that of (Raffel and Ellis, 2015b), which also resulted in substantial performance gains.

Pruning Subsequence Search with Attention-Based Embedding

Our feed-forward attention-based sequence embedding model was first proposed in a paper at the 41st International Conference on Acoustics, Speech, and Signal Processing (Raffel and Ellis, 2016c). As presented in chapter 6, it is largely unchanged, except that we present a broader overview of previous work on attention and embedding and remove sections which are redundant with chapter 5. We also re-performed the hyperparameter optimization and experimented with various other system design choices as outlined in section 6.2.1.

Appendix B

Software Prepared

Over the course of my PhD, I wrote over 50,000 lines of Python code, all of which are open-source and publicly available online.[1] This includes the requisite code for running the experiments in this thesis and my other research projects, in addition to contributions to more generally-applicable software libraries. This is arguably the main product of my PhD work, so in this appendix I give an overview of the software utilized in this thesis which I developed or helped develop.

B.1 `midi-dataset`

The repository `midi-dataset`[2] is a collection of code which implements all of the experiments in chapters 5 to 7. Specifically, it includes the following scripts, in order of their intended use:

`create_msd_cqts.py` Pre-computes constant-Q spectrograms for all of the 7digital preview clips (Schindler et al., 2012) from the entire Million Song Dataset

[1] http://github.com/craffel
[2] http://github.com/craffel/midi-dataset

(Bertin-Mahieux et al., 2011) in the format described in section 5.2.2 and used throughout this thesis.

`create_whoosh_indices.py` Builds search indices for the Python library `whoosh` of all datasets used in these experiments (the MSD, the "clean MIDI subset", uspop2002, CAL10k, and CAL500) to allow for the fuzzy text-based matching utilized in section 5.2.1.

`text_match_datasets.py` Runs fuzzy text matching of the clean MIDI subset against each of the aforementioned audio datasets using `whoosh` in order to obtain candidate matches.

`align_text_matches.py` Runs the audio-to-MIDI alignment process described in section 4.4.1 on all of the text matches to find true ground truth MIDI-audio matches as described in section 5.2.1.

`split_training_data.py` Creates the fixed train/validation/development/test split *by song* of the ground truth MIDI-audio matches.

`experiments` A collection of experiments for evaluating all of the matching techniques we proposed and described. Each experiment optionally includes a script `parameter_search.py` which optimizes the parameters of the system using the training and validation sets. All experiments include scripts `precompute.py` and `match_msd.py` which respectively pre-compute all necessary data for the experiment (for example, fixed-length embeddings of the entire MSD and clean MIDI test set) and evaluate the matching on the test set. Experiments included are:

`dhs` for the learning-based method for creating downsampled hash sequences described in section 5.2.2.

`dhs_piano` for the corresponding experiment using piano rolls as a MIDI representation rather than synthesized spectrograms, described in section 5.2.3.

`tpaa` for the "thresholded piecewise aggregate approximation" baseline described in section 5.2.4.

`pse` for the learning-based method for producing similarity-preserving embeddings of variable-length sequences using a feed-forward network with attention described in section 6.2.

`stats` for the feature statistics-based embedding baseline described in section 6.2.2.

`combined` for the experiment from chapter 7 evaluating the performance of the combination of the above learning-based methods and the DTW-based matching system described in section 4.4.1.

`match.py` Matches one or more MIDI files to the Million Song Dataset quickly and accurately using the cascade of methods proposed in this thesis, as described in chapter 7.

Also included are `experiment_utils.py`, `feature_extraction.py`, and `whoosh_search.py` which contain shared helper functions for the experiments, extracting features, and text-based matching respectively.

B.1.1 `dhs`

To facilitate its usage in contexts outside of the experiments in this thesis, the code for the learning-based downsampled hash sequence technique proposed in chapter 5 is factored out in a separate library called `dhs`.[3] `dhs` includes the following functionality:

[3] http://github.com/craffel/dhs

`train.py` Given a dataset of sequential data with known-matching pairs and an un-trained pair of neural network models, train the networks by optimizing the similarity-preserving loss equation (5.2).

`match.py` Given hash sequences represented as lists of integers, rapidly compute the pairwise distance matrix using exclusive-or and POPCNT operations and the DTW distance corresponding to the lowest-cost path through this matrix.

`utils.py` Tools for iterating over sequential data and constructing "dissimilar" pairs by randomly swapping matching pairs.

This library makes use of `theano` (Bergstra et al., 2010; Bastien et al., 2012; Al-Rfou et al., 2016) and `lasagne` (Dieleman et al., 2015) to construct the loss function, automatically compute its derivative, and optimize the networks' parameters with respect to it. It also utilizes `numba`[4] to rapidly compute the DTW distance and `cython`[5] to access the SSE 4.2 POPCNT instruction directly from Python code. While thus far this library has only been used for the experiments in this thesis, we hope it will be facilitate research into other problem settings.

B.1.2 `pse`

In the same vein as `dhs`, `pse`[6] is a library which covers the pairwise sequence embedding approach of chapter 6. In general, it has very similar functionality to the `dhs` library except that it utilizes non-aligned pairs of sequences, it expects the networks to produce a single vector for each input sequence, and it does not contain functionality for matching the resulting embeddings because this can be achieved by

[4] http://numba.pydata.org
[5] http://cython.org
[6] http://github.com/craffel/pse

a simple Euclidean distance calculation. It also includes an implementation of the feed-forward attention mechanism described in section 6.1.3.

B.2 `alignment-search`

Our code is also available online for the experiments in chapter 4.[7] This includes the following scripts, in order of their intended usage:

`create_data.py` Given a collection of MIDI files, creates "easy" and "hard" artificial synthetic corrupted versions as described in section 4.2.

`parameter_experiment_random.py` **and** `parameter_experiment_gp.py` Optimize DTW system hyperparameters over the "easy" synthetic dataset, first randomly and then using Bayesian optimization, as described in section 4.3.

`find_best_aligners.py` Finds the best-performing alignment system produced by hyperparameter optimization and all of the systems which were not significantly worse.

`confidence_experiment.py` Given these high-performing systems, determine which of the confidence score reporting schemes discussed in section 4.4 best correlates with alignment error over both the "easy" and "hard' datasets.

`align_real_world.py` Aligns a collection of real-world data using the "gold standard" system of section 4.4.1 for the qualitative evaluation in section 4.5.

It also includes the collections of utility functions `corrupt_midi.py`, `align_dataset.py`, and `db_utils.py` respectively for applying synthetic corruptions to MIDI files, aligning a collection of MIDI files given DTW system parameters, and for storing and loading results.

[7] http://github.com/craffel/alignment-search

B.3 pretty_midi

Because MIDI files utilize a low-level binary protocol to represent musical scores, manipulating MIDI data directly can be cumbersome. In addition, the timing of an event in MIDI is represented by tempo-dependent "ticks" relative to the previous event, making direct interpretation in terms of absolute time (in seconds) difficult. We believe that the most intuitive representation is a hierarchical one, consisting of a list of instruments, each of which contains a sequence of events (notes, pitch bends, etc.). This is analogous to a per-instrument piano roll, the visualization commonly used for manipulating MIDI data in digital audio workstations.

Most software libraries for dealing with MIDI files either represent MIDI data in a lower-level (directly corresponding to the bit-level representation) or a higher-level manner (only as musical features). This can make simple manipulations and analysis either require a great deal of source code and expertise or, in the case of modifications to aspects not supported by an abstract library, impossible. For example, in the python-midi module,[8] shifting up the pitch of all notes in a MIDI file by 2 semitones takes only a few lines of code. However, constructing a piano roll representation can take a few hundred lines of code because MIDI ticks must be converted to time in seconds, note-on events must be paired with note-offs, drum events must be ignored, and so on.

Based on these issues, we created the Python module `pretty_midi`[9] for creating, manipulating and analyzing MIDI files. It is intended to make the most common operations applied to MIDI data as straightforward and simple as possible. The module includes functionality for parsing and writing MIDI files, creating and manipulating MIDI data, synthesis, and information extraction.

[8] http://github.com/vishnubob/python-midi
[9] http://github.com/craffel/pretty-midi

`pretty_midi` represents MIDI data as a hierarchy of classes. At the top is the `PrettyMIDI` class, which contains global information such as tempo changes and the MIDI resolution. It also contains a list of `Instrument` class instances. Each `Instrument` is specified by a program number and a flag denoting whether it is a drum instrument. `Instrument` class instances also contain three lists, one each for `Note`, `PitchBend`, and `ControlChange` class instances. The `Note` class is a container for MIDI notes, with velocity, pitch, and start and end time attributes. Similarly, the `PitchBend` and `ControlChange` classes simply have attributes for the bend or control change's time and value.

The top-level `PrettyMIDI` class can be instantiated with a path to an existing MIDI file, in which case the class will be populated by parsing the file. It can also be instantiated without a pre-existing file for creating MIDI data from scratch. For output, the `PrettyMIDI` class has a `write` function which exports its data to a valid MIDI file.

`PrettyMIDI` class instances have functions for performing analysis on the data they contain, some of which have a corresponding function in the `Instrument` class. Some of the implemented functions include:

`get_tempo_changes` Returns a list of the times and tempo (in beats per minute) of all MIDI tempo change events

`estimate_tempo` Returns an empirical tempo estimate according to inner-onset intervals, as described in (Dixon, 2001).

`get_beats` Returns a list of beat locations based on the MIDI tempo change events.

`get_onsets` Returns a list of all of the onsets (start times) of each MIDI note.

`get_piano_roll` Returns a piano roll matrix representation of MIDI notes, as used in section 5.2.3.

`get_chroma` Computes chroma features (also known as pitch class profile) (Fujishima, 1999) of the MIDI data based on the piano roll representation.

In `pretty_midi`, MIDI data can be synthesized as audio using either the `synthesize` or `fluidsynth` methods. `synthesize` uses a periodic function (e.g. sin) to synthesize the each note, while `fluidsynth` utilizes the `fluidsynth` program[10] which performs General MIDI synthesis using a SoundFont file. Both methods return arrays of audio samples at some specified sampling rate.

`pretty_midi` also has utility functions for converting between representations of MIDI notes (name, note number, frequency in Hz, and drum name for percussion instruments), program number and instrument name/class (according to the General MIDI standard) and pitch bend value and semitones. Each `PrettyMIDI` class instance can also readily convert between MIDI ticks and absolute seconds. These functions allow for semantically meaningful creation and representation of MIDI data.

[10] http://www.fluidsynth.org