UC Berkeley
UC Berkeley Electronic Theses and Dissertations

Title
Techniques for Machine Understanding of Live Drum Performances

Permalink
https://escholarship.org/uc/item/6jf2g52n

Author
Battenberg, Eric Dean

Publication Date
2012

Peer reviewed|Thesis/dissertation

eScholarship.org Powered by the California Digital Library
University of California

Techniques for Machine Understanding of Live Drum Performances

by

Eric Dean Battenberg

A dissertation submitted in partial satisfaction of the
requirements for the degree of
Doctor of Philosophy

in

Engineering - Electrical Engineering and Computer Sciences

in the

Graduate Division

of the

University of California, Berkeley

Committee in charge:

Professor Nelson Morgan, Chair
Professor David Wessel
Professor Kurt Keutzer

Fall 2012

Techniques for Machine Understanding of Live Drum Performances

Copyright 2012
by
Eric Dean Battenberg

Abstract

Techniques for Machine Understanding of Live Drum Performances

by

Eric Dean Battenberg

Doctor of Philosophy in Engineering - Electrical Engineering and Computer Sciences

University of California, Berkeley

Professor Nelson Morgan, Chair

This dissertation covers machine listening techniques for the automated real-time analysis of live drum performances. Onset detection, drum detection, beat tracking, and drum pattern analysis are combined into a system that provides rhythmic information useful in performance analysis, synchronization, and retrieval. The techniques are designed with real-time use in mind but can easily be adapted for offline batch use for large scale rhythm analysis.

At the front end of the system, onset and drum detection provide the locations, types, and amplitudes of percussive events. The onset detector uses an adaptive, causal threshold in order to remain robust to large dynamic swings.

For drum detection, a gamma mixture model is used to compute multiple spectral templates per drum onto which onset events can be decomposed using a technique based on non-negative matrix factorization. Unlike classification-based approaches to drum detection, this approach provides amplitude information which is invaluable in the analysis of rhythm. In addition, the decay of drum events are modeled using "tail" templates , which when used with multiple spectral templates per drum, reduce detection errors by 42%.

The beat tracking component uses multiple period hypotheses and an ambiguity measure in order to choose a reliable pulse estimate. Results show that using multiple hypotheses significantly improves tracking accuracy compared to a single period model.

The drum pattern analysis component uses the amplitudes of the detected drum onsets and the metric grid defined by the beat tracker as inputs to a generatively pre-trained deep neural network in order to estimate high-level rhythmic information. The network is tested with beat alignment tasks, including downbeat detection, and reduces alignment errors compared to a simple template correlation approach by up to 59%.

To my parents.

Contents

Contents **ii**

1 Introduction **1**
1.1 Applications . . . 1
1.2 Multimedia Content Analysis . . . 2
1.3 System Overview . . . 3

2 Onset Detection **4**
2.1 What is an Onset? . . . 4
2.2 Common Approaches to Onset Detection . . . 4
2.3 Real-Time Onset Detection for Drums . . . 6
2.4 Onset Detection Accuracy . . . 8

3 Drum Detection **10**
3.1 Introduction . . . 10
3.2 Approaches to Drum Detection . . . 10
3.3 System Overview . . . 11
3.4 Extraction of Spectrogram Slices . . . 12
3.5 Training drum templates . . . 14
3.5.1 Clustering with the Itakura-Saito Divergence . . . 14
3.5.2 The Gamma Distribution . . . 15
3.5.3 The Gamma Mixture Model . . . 16
3.5.4 Agglomerative Clustering with Gamma Mixture Models . . . 18
3.6 Decomposing Drum Onsets . . . 19
3.6.1 Non-negative matrix factorization . . . 19
3.6.2 Non-negative vector decomposition . . . 20
3.7 Drum Detection Evaluation . . . 21
3.7.1 Test Setup . . . 21
3.7.2 Results . . . 22
3.7.3 Discussion . . . 24

4 Beat Tracking **26**

4.1 What is Beat Tracking? . . . 26
4.2 Existing Approaches . . . 27
4.2.1 Beat Period Estimation . . . 28
4.2.2 Phase Estimation . . . 30
4.3 A New Approach to Beat Tracking for Live Drums . . . 31
4.3.1 Multi-Signal, Multi-Scale Autocorrelation . . . 32
4.3.2 Base Period Model . . . 33
4.3.3 Multi-Hypothesis Pulse Tracking . . . 38
4.3.4 Meter and Tatum Inference . . . 42
4.4 Evaluation . . . 43
4.4.1 Test Set . . . 43
4.4.2 Beat Tracking Evaluation Method . . . 43
4.4.3 Results . . . 44

5 Analyzing Drum Patterns 50
5.1 Introduction . . . 50
5.2 Previous Work on Downbeat Detection . . . 51
5.3 Deep Learning . . . 52
5.3.1 The Restricted Boltzmann Machine . . . 52
5.3.2 Stacking RBMs . . . 54
5.3.3 The Conditional Restricted Boltzmann Machine . . . 55
5.4 Modeling and Analyzing Drum Patterns . . . 56
5.4.1 Bounded Linear Units . . . 56
5.4.2 Label Units . . . 57
5.4.3 Modeling Drum Patterns . . . 58
5.5 Training the System . . . 59
5.5.1 Training Data . . . 59
5.5.2 Network Configurations . . . 59
5.5.3 Network Training . . . 60
5.5.4 HMM Filtering . . . 63
5.6 Downbeat Detection Results . . . 63
5.6.1 Classifying Subdivisions . . . 63
5.6.2 Using HMM Filtering . . . 66
5.7 Discussion of Results . . . 67

6 Discussion 71
6.1 Practical Considerations . . . 72
6.1.1 Computation . . . 72
6.1.2 Scheduling . . . 73
6.2 Contributions . . . 74
6.3 Future Directions . . . 75

Bibliography **76**

Acknowledgments

First I would like to thank my advisors David Wessel and Nelson Morgan for their guidance and help throughout this long process of getting a PhD. David, your many ideas inspired everything I worked on, and I'm grateful that I had the opportunity to work on music applications with you. Morgan, I would've never finished if it hadn't been for your advice on research organization and goal setting. Thank you to Ian Saxton for lending his time to perform all of the recorded drum tracks used in chapters 2 and 3. Lastly, thank you to my parents for always being there for me through the ups and downs and for instilling within me a love of music and mathematics.

Chapter 1

Introduction

The goal of this thesis is to provide new techniques that enhance the automated analysis of rhythm in music. In particular, the focus is on creating techniques that would be useful in the real-time rhythmic analysis of a live drum performance, but the approaches covered can easily be applied to other applications in machine listening, such as polyphonic beat tracking, source separation, or the analysis of audio sequences. There are three primary contributions of this thesis, each of which represents a piece in a complete drum understanding system. Figure 1.1 shows the three primary components: drum detection, multi-hypothesis beat tracking, and drum pattern analysis. In addition to these three, a fourth component, onset detection, serves as a pre-processing step for both the drum detection and beat tracking components. Each of these is designed with real-time operation in mind, so a purely causal approach is taken with an attention to throughput latency; however, these components can easily be modified for offline operation and their accuracy will surely be improved with the addition of non-causal information.

1.1 Applications

The aim of the proposed system is to enable percussionists and drummers to enhance their practice and performance experience. The basic idea is that a drummer plays a percussive rhythm, and in real-time, the system outputs information such as the tempo, beat locations, and rhythmic structure of the drum beat. These outputs can be used by separate systems in order to generate appropriate musical accompaniment, synchronize backing tracks or lighting effects, or tightly synchronize aspects of a computer music performance by a separate musician.

Ideally, such functionality will allow a drummer to greatly expand his or her musical contribution. Since the system will track the rhythmic beats of what

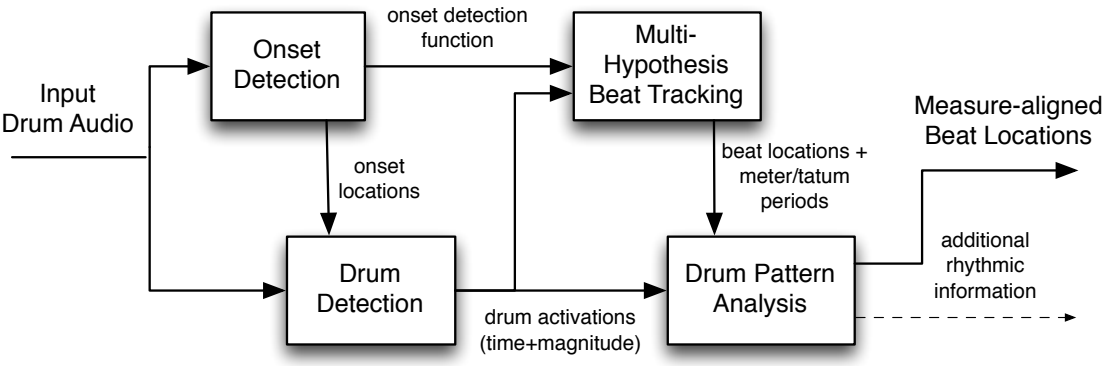

Figure 1.1: *The components of the drum understanding system. These four components are covered in the next four chapters.*

is being played, a synthesized bass line, chord harmony, or pre-arranged accompaniment could be synchronized with the drumming. The backing tracks that drummers commonly have to synchronize with during a live performance could instead be synchronized with the drumming. Instead of playing to a click track, the drummer creates the click track by playing a rhythm. This functionality could also be used to enhance practice sessions or music education by providing timing or accuracy feedback or allowing a virtual band to play along with the drumming. The key idea is that instead of the drummer having to play along with the music, the system would enable a computer to play along with the drummer.

1.2 Multimedia Content Analysis

Multimedia content analysis is becoming an increasingly important field given the exponential growth of audio and video databases. Such techniques are important for efficient and effective content-based search, recommendation, navigation, and tagging. In the audio realm, the development of content analysis techniques has been grouped into fields with names such as *machine listening* or *machine hearing*. Speech processing is probably the most mature sub-field within machine listening, but the growing body of *music information retrieval* (MIR) research points to the expanding need for other types of audio analysis. Within MIR, automated rhythm analysis techniques augment spectral qualities, such as pitch, harmony, and timbre, by adding information about temporal qualities, such as beat, tempo, meter, and groove. Together, these spectral and rhythmic qualities form a more complete picture of a piece of music as a whole; in music, you can't have one without the other.

There are an abundance of MIR techniques that are designed to operate on databases of recorded music; however, the real-time analysis of live content has seen much less work. Despite this, my work on live drum analysis does not stand alone. Robertson [69] and Collins [16] have both focused on the practicalities of implementing, using, and evaluating real-time drum analysis systems in the context of interactive performance. Their work demonstrates the fact that rhythm analysis techniques can in fact be applied in real-time with good results. What is missing in much of the work on rhythm analysis is the application of machine learning techniques to replace or augment the heuristically designed rules that dominate existing tasks such as beat tracking and meter analysis. My work on drum detection and drum pattern analysis is an attempt to bring more machine learning into rhythm analysis.

1.3 System Overview

The four components shown in Figure 1.1 (onset detection, drum detection, beat tracking, and drum pattern analysis) are covered in the next four chapters. The onset detector, which is covered in Chapter 2, locates rhythmic events (drum hits) in time and communicates these locations to the drum detector. Also, the beat tracker makes use of the intermediate onset detection function, which contains the onset likelihood at each point in time. Chapter 3 covers the drum detection component, which decomposes each onset onto a set of learned drum templates in order to quantify how much each drum contributed to an onset. This drum-wise onset information is used by both the beat tracker and the pattern analysis component. The beat tracking system (covered in Chapter 4) uses the onset detection function and drum onset locations to output the locations of rhythmic beats along with additional metrical information, such as the meter or beat subdivisions. Lastly, Chapter 5 discusses my approach to drum pattern analysis. This component analyzes the output of the beat tracker and drum detector in order to make higher-level rhythmic inferences such as beat-measure alignment, perceived tempo, or rhythmic style information.

Chapter 2

Onset Detection

2.1 What is an Onset?

Music is a temporal art form. Each sound, pitch, tone, note, chord, or noise that makes up a piece of music occurs at a meaningful point in time. Even the sound pressure waves that embody each musical event are dependent on the continued progression of time. In the analysis of rhythm, *when* things occur is of primary importance, and a simple way to measure when things occur is to detect the beginning, or onset, of each event.

In [5], Bello makes distinctions between the attack, transient, and onset of a note. If a *transient* is a short interval of time in which the musical signal evolves in a relatively unpredictable way, then the *attack* is the time interval during which the amplitude envelope is increasing at the beginning of the transient. From these definitions, the onset is a location in time chosen to represent the beginning of the attack, or as Bello says, the earliest time at which the transient can be reliably detected.

In the case of percussive instruments such as drums, the attacks are typically very well localized, so onsets are much easier to detect compared to some other classes of instruments, such as bowed stringed instruments or woodwinds. Because of this, there wasn't a great need for innovation in my approach to onset detection. Nevertheless, accurate results are always desired, so I outline the approach I used in this short chapter.

2.2 Common Approaches to Onset Detection

The most common methods used to detect onsets are covered by Bello in [5]. His tutorial breaks down onset detection into three steps: pre-processing, reduction, and peak picking.

First is a pre-processing step which computes some type of transformation on the input audio signal in order to improve the ability of subsequent steps to detect onsets. This step typically involves a multiple frequency band decomposition or a transient/steady-state decomposition. The multi-band decomposition aids robustness by allowing each sub-band to be analyzed separately for transients. The transient/steady-state decomposition accomplishes the same by isolating the transient parts of the signal.

The second common step can be referred to as the "reduction" stage, in which the pre-processed input signal is reduced to a simple *onset detection function* (ODF) from which onsets can be located much more easily. The simplest reduction involves computing the envelope of the input audio.

$$\text{Env}(n) = \frac{1}{N} \sum_{m=-N/2}^{N/2-1} |x(n+m)|w(m) \tag{2.1}$$

Where $x(n)$ is the audio signal, and $w(n)$ is a window function of length N.

From such a detection function, the local maxima can be located, and the hope is that these maxima correspond to onsets. However, the above simple envelope follower is not a useful onset detection function because envelope peaks do not tend to correspond to perceptual onsets. An improvement would be to take the first order difference (derivative) of the envelope to detect the locations where the envelope is most strongly increasing. Typically the output from this discrete derivative is half-wave rectified, retaining only the positive changes in amplitude while rounding the negative changes up to zero.

The Weber-Fechner law states that the perceptual just-noticeable-difference between stimuli is proportional to the magnitude of the stimuli. Klapuri uses this fact in a compressed multi-band energy detection function [47]. This approach uses a multi-band decomposition in its pre-processing step followed by a compressive logarithmic differentiation of the energy in each band which simulates the perceptual effect stated in the Weber-Fechner law because:

$$\frac{d \log\left(E(t)\right)}{dt} = \frac{dE(t)}{dt} \frac{1}{E(t)} \tag{2.2}$$

Where $E(t)$ is the energy in a band at time t.

After this transformation, the band-wise signals are summed into a single detection function. My method uses an onset detection function similar to this.

Other approaches to computing the ODF include a spectral novelty measure that measures the difference between the current complex spectrum and a prediction of the current spectrum conditioned on the past [6]. This approach has been shown to work better for strongly pitched instruments that lack percussive onsets. A more recent and novel approach uses a temporal Recurrent Neural Network (RNN) which is trained to output the probability that an onset is occurring

at the current frame [25][13]. The RNN is fed with local spectrum data and first order spectrum differences in order to make its predictions. This RNN-based approach is one of the few approaches to onset detection that uses machine learning to compute the ODF, and it achieves state-of-the-art performance.

After computing the onset detection function, a *peak picking* procedure is used to identify onset locations. This procedure could be as simple as identifying all local maxima in the ODF above a constant threshold; or, more complex processing and thresholding of the ODF could be used. Many of the techniques that compute reliable peak thresholds operate in a non-causal manner, which precludes their use in real-time systems. These thresholds are typically computed as a function of statistics describing a local window of the ODF, such as a percentage of the mean, median, or maximum. Some methods use a heuristic rule that limits the temporal proximity of detect peaks. The application of these common peak picking methods to real-time onset detection is examined in [12].

2.3 Real-Time Onset Detection for Drums

An overview of my approach to computing an onset detection function is shown in Figure 2.1. My approach is similar to that presented by Klapuri in [47], in that I compute the ODF using the differentiated log-energy of multiple sub-bands. The reason for analyzing the signal in multiple bands is to allow for the robust detection of onsets even in instances when much of the spectrum is still filled with energy from a previous onset. Analyzing the logarithmic energy landscape in each band individually can help detect less energetic onsets that the human ear is capable of detecting.

First, I compute Fast Fourier Transform (FFT) on each frame of audio using a window size of 23ms and a hop size of 5.8ms (1024 and 256 samples, respectively, at 44.1kHz). The energy in each band is computed by summing the energy in overlapping triangularly shaped frequency windows. Compared to Klapuri's method, I use a larger number of sub-bands (20 per channel) spaced according to the Bark scale. The energy in the ith sub-band of the nth frame of samples, $s_i(n)$, is processed using mu-law compression as a robust estimation of pure logarithmic compression, according to eq. (2.3).

$$c_i(n) = \frac{\log(1 + \mu s_i(n))}{\log(1 + \mu)} \tag{2.3}$$

I use a large value of $\mu = 10^8$ in the above in order to enhance the detection of more subtle drum notes. In order to limit the detection of spurious, non-musical onsets, the compressed band energies, $c_i(n)$, are smoothed using a linear phase Hann (a.k.a. Hanning) window [59] with a 3dB cutoff of 20Hz to produce $z_i(n)$.

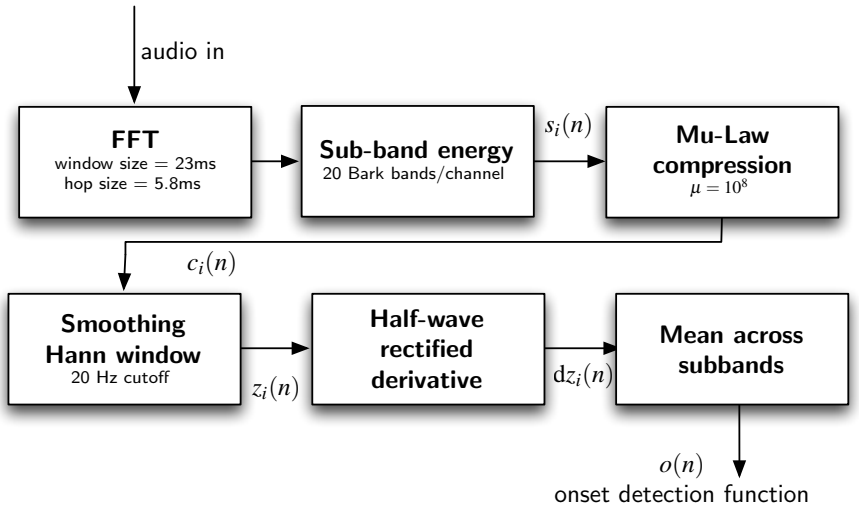

Figure 2.1: *Diagram of the onset detection system. The audio sample rate is 44.1kHz*

Then $z_i(n)$ is differentiated and half-wave rectified to get $dz_i(n)$. To arrive at the final onset detection function, $o(n)$, I take the mean of $dz_i(n)$ across sub-bands.

To pick onset locations from the detection function, I first find the local maxima of $o(n)$. Then, local maxima are labeled as onsets if they are larger than a dynamic threshold, T_{dyn} [eqs. (2.4–2.6)]. In addition, the detection function must have dropped below the local dynamic threshold since the last detected onset. These criteria were chosen with causality requirements in mind.

I have experimented with many types of dynamic thresholds. Typical ways to compute a dynamic threshold are covered in [5]. The primary dynamic threshold I utilize is shown in eq. (2.4), where $P_i(n)$ represents the ith percentile of the set

$$\mathbf{O}(n) = \{o(n), o(n-1), \ldots, o(n-N)\},$$

where N is chosen so that $\mathbf{O}(n)$ contains the most recent 1 second of the detection function.

$$T_1(n) = 1.5(P_{75}(n) - P_{25}(n)) + P_{50}(n) + 0.05 \tag{2.4}$$

However, due to causality requirements, we also need a way to quickly increase the threshold after a passage of silence. This is accomplished using a percentage of the maximum value, $P_{100}(n)$, of $\mathbf{O}(n)$ as shown in eq. (2.5).

$$T_2(n) = 0.1P_{100}(n) \tag{2.5}$$

The final dynamic threshold is calculated as the power mean between $T_1(n)$ and $T_2(n)$ [eq. (2.6)]. The power mean is chosen to simulate a softened maximum

between the two values.

$$T_{dyn}(n) = \left(\frac{T_1(n)^p + T_2(n)^p}{2} \right)^{1/p} \tag{2.6}$$

I use a value of $p = 2$, which results in a fairly soft maximum. As $p \to \infty$, the power mean approaches the true maximum.

2.4 Onset Detection Accuracy

In the overall drum understanding system, the onset detector determines which frames are analyzed for the presence of individual drums (as described in Chapter 3), and it produces the input which drives beat tracking (as described in Chapter 4); therefore, its accuracy is of the utmost importance. The accuracy of my approach to onset detection is evaluated using a dataset containing recorded audio of drum set performances. The audio was recorded using multiple microphones and mixed down to two channels of audio, as is common during live performances. Audio samples were recorded at a 44.1kHz sampling rate with 24 bits of resolution. Overall 23 minutes of audio were recorded consisting of 8022 annotated onsets. The tempo of the performed rhythms varies between 94 and 181 beats per minute (BPM). The shortest annotated interval between onsets is ∼83ms (16th notes at 181 BPM), though this time interval is infrequent. The shortest commonly occurring interval is 16th notes at 100 BPM, which is equivalent to 150ms between onsets.

I evaluated onset detection accuracy using precision and recall rates:

$$\text{Precision} = \left(\frac{\#\text{ correctly detected onsets}}{\#\text{ detected onsets}} \right) \tag{2.7}$$

$$\text{Recall} = \left(\frac{\#\text{ correctly detected onsets}}{\#\text{ actual onsets}} \right) \tag{2.8}$$

To count correctly detected onsets, I iterate through the list of annotated onsets and attempt to match them with the detected onsets. If the difference between a detected onset time and an actual onset time was within 10 frames (roughly 58ms), it was counted as correct. The onset detection results are shown in Figure 2.2.

Precision, recall, and F-score are plotted against a linear adjustment to the dynamic threshold, T_{dyn}. The F-score is defined as the harmonic mean of precision and recall.

$$F_1 = 2 \times \frac{\text{precision} \times \text{recall}}{\text{precision} + \text{recall}} \tag{2.9}$$

The threshold adjustment (+0.028) corresponding to the maximum F-score (0.948) is labeled in the plot. As is the case in most detection problems, the threshold

adjustment controls a trade-off between precision and recall, so the optimal adjustment depends on the particular application. At the original threshold (+0.0), the F-score is 0.941 with a recall of 0.944 and a precision of 0.939. These values are deemed to be quite satisfactory, so I use this original threshold to locate onsets for the drum detection application presented in the next chapter.

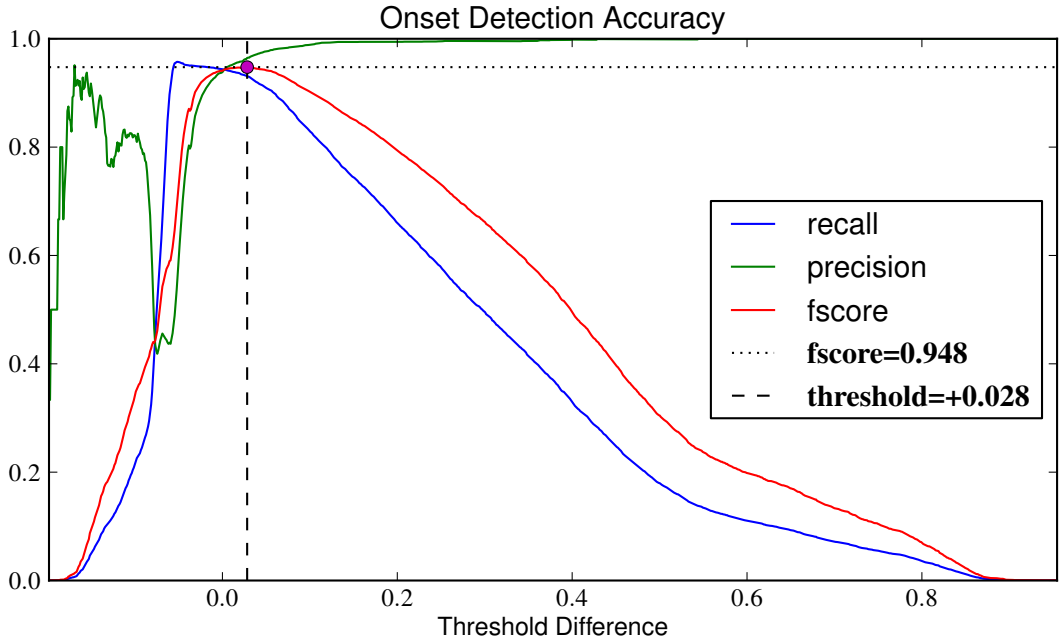

Figure 2.2: *Precision, recall, and F-score for onset detection with varying threshold. The maximum F-score and its corresponding threshold adjustment are denoted by a pink dot and dotted lines. This threshold adjustment (+0.028) gives a recall of 0.931 and a precision of 0.965.*

Chapter 3

Drum Detection

3.1 Introduction

There is much more to rhythm analysis than the locations of onsets. In the analysis of pitched instruments, each onset may correspond to a different note or chord, so that a series of onsets could describe a specific melody or harmonic progression that has a completely different musical meaning than a different series of notes with the same relative locations in time. When analyzing a series of drum onsets, *which* drum is being struck and *how loudly* determines the feel of the rhythm and helps to define a rhythmic pattern. My motivation for developing live drum detection techniques stems from the fact that the activation (onset time and amplitude) of individual drum events is much more informative than non-drum-specific onset locations when determining perceptual beat locations and classifying rhythmic patterns. For example, in rock and jazz music, the drummer typically keeps time on one of the cymbals by playing some sort of repeating pattern (e.g., straight eighth or quarter notes or a swung ride pattern), while the snare and bass drum will typically accentuate certain beats within a measure.

3.2 Approaches to Drum Detection

Early approaches to drum detection focus on classifying single isolated drum sounds [40][41] or using spectral features to classify each drum onset [34][30]. The approaches used by Paulus [61] and FitzGerald [28] use source separation techniques to separate drum audio into drum-wise source signals and then detect onsets from each signal belonging to each type of drum. The results presented by Paulus show very accurate transcription results for performances on three different drums.

More recent work on drum detection attempts to transcribe a drum performance from within a polyphonic multi-instrument audio track. This task is much

more difficult than drums-only transcription due to the presence of non-target instruments and the typically low volume level of drum audio compared to other instruments. Tanghe [71] uses spectral features extracted at onset locations to classify which drum an onset came from, if any. Gillet [31] augments this featured-based classification approach by adding pre-processing steps that enhance the drum signal relative to other instruments. Yoshii [76] uses a novel adaptive approach to polyphonic drum transcription by starting with general drum templates, matching the templates to reliable drum onsets, and then refining the templates to better match the specific drums used on the track.

There has also been work on isolating the drums from the rest of the instruments using source separation techniques, primarily Non-negative Matrix Factorization (NMF) [38][75][3]. Source separation is an important problem in the field of music information retrieval that attempts to isolate sound sources (instruments, notes, sound samples, noise, etc.) as separate signals [63]. This can greatly enhance music analysis tasks such as genre/artist classification, beat-tracking, lyrics synchronization, and automatic transcription.

My approach to drum detection uses techniques derived from NMF [52] to compute the contribution of individual drums to spectrogram slices that are extracted at onset locations. Each drum is modeled by one or more spectral templates onto which drum onsets can be decomposed. The decomposition provides drum-wise amplitude information that is missing in classification-based approaches. The next section gives an overview of this approach to drum detection.

3.3 System Overview

My drum detection system is comprised of four primary components: onset detection, spectrogram slice extraction, drum template training, and non-negative vector decomposition (NVD). Onset detection (covered in Chapter 2) is used to locate significant rhythmic events to be spectrally analyzed. During the training of the spectral template(s) of a particular drum, spectrogram slices at onset locations are clustered using a probabilistic formulation of the Itakura-Saito divergence [65]. The cluster centers are then used as time-frequency templates for each trained drum. Live drum detection is accomplished by locating drum events and decomposing the spectrogram slices of each event as a non-negative mixture of drum templates. A block diagram of the system is shown in Figure 3.1.

In a live drum detection system, we have the luxury of training templates for a single target drum kit. It is assumed that sound samples of the individual drums can be gathered during sound check or sometime before a performance, so we can specialize the spectral templates for the specific drums that will be used.

My approach differs from previous approaches in that it learns multiple templates per drum to account for the vast array of sound qualities that can be

produced by a particular drum. For example, a snare drum can produce a large variety of sounds depending upon the striking velocity and whether the drum stick hits the center, edge, or rim of the drum.

In addition, my system incorporates the use of "tail" templates to address the problem that occurs when long-decay percussion instruments (e.g., cymbals) overlap with successive drum onsets, thereby degrading detection accuracy.

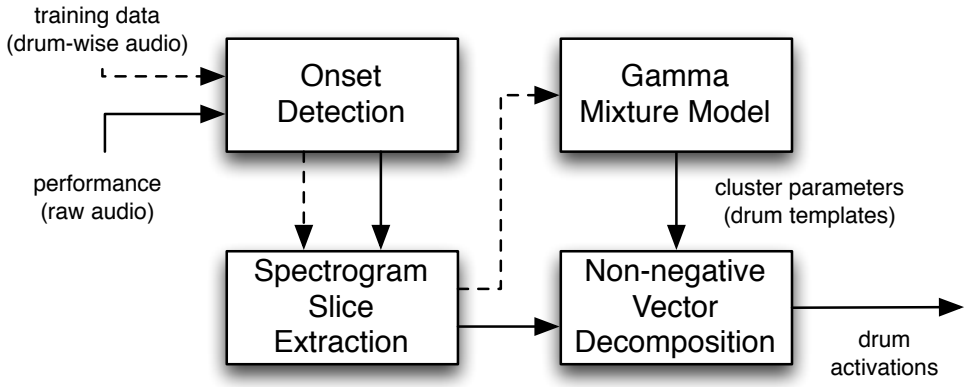

Figure 3.1: *System overview. The dotted connections are present during training, while the solid connections are used during live performance*

In the following three sections, I describe the approach used for spectrogram slice extraction, drum template training, and non-negative vector decomposition, respectively.

3.4 Extraction of Spectrogram Slices

The primary feature I use to perform drum detection is a spectrogram slice that is local to a detected onset event. In previous work that applies non-negative matrix factorization to drum transcription, the features used in drum detection are simply the local spectra which contain no time-based transient information [61]. Like in Yoshii's template-based approach to drum transcription [76], I use a number of adjacent spectral frames to detect the presence of drums. This allows us to capture characteristics of the attack and decay of a particular drum.

The short-time spectra that make up a spectrogram slice are extracted using the same 23ms Hann window and 5.8ms hop size as is used for onset detection. The energy spectrum is then dimensionality-reduced by summing the energy into 80 sub-bands per channel spaced according to the Bark scale. In [3], I have shown that NMF used for drum source separation can be sped up significantly using this sub-band-based, dimensionality-reduction approach with no loss of separation quality.

A spectrogram slice is comprised of 100ms worth of adjacent spectral frames (about 17 frames). The window of frames used in the slice begins 33ms before the detected onset and ends 67ms after the onset as shown in Figure 3.2. This offset helps to ensure that the window contains the entire attack of the onset. In addition, during training, I extract a further 100ms of spectrum frames, after the initial slice, to be used in computing "tail" templates. When performing non-negative vector decomposition (NVD) on input spectrogram slices, these tail templates serve as "decoys" to prevent the long decay of a previous drum onset (e.g., cymbal crash) from incorrectly showing up in the drum activations of the current onset. These tail templates account for the intrusion of these long decays into subsequent spectrogram slices, and the activations of tail templates are ignored in the final drum detection output.

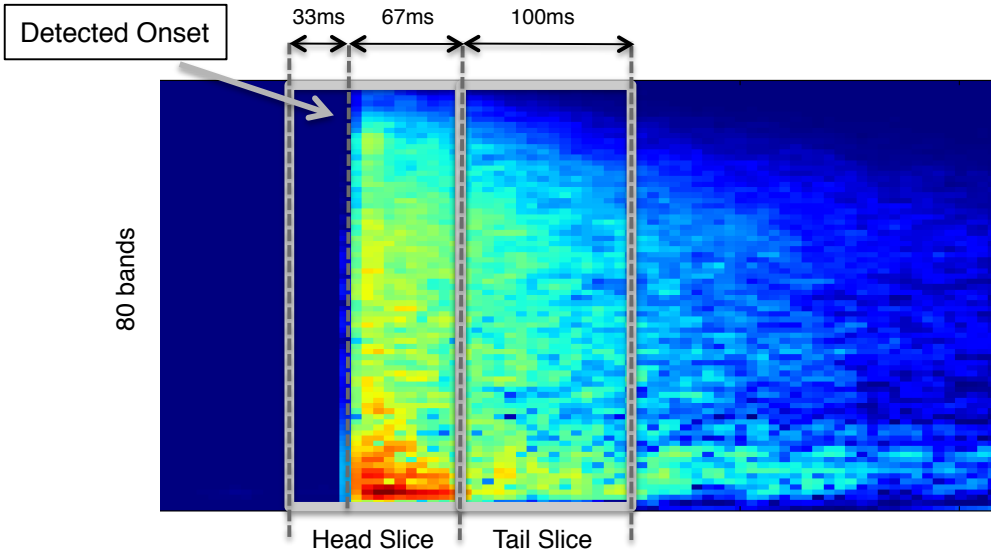

Figure 3.2: *Spectrogram slice alignment. A head slice is 100ms wide and starts 33ms before the corresponding detected onset. The tail slice starts 100ms after the beginning of the head slice.*

The head and tail spectrogram slices are both smoothed across frames using a 29ms Hann window and downsampled by a factor of 2 in order to allow the slices to be more robust to variations in microtiming. Finally, the square root is taken to move the data from the energy domain to the magnitude domain. During training, these head and tail slices are fed to the drum template training stage covered in the following section. During live performance, only the head slices are used as input to the template-based decomposition covered in Section 3.6.

3.5 Training drum templates

To model the time-frequency characteristics of individual drums given a target drum set and specific microphone placements, I cluster the spectrogram slices extracted from the training data for each drum. It is assumed that we have access to isolated, single-drum samples for each drum as is typically the case during a pre-performance sound check. Once I have accumulated a training set of spectrogram slices (both head and tail for each detected onset) for a particular drum, I follow the agglomerative clustering procedure outlined in this section to arrive at a small number of spectrogram templates that compactly describe the range of sounds that can be produced by the drum. The head and tail slices for each drum are clustered separately to produce separate sets of templates.

3.5.1 Clustering with the Itakura-Saito Divergence

The speech recognition community has made frequent use of Gaussian Mixture Models (GMMs) for modeling speaker-specific data [67][66]. Clustering data using GMMs enforces a squared Euclidean distance measure when determining the cluster membership probabilities of individual observations. As shown by Banerjee [2], other members of the Bregman divergence family can be used for clustering by using mixture models built from other exponential-family priors. An important Bregman divergence to the speech and music community is the Itakura-Saito divergence (IS divergence), which is frequently used as a measure of the perceptual distance between audio spectra [65][26]. In order to perform soft clustering using the IS divergence we can use a mixture model composed of exponential distributions [2], or more generally, gamma distributions.

The fact that a mixture model composed of gamma distributions allows us to cluster using the perceptual IS divergence is the primary motivation behind this approach to clustering. In addition, the clustering procedure covered in this section avoids the covariance matrix updates present in GMM training. When using a large training set or high dimensional data, GMM covariance updates can greatly increase computational costs. For small training sets (e.g., when clustering a handful of drum hits), even variance-only diagonal covariance matrix computations can become unstable, inaccurate, or poorly defined in clusters with few members.

3.5.2 The Gamma Distribution

The probability density function of the univariate gamma distribution can be written as follows:

$$\begin{aligned} \mathrm{p}(y|\lambda,\eta) &= y^{\eta-1}\frac{\lambda^{\eta}e^{-\lambda y}}{\Gamma(\eta)}, \quad y \geq 0;\ \lambda,\eta > 0 && (3.1)\\ E[y] &= \mu \qquad\qquad = \eta/\lambda && (3.2)\\ \mathrm{Var}(y) &= \mu/\eta \qquad\qquad = \eta/\lambda^2 && (3.3) \end{aligned}$$

Where the mean is shown in eq. (3.2), and $\Gamma(\eta)$ is the gamma function, which is a generalization of the factorial function to real numbers. The gamma distribution generalizes the Erlang distribution (which models the sum of η independent identically distributed exponential random variables) to continuous $\eta > 0$. Figure 3.3 shows the gamma distribution with constant mean for various values of shape parameter η. We can see that as η is increased, the variance relative to the mean is decreased, so adjusting η is a way to control the spread of each cluster. However, unlike with a Gaussian distribution, the variance of the gamma distribution increases as the mean increases (see eq (3.3)), which is a desirable trait given the logarithmic amplitude sensitivity of the human auditory system.

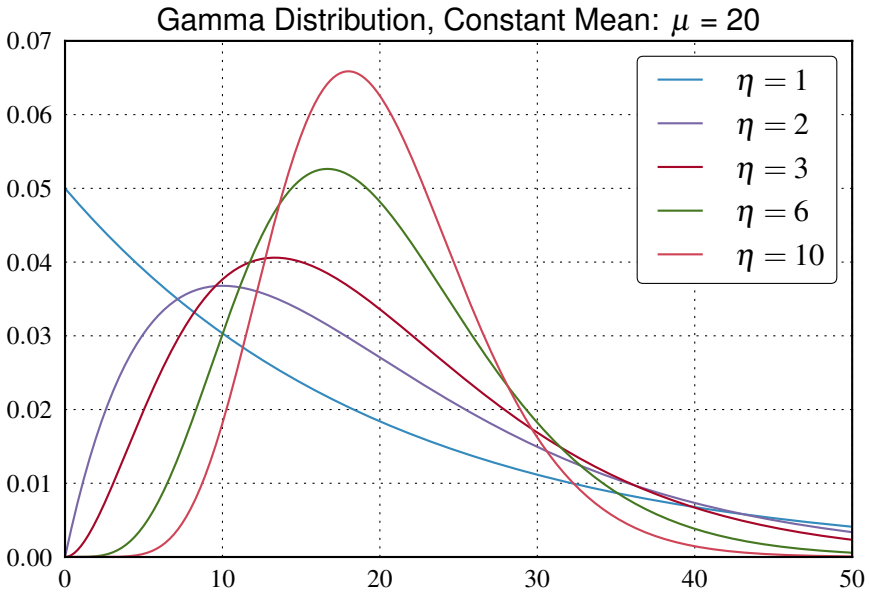

Figure 3.3: *The gamma distribution for various values of η and constant mean.*

As shown in [2], we can write the gamma distribution in its Bregman divergence form [eq. (3.4)]. The Bregman divergence inside of the exponential, $d_{\mathrm{IS}}(y,\mu)$, is

the IS divergence.

$$\begin{aligned} \mathrm{p}(y|\lambda,\eta) &= \exp\left(-\eta\, d_{\mathrm{IS}}(y,\mu)\right) b_0(y) && (3.4)\\ d_{\mathrm{IS}}(y,\mu) &= \tfrac{y}{\mu} - \log\tfrac{y}{\mu} - 1 && (3.5)\\ b_0(y) &= \frac{e^{-\eta}\eta^k y^{-1}}{\Gamma(\eta)}, \quad \mu = \eta/\lambda && (3.6) \end{aligned}$$

To model multidimensional data, I construct a multivariate gamma distribution from independent gamma distributions:

$$\begin{aligned} \mathrm{p}(\mathbf{y}|\boldsymbol{\lambda},\eta) &= \prod_{i=1}^{M} \frac{\lambda_i^\eta y_i^{\eta-1} e^{-\lambda_i y_i}}{\Gamma(\eta)} && (3.7)\\ &= \frac{|\boldsymbol{\lambda}|^\eta |\mathbf{y}|^{\eta-1} \exp(-\boldsymbol{\lambda}\cdot\mathbf{y})}{(\Gamma(\eta))^M} && (3.8)\\ E[\mathbf{y}] = \boldsymbol{\mu} &= \eta/\boldsymbol{\lambda} && (3.9) \end{aligned}$$

where $|\cdot|$ denotes the product of the elements in a vector, $\mathbf{a}\cdot\mathbf{b}$ denotes a dot product between vectors, and division involving vectors is performed element-wise.

The log-likelihood can be written:

$$\log\big(\mathrm{p}(\mathbf{y}|\boldsymbol{\lambda},\eta)\big) = \eta\log|\boldsymbol{\lambda}| + (\eta-1)\log|\mathbf{y}| - \boldsymbol{\lambda}\cdot\mathbf{y} - M\log\big(\Gamma(\eta)\big)$$

3.5.3 The Gamma Mixture Model

The Gamma Mixture Model (ΓMM) has the following distribution for observations, $\mathbf{y}_n$:

$$\begin{aligned} \mathrm{p}(\mathbf{y}_n|\theta) &= \sum_{l=1}^{K} \pi_l \mathrm{p}(\mathbf{y}_n|\boldsymbol{\lambda}_l,\eta) && (3.10)\\ \pi_l &= \mathrm{p}(x_n = l) && (3.11) \end{aligned}$$

where $\theta = \{\boldsymbol{\lambda}_l, \pi_l\}_{l=1}^K$, $\mathbf{y}_n \in \mathbb{R}^M$, and x_n is a hidden variable that denotes which mixture component $\mathbf{y}_n$ was generated from.

In order to learn the ΓMM parameters, θ, from a set of training data, $\mathbf{Y} = \{\mathbf{y}_n\}_{n=1}^N$, I employ the Expectation-Maximization (EM) algorithm [9]. Because maximizing the log-likelihood of the parameters, $\log \mathrm{p}(\mathbf{Y}|\theta)$, is intractable, instead the EM algorithm iteratively optimizes an auxiliary function:

$$Q(\theta|\theta^{(t)}) = E\left[\log \mathrm{p}(\mathbf{Y},\mathbf{X}|\theta)\middle|\mathbf{Y},\theta^{(t)}\right] \quad (3.12)$$

where $\theta^{(t)}$ are the current parameters at iteration t, θ are the parameters to be optimized during the current iteration, and $\mathbf{X} = \{x_n\}_{n=1}^N$. For this ΓMM, we

eventually arrive at the simplified expression:

$$Q(\theta|\theta^{(t)})= \tag{3.13}$$
$$N_l^* \sum_{l=1}^{K} \left\{ \log \pi_l + \eta \log |\boldsymbol{\lambda}_l| - \eta \left(\boldsymbol{\lambda}_l \cdot \tfrac{1}{\boldsymbol{\lambda}_l^*} \right) \right\}$$
$$+(\eta - 1) \sum_{n=1}^{N} \log |\mathbf{y}_n| - NM \log \Gamma(\eta)$$

where

$$N_l^* = \sum_{n=1}^{N} \mathrm{p}(x_n = l|\mathbf{y}_n, \theta^{(t)}) \tag{3.14}$$

$$\boldsymbol{\lambda}_l^* = \frac{\eta N_l^*}{\sum_{n=1}^{N} \mathbf{y}_n \mathrm{p}(x_n = l|\mathbf{y}_n, \theta^{(t)})} \tag{3.15}$$

Eqs. (3.14,3.15) rely on the posterior $\mathrm{p}(x_n = l|\mathbf{y}_n, \theta^{(t)})$ which can be calculated using Bayes' rule:

$$\mathrm{p}(x_n = l|\mathbf{y}_n, \theta^{(t)}) = \frac{\mathrm{p}(\mathbf{y}_n|x_n = l, \theta^{(t)})\mathrm{p}(x_n = l|\theta^{(t)})}{\mathrm{p}(\mathbf{y}_n|\theta^{(t)})} \tag{3.16}$$

$$= \frac{\pi_l \exp\left(-\eta\, d_{\mathrm{IS}}(\mathbf{y}_n, \boldsymbol{\mu}_l)\right)}{\sum_{j=1}^{K} \pi_j \exp\left(-\eta\, d_{\mathrm{IS}}(\mathbf{y}_n, \boldsymbol{\mu}_j)\right)} \tag{3.17}$$

$$= \frac{\pi_l |\boldsymbol{\lambda}_l|^{\eta} \exp(-\boldsymbol{\lambda}_l \cdot \mathbf{y}_n)}{\sum_{j=1}^{K} \pi_j |\boldsymbol{\lambda}_j|^{\eta} \exp(-\boldsymbol{\lambda}_j \cdot \mathbf{y}_n)} \tag{3.18}$$

where $\boldsymbol{\mu}_l = \eta/\boldsymbol{\lambda}_l$. Notice that in eq. (3.17), the posterior (or cluster membership probabilities) can be viewed as a weighted sum of the IS divergence (rescaled by the exponential function) between an observation and the cluster mean. Banerjee shows that soft clustering using exponential family distributions aims to minimize the expected value of the associated Bregman divergence between the cluster means and the cluster members [2]. For the gamma distribution, the associated Bregman divergence is in fact the Itakura-Saito divergence.

Now we can compute updated optimal values of the parameters using $Q(\theta|\theta^{(t)})$, which is maximized with the following parameter values:

$$\pi_l = \frac{N_l^*}{N}, \quad \boldsymbol{\lambda}_l = \boldsymbol{\lambda}_l^* \tag{3.19}$$

with N_l^* and $\boldsymbol{\lambda}_l^*$ from eqs. (3.14),(3.15).

We can continue to alternate between updating the posterior using eq. (3.18) and updating the parameters using eqs. (3.14),(3.15),(3.19) until $Q(\theta|\theta^{(t)})$ converges.

3.5.4 Agglomerative Clustering with Gamma Mixture Models

The ΓMM training procedure covered in 3.5.3 relies on a fixed number (K) of mixture components or clusters. In order to choose a value of K that complements the training data, I use the agglomerative clustering approach in [14]. This approach starts with an initial maximum value of $K = K_0$ and iteratively merges similar clusters until we are left with a single cluster ($K = 1$).

The agglomerative procedure begins by initializing the K_0 cluster means, $\boldsymbol{\mu}_l = \eta/\boldsymbol{\lambda}_l$, to be equal to randomly chosen data points from the training set, $\mathbf{Y}$. The initial cluster prior probabilities, π_l, are uniformly initialized.

Then the parameters are iteratively optimized for this value of K until convergence. Upon convergence, we merge the two most similar clusters, decrement K, and again update the parameters until convergence. The similarity of two clusters (l, m) is computed using the following distance function:

$$D(l,m) = Q(\theta^*|\theta^{(t)}) - Q(\theta^*_{(l,m)}|\theta^{(t)}) \tag{3.20}$$

where θ^* is the set of parameters that optimizes $Q(\theta|\theta^{(t)})$, and $\theta^*_{(l,m)}$ is the optimal set of parameters with the restriction that:

$$\boldsymbol{\mu}^*_l = \boldsymbol{\mu}^*_m = \frac{\pi_l\boldsymbol{\mu}_l + \pi_m\boldsymbol{\mu}_m}{\pi_l + \pi_m} \tag{3.21}$$

Using eqs. (3.9),(3.13),(3.14),(3.15),(3.21) along with the convergence assumption of $\theta^* = \theta^{(t)}$, eq. (3.20) simplifies to:

$$\begin{aligned} &D(l,m) = \\ &\eta\left\{(N^*_l + N^*_m)\left[\log\left|\tfrac{N^*_l}{\boldsymbol{\lambda}_l} + \tfrac{N^*_m}{\boldsymbol{\lambda}_m}\right| - M\log(N^*_l + N^*_m)\right]\right\} \\ &\quad + N^*_l \log|\boldsymbol{\lambda}_l| + N^*_m \log|\boldsymbol{\lambda}_m| \end{aligned} \tag{3.22}$$

The two clusters that minimize $D(l, m)$ are deemed the most similar and are merged into a single cluster with parameters:

$$\boldsymbol{\mu}_{l,m} = \frac{\pi_l\boldsymbol{\mu}_l + \pi_m\boldsymbol{\mu}_m}{\pi_l + \pi_m}, \quad \pi_{l,m} = \pi_l + \pi_m \tag{3.23}$$

However, before merging, the parameter set, θ^*, for the current value of K is saved along with a measure of how efficiently the parameters describe the training data. As in [14], I use the Minimum Description Length (MDL) introduced by Rissanen [68].

$$\mathrm{MDL}(K,\theta) = \quad -\sum_{n=1}^{N}\log\left(\sum_{l=1}^{K}\mathrm{p}(\mathbf{y}_n|\boldsymbol{\lambda}_l)\pi_l\right) + \tfrac{1}{2}L\log(NM) \tag{3.24}$$

with L equal to the number of free parameters.

$$L = KM + (K-1) \tag{3.25}$$

For the ΓMM, eq. (3.24) simplifies to:

$$\begin{aligned}&\mathrm{MDL}(K,\theta) = \\ &-\sum_{n=1}^{N}\left[\log\left(\sum_{l=1}^{K}\pi_l|\boldsymbol{\lambda}_l|^{\eta}\exp\left(-\boldsymbol{\lambda}_l\cdot\mathbf{y}_n\right)\right)\right] \\ &-\sum_{n=1}^{N}\left[(\eta-1)\log|\mathbf{y}_n|\right] + NM\log\Gamma(\eta) + \tfrac{1}{2}L\log(NM)\end{aligned} \tag{3.26}$$

So for each K, I run the EM algorithm until $Q(\theta|\theta^{(t)})$ converges, then save the parameters, θ^*, along with $\mathrm{MDL}(K,\theta^*)$. I merge the two most similar clusters using eqs. (3.22) and (3.23), and then repeat the EM algorithm for the new smaller K. Once we reach $K = 1$, we choose the K and corresponding set of parameters with the minimum MDL.

We then compute the mean vector of each cluster, $\mu_\mathbf{l}$, from the winning cluster parameters, $\boldsymbol{\lambda}_l$, using eq. (3.9). These mean vectors are used as the drum templates.

3.6 Decomposing Drum Onsets

Now that we have trained a number of head and tail templates for each drum, we can compute the activations of these templates from live drum audio input. In order to make the amplitude of each template activation meaningful, the head templates are normalized so that all of the templates for a particular drum have the same approximate loudness. To do this, the head templates for a single drum are rescaled to have the same energy as the template with the maximum energy for that drum. No normalization is required for the tail templates, since as you will see, their activations are discarded.

3.6.1 Non-negative matrix factorization

Using the tail templates and normalized head templates, we can decompose live drum onsets as non-negative linear combinations of the templates. The activations of all the head templates corresponding to a single drum are summed to get the activation of that drum, and the activations of tail templates are discarded. To determine the activation of each template for a particular input spectrum, I employ a multiplicative algorithm used in non-negative matrix factorization

(NMF) [52]. NMF poses the problem:

$$\min_{W,H} D(X||WH), \quad W \geq 0, H \geq 0 \tag{3.27}$$

$$X \in \mathbb{R}^{M \times F},\ W \in \mathbb{R}^{M \times T},\ H \in \mathbb{R}^{T \times F} \tag{3.28}$$

where the matrix inequality constraint applies to every element, and X is the matrix to be factorized. $D(X||WH)$ is an associated cost function that measures the error between X and its approximate factorization WH. Typical NMF cost functions include the squared Euclidean distance, KL divergence, and IS divergence [26]. These cost functions all belong to the family of Bregman divergences and, additionally, belong to a subclass of the Bregman divergence called the *Beta divergence*[39], shown below.

$$d_\beta(x||y) = \begin{cases} \frac{x}{y} - \log\frac{x}{y} - 1, & \beta = 0 \\ x(\log\frac{x}{y} + y - x), & \beta = 1 \\ \frac{x^\beta + (\beta-1)y^\beta - \beta x y^{\beta-1}}{\beta(\beta-1)}, & \beta \in \mathbb{R}\backslash\{0,1\} \end{cases} \tag{3.29}$$

The Euclidean distance, KL divergence, and IS divergence are Beta divergences with $\beta = 2$, 1, and 0, respectively. In order to compute the Beta divergence between two matrices, I simply sum the Beta divergence between corresponding elements in the matrices.

3.6.2 Non-negative vector decomposition

The drum detection task does not carry out full NMF on the input spectrogram slices. Because we have already trained drum templates, the matrix W has already been determined, with the templates contained in its columns. Because W is not being optimized, the problem in (3.27) decouples into F independent optimization problems

$$\min_{\mathbf{h}_i} D(\mathbf{x}_i||W\mathbf{h}_i), \quad \mathbf{h}_i \geq 0 \tag{3.30}$$

$$\mathbf{x}_i \in \mathbb{R}^M,\ W \in \mathbb{R}^{M \times T},\ \mathbf{h}_i \in \mathbb{R}^T \tag{3.31}$$

where $\mathbf{x}_i$ and $\mathbf{h}_i$ are the ith columns of X and H, respectively.

To pursue an optimal solution to (3.30), I use multiplicative, gradient-based updates first introduced by Lee and Seung for Euclidean and KL divergence cost functions [52] and later generalized to Beta divergences [39][15].

$$\mathbf{h}_i \leftarrow \mathbf{h}_i . \frac{W^T((W\mathbf{h}_i)^{.\beta-2}.\mathbf{x_i})}{W^T(W\mathbf{h}_i)^{.\beta-1}} \tag{3.32}$$

Where dotted operations and division are carried out element-wise. In order to pursue an optimal mixture of templates in terms of the IS divergence, we can

iterate on the above update with $\beta = 0$; however, $d_\beta(x||y)$ is not convex in terms of y for $\beta = 0$, so there is the possibility of convergence to local minima. For $1 \leq \beta \leq 2$, $d_\beta(x||y)$ is convex in y. Bertin et al. [8] use this fact to design a "tempering" algorithm which begins running the updates with β in the convex range and then slowly reduces β to 0. This approach is shown to help avoid local minima and reduce the final converged cost. In my experiments, using $\beta = 0$ produced superior results compared to $\beta = \{1, 2\}$, and starting with $\beta = 2$ and slowly reducing it to 0 yielded slightly better results compared to using $\beta = 0$ during all updates. This is in contrast to my previous results which reported no benefit when using the tempering approach [4].

Adding an $L1$ penalty term to the updates also helped slightly by gently encouraging sparsity amongst the activation values of a single onset. Eq. (3.32) can be modified to include a scalar $L1$ penalty term, λ_{L1}, as follows.

$$\mathbf{h}_i \leftarrow \mathbf{h}_i . \frac{W^T((W\mathbf{h}_i)^{.\beta-2}.\mathbf{x_i})}{W^T(W\mathbf{h}_i)^{.\beta-1} + \lambda_{L1}} \tag{3.33}$$

In experiments, using $0.001 < \lambda_{L1} < 0.01$ slightly outperformed $\lambda_{L1} = 0$.

3.7 Drum Detection Evaluation

3.7.1 Test Setup

My original evaluation of this approach [4] used test data created using electronic drums along with a high-quality, multi-sampled drum sound library. This approach was used so that solid ground truth data could be easily obtained from the MIDI data recorded from the electronic drum kit. There were some critiques of this method of evaluation due to the fact that real drums can produce an infinite array of acoustic waveforms. Therefore, I took the effort to record and annotate a test set consisting of recordings of real acoustic drum performances. The test set is actually the same data used to evaluate the onset detection algorithm as described in Section 2.4. As mentioned in the aforementioned section, the 23 minutes of drum audio contains 8022 annotated drum onsets. There is a strong rock bias with many back-beat-heavy rhythms containing bass drum syncopation. Many cymbal pattern variations are recorded including open and closed hi-hats, ride cymbal bow and bell, and patterns that alternate between these options. There are a fair number of 16th note fills across snare and toms recorded at 100 BPM, and an accented multi-drum march-like rhythm containing many ghosted 16th notes at 94 BPM. A double time rock rhythm with open hi-hat is recorded at 181 BPM and contains a few 16th note intervals.

Each annotated onset is labeled with the drum it came from and one of three dynamic levels, representing quiet, medium, or loud onsets. This three level loudness labeling gives substantially less information than the 7 bits (128 levels) of MIDI velocity information captured by the electronic drums in my previous work, but it has the effect of at least capturing the difference between accented, unaccented notes, and ghost notes.

The drum audio was created using a total of 7 drums and cymbals: snare drum, bass drum, two toms, hi-hat, ride cymbal, and crash cymbal. For training, templates were trained separately for open and closed hi-hat variations, but the activations coming from these two sets of templates were combined in the end. This was done because annotating whether each hi-hat note came from an open or closed hi-hat can be difficult given the continuous amount of hi-hat openness that can be achieved using the hi-hat pedal.

For each of the 7 drum classes, head and tail templates are trained on notes played at a variety of dynamic levels and striking locations (center, rim, edge, bow, bell, etc.). The number of training onsets for each class is between 50 and 100.

Parameters that were varied in the training of templates include the number of initial head and tail templates, K_H and K_T, and the gamma shape parameter for the head and tail templates, η_H and η_T. Larger values of η tend to produce a larger number of clusters, and therefore, a larger number of templates per drum.

The parameters that were varied during the decomposition of onsets onto templates include the initial and final β values in the NVD updates, β_0 and β_f, and the $L1$ sparsity penalty, λ_{L1}. As reported in the previous section, the best NVD results were achieved using $\beta_0 = 2$, $\beta_f = 0$, and $0.001 < \lambda_{L1} < 0.01$.

I implemented all algorithms in Python[1] with the help of the Scipy[2] numerical library. This environment allowed rapid, productive prototyping of the algorithms, while retaining computational efficiency in the most demanding routines, such as FFT and matrix multiplication. Spectrogram slice extraction and NVD could easily be performed in real-time. The agglomerative training of multiple spectral templates took between 1 and 5 seconds per drum depending on the number of training strikes. Using optimized parallel implementations of these algorithms would likely allow training to be carried out virtually instantaneously due to the heavy use of matrix multiplication in the EM updates and cluster comparisons.

3.7.2 Results

To evaluate the system, I report both detection accuracy numbers and an amplitude fidelity measure. Detection accuracy was computed by attempting to match

[1] `www.python.org`
[2] `www.scipy.org`

each detected drum onset to an annotated drum onset. To be deemed a match, annotated and detected onsets must occur within 5 frames of each other (about 29ms). The detection threshold for the drum activation amplitudes computed using NVD was varied for each drum. The threshold producing the highest F-score (eq. (2.9)) for each drum was used when computing an overall F-score for all drums. Also, the ROC Area Under the Curve (AUC) is reported along with the Precision-Recall AUC (AUC-PR).

To compute the quality of the estimated activation amplitude for each drum, I use the cosine similarity, which is basically a normalized dot product between vectors.

$$S_{\cos}(\mathbf{x}, \mathbf{y}) = \frac{\mathbf{x} \cdot \mathbf{y}}{\|\mathbf{x}\| \|\mathbf{y}\|} \tag{3.34}$$

where $\| \cdot \|$ indicates the 2-norm.

The two vectors to be compared using the cosine similarity contain annotated and estimated drum amplitudes, respectively. Each amplitude vector contains the corresponding amplitudes from matching onset pairs; however, if an onset from either the annotated or estimated onset list has no matching counterpart, then a zero amplitude value is added at that location in the opposite amplitude vector.

Figure 3.4 shows the overall detection accuracy and amplitude quality while varying the initial number of head and tail templates, K_H and K_T. When $K_H = K_T = 30$, the initial number of templates is set to 30, but the agglomerative clustering procedure will most likely arrive at an optimal number of templates that is smaller than this. Setting $K_{\{H,T\}} = 1$ forces a single template for each drum, and $K_T = 0$ eliminates all tail templates. Larger values of the parameters η_H and η_T encourage a larger optimal number of templates. In this trial, $\eta_H = \eta_T = 1$. Other parameters used here are $\beta_0 = 2$, $\beta_f = 0$, and $\lambda_{L1} = 0$.

From Figure 3.4 it is apparent that using more than zero tail templates is clearly beneficial. This shows how important a role the tail templates play in preventing false positives caused by previous drum onsets with long decays. Using more than one head template per drum has a significant effect as well, and I attribute this to the fact that a drum is by no means a one-sound instrument. Using more than one tail template shows a very slight positive effect.

In Figure 3.5, the parameter η_H is varied, while $K_H = K_T = 30$ and $\eta_T = 1$. The NVD parameters are the same as in the previous trial. The results are ordered according to F-score, and it is clear that there is no clear winner across all metrics here. A large η_H encourages more templates per drum ($\eta_H = 7$ gives between 3–9 head templates per drum, while $\eta_H = 0.1$ yields 1–2 templates per drum), and there is definitely a trend favoring values of $\eta_H \geq 1$. The fact that the configurations with $\eta_H \in \{0.25, 0.1\}$ did worse in most cases suggests that at least one drum ended up with too few templates to adequately describe its range of sounds.

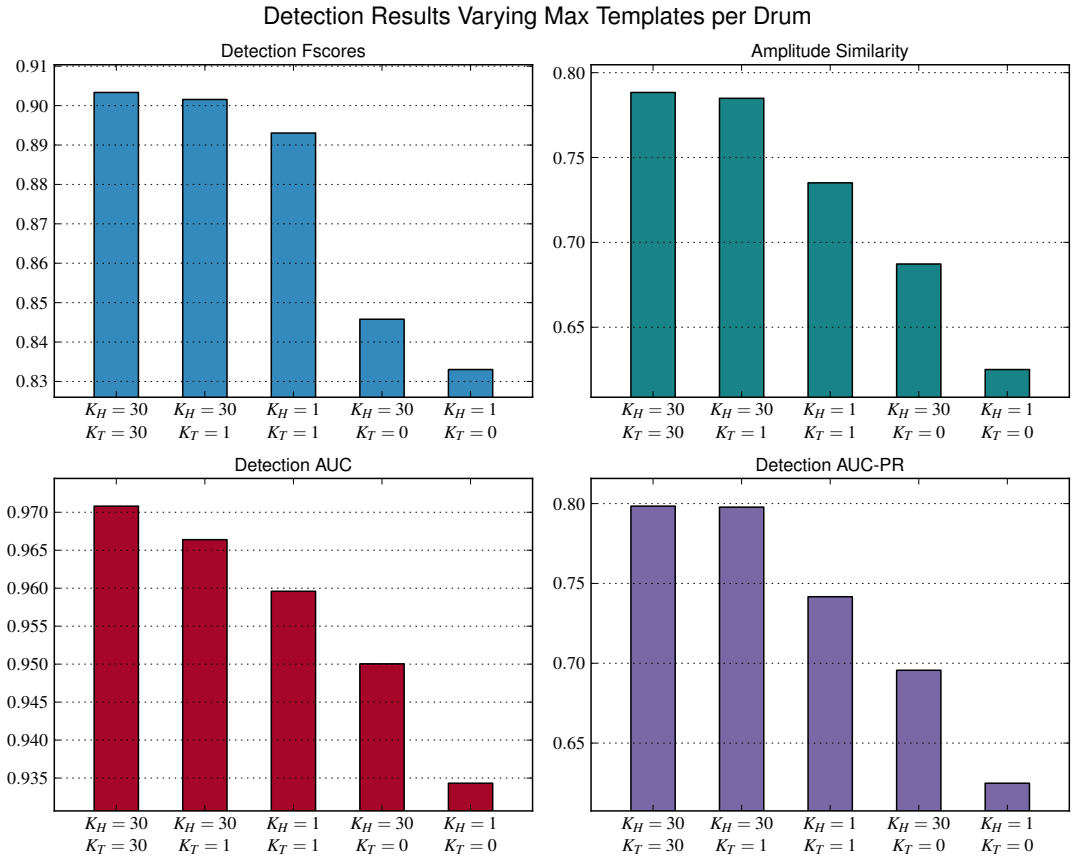

Figure 3.4: *Detection accuracy and amplitude quality while the maximum number of head and tail templates per drum, K_H and K_T, is varied.*

3.7.3 Discussion

The results I have presented suggest that the non-negative decomposition of drum onsets onto spectral templates can be greatly improved by the use of at least one tail template per drum class. In addition, using multiple head templates per drum has a positive effect on both detection accuracy and amplitude fidelity. To determine these multiple drum templates, I have proposed a gamma mixture model, a probabilistic agglomerative clustering approach that takes the perceptual spectral similarity of training data into account and does not rely on the computationally expensive and possibly unstable covariance calculations required by Gaussian mixture models.

The output produced by the drum detector is used by the beat tracking component presented in the next chapter and by the drum pattern analysis component presented in Chapter 5.

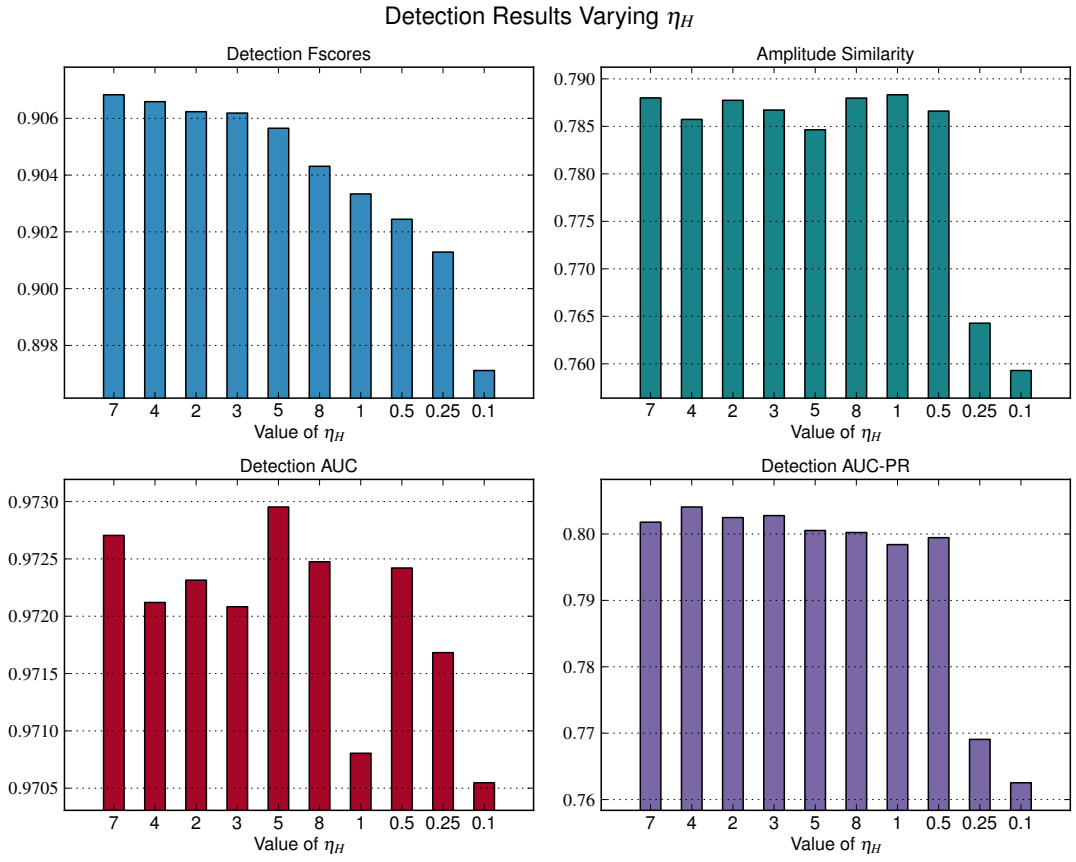

Figure 3.5: *Detection accuracy and amplitude quality while* η_H *is varied.* $K_H = K_T = 30$ *and* $\eta_H = 1$.

Chapter 4

Beat Tracking

4.1 What is Beat Tracking?

In order to define beat tracking, we must first define what is meant by this property called *beat* which we wish to track. In music, "beat" can refer to many different but related concepts in rhythm, such as pulse, tempo, meter, or groove. In popular music, many listeners refer to the contribution of the entire rhythm section and backing instruments as the "beat" (e.g., "This song has a good beat."). This definition of beat may overlap somewhat with the concept of *groove*, the experience of which has been defined in music perception research as "wanting to move some part of the body in relation to some aspect of the sound pattern" [54]. In contrast to these high-level psycho-musicological meanings of beat, the lower-level temporal-rhythmic concepts of pulse, tempo, and meter are typically the properties being estimated by a beat tracker.

In this sense, *beat* is a colloquial term for what musicologists call the *pulse* [37], or more formally, the *tactus* [53]. While these three terms will be used interchangeably in this chapter, it is important to note that theoretical purists may point out subtle distinctions between their definitions. This chapter will use these terms to refer to an underlying perceptual periodic pulse train with which listeners synchronize as they tap, clap, or dance along with music [27]. The *tempo* of a piece of music is the frequency of this pulse, which is typically measured in beats-per-minute (BPM). While the beat denotes the basic foot-tapping pulse in a piece of music, it is not the most fine-grained subdivision in the rhythmic hierarchy. Notes can occur at a rate much faster than the tempo, and this perceived higher-frequency pulse, called the *tatum*, occurs with a period equal to the smallest commonly occurring time interval between successive notes. The term tatum was coined by Jeff Bilmes to stand for "temporal atom" and as an homage to jazz pianist Art Tatum. Another way of thinking about the tatum is the "time division which most highly coincides with all note onsets" [10]. While the tatum subdivides

the beat, the *meter* is a metric structure that designates a higher-level period of musical repetition and synchronization and typically consists of an integer number of beats. The meter defines the number of beats that make up a *measure* (or *bar*), which is composed of a group of contiguous beats, the first of which is called the *downbeat*, or informally, "the one".

A measure of drum music is pictured in Figure 4.1, which helps to illustrate these concepts.

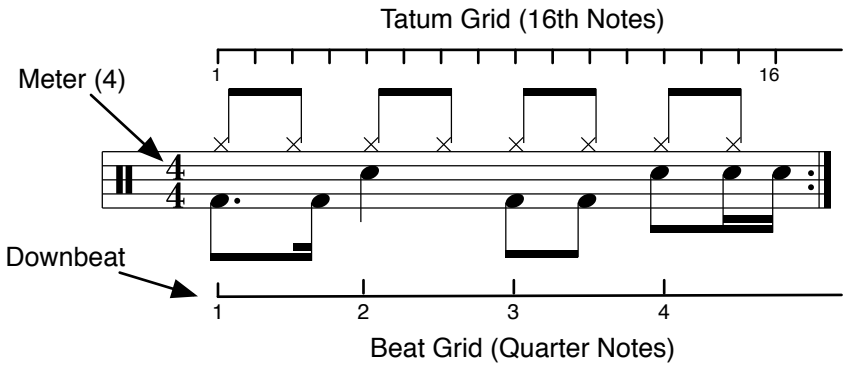

Figure 4.1: *Illustration of metrical structure concepts.*

The level of rhythmic *syncopation* in a piece of music describes the extent to which musical events occur at times which do not coincide with beat times (i.e., syncopated rhythms can be said to contain many off-beat or unexpected events) [27]. Highly syncopated rhythms are particularly problematic for most beat trackers, and this issue is addressed in the techniques covered in this chapter and Chapter 5.

Among the rhythmic elements defined above, beat trackers strive to make accurate estimates of the tactus pulse (both the tempo and beat locations). In addition to this required output, some beat trackers supplement the output with meter, downbeat, and/or tatum estimates, which help to characterize the hierarchical grid on which musical rhythm can be analyzed [32][48][21].

4.2 Existing Approaches

The vast majority of beat tracking work has focused on polyphonic music and is evaluated using datasets composed of popular music or ballroom dance tunes [70, 32, 48, 21, 35, 51]. Most such approaches make assumptions such as a steady tempo, are restricted to common meters such as 4/4 time, and are designed for offline analysis.

In contrast, early work by Dannenberg, Mont-Reynaud, and Allen focused on tracking beats and chord progressions in live jazz improvisations [20][1].

Compared to the beat tracking of solo drums, some aspects of polyphonic beat tracking, such as onset detection, can be more difficult due to the ambiguous onset locations of certain pitched instruments, while others, such as meter and downbeat detection, can be easier due to the presence of transitional cues from additional instruments.

Approaches that focus solely on tracking drums-only audio in real-time include work by Robertson [69] and Collins [17]. Robertson's approach tracks input from a bass drum microphone and makes initialization and steadiness assumptions on the beat pulse to be tracked. His implementation has been incorporated into a fully functioning real-time interactive percussion synchronization system that serves as a case study in his dissertation. Collins' work combines the beat tracking work of Laroche [51] and Goto [32] into a real-time drum beat tracker. The real-world functionality of these real-time drum tracking systems is impressive; however, the inflexibility imposed by the strong assumptions made leaves much more to be desired in terms of true rhythmic understanding.

The various methods used by existing beat tracking systems will now be covered. The task of determining the overall beat pulse is typically divided into beat period (or tempo) estimation and beat phase (or alignment) estimation. As a preprocessing step, the raw audio signal is converted into some sort of representation that contains information about the location of note onsets. This representation could be a list of onset times or a signal that shows the rate of energy increase or spectral novelty, similar to the many varieties of onset detection function (ODF) covered in Chapter 2.

4.2.1 Beat Period Estimation

The task of beat period estimation or tempo induction has been studied extensively outside of the realm of beat tracking. The ability to automatically detect the tempo of a song is useful for navigating music collections and can help listeners and DJs make more informed decisions on what to play. Many beat tracking systems carrying out tempo estimation as a first step before determining the alignment of the overall beat pulse.

Tuning a Non-Linear Oscillator

The beat tracking work of Large [50] uses an oscillator with the ability to adjust its period and phase in response to the arrival of rhythmic events. The oscillator has a "temporal receptive field" that is dynamically adjusted in order to tune the responsiveness of the period and phase updates. This approach is interesting in that the oscillator representing the beat pulse estimate itself is what is being dynamically adjusted during the beat tracking process; however, the approach

hasn't seen much use due to its lack of invariance to syncopation or tatum-interval onsets.

Clustering Inter-Onset Intervals

The beat trackers of Goto [33] and Dixon [23] take a list of onset times and cluster (or histogram) the intervals between adjacent onsets in order to determine the most prominent pulse period. This approach has the potential to be biased toward detecting the tatum period rather than the true beat period due to the comparison of only adjacent onsets. To deal with this, the interval between pairs of non-adjacent onsets can be considered as well.

Comb Filter Resonators

The work of Scheirer [70] and Klapuri [48] employs a bank of comb filter resonators to detect the relative strength of different beat period candidates. The comb filter for each candidate period processes an input onset detection function. The energy in a recent segment of the output of a comb filter represents the strength of the period of the filter.

Autocorrelation

Autocorrelation is a simple mathematical procedure for finding periodicities in random processes, so it makes sense that it is popular in the beat tracking world. The work of Davies [21], Ellis [24], later work by Goto [32], and many more utilize autocorrelation to detect salient periods. Because an autocorrelation function uses a range of time lags, it is an improvement over the clustering of inter-onset intervals. And compared to the comb filter approach, autocorrelation seems to provide a less noisy and more robust output signal to work with due to its use of multiplication between shifted signals rather than addition.

Multi-Agent System

Multi-agent systems track multiple simultaneous hypotheses of the beat period or phase and use some evaluation criteria to delegate between the estimates of each hypothesis. Each hypothesis can use slightly different parameters or strategies when processing the input audio and making period decisions. For example, Goto varies the onset detection parameters across each agent and a "manager" decides which agent is making the most reliable beat estimates [33].

Probabilistic Modeling

Some authors have formed probabilistic graphical models to exploit musical knowledge when selecting a suitable beat period. Hainsworth uses a Kalman filter to make state estimates for his "Rao-Blackwellised" and Brownian motion models [35]. Klapuri uses a joint Hidden Markov Model (HMM) [64] to infer tatum, tactus, and meter periods from comb filter energies [48]. A diagram representing the HMM is shown in Figure 4.4. Davies uses Rayleigh and Gaussian priors to accentuate certain periods of an input autocorrelation function [21]. Many of these approaches model both the prior probability of a certain beat period and the probability of transition between beat periods.

Dynamic Programming

In beat tracking, dynamic programming (DP) can be used to select a likely path of beat period estimates through time. Laroche uses dynamic programming techniques to efficiently compute a tempo path using a windowed autocorrelation function [51]. Klapuri uses the Viterbi algorithm to decode a likely path through his HMM-based period model [48]. The DP-based approach of Ellis computes an optimal path through his beat-onset alignment and tempo variation cost function [24].

4.2.2 Phase Estimation

For many beat trackers, phase estimation is done only after computing a beat period estimate. For others, beat phase and period estimates are made simultaneously. The dynamic non-linear oscillator of Large uses two very similar rules to adjust both its period and phase. Klapuri and Scheirer use the output of their comb filters (rather than the windowed comb filter energy as is used for period estimates) to determine the strength of beat alignment candidates. Goto and Davies cross-correlate the onset signal with a comb template at the detected beat period in order to determine beat phase. The dynamic programming approaches of Laroche and Ellis attempt to optimize a cost function which encourages alignment of beats and onsets.

Many of these approaches also use some sort of probabilistic weighting to prevent discontinuities in the beat phase estimates. Klapuri employs an HMM to make smooth phase estimates from the comb filter outputs. Davies uses a Gaussian weighting on his cross-correlation function to encourage the phase estimate to remain at its current value.

4.3 A New Approach to Beat Tracking for Live Drums

The approach to beat tracking presented here combines many of the techniques mentioned above with the goal of balancing both stability and responsiveness when tracking live, possibly improvised, drum performances. Beyond considering the real-time, causal, and reactive needs of such a system, there are no constraining assumptions made other than that there is a beat to track.

The approach outlined in the following uses a probabilistic graphical model to make an initial base period estimate, which seeds multiple context-dependent period and phase hypotheses. Figure 4.2 shows a block diagram of the system. In addition to the onset detection function (ODF), input to the system includes the drum-wise activation signals computed as described in Chapter 3.

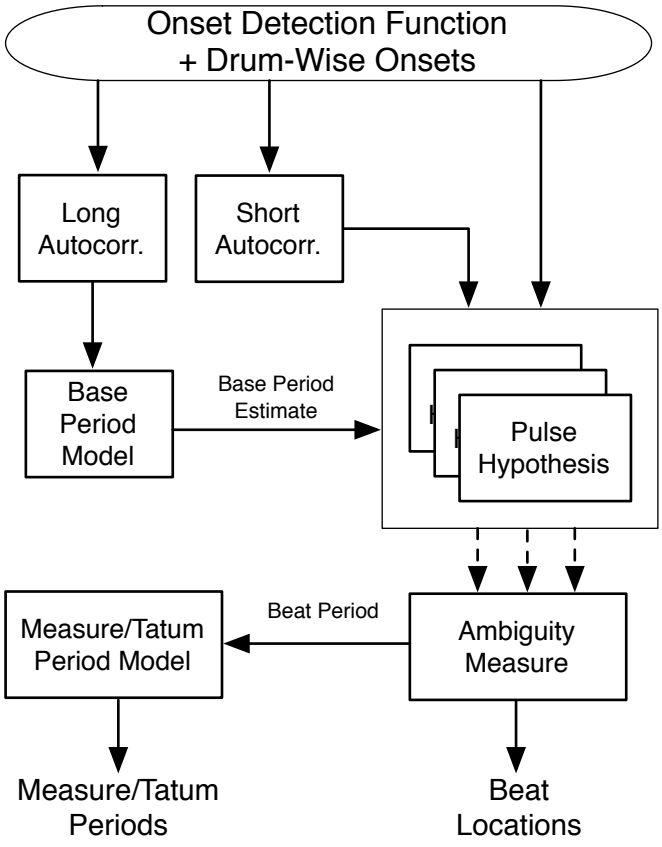

Figure 4.2: *Components of the beat tracking system described in this section.*

4.3.1 Multi-Signal, Multi-Scale Autocorrelation

For period detection, autocorrelation (AC) is carried out on the inputs to the beat tracker, i.e., the ODF and each drum-wise onset signal computed as described in Chapter 3. A separate AC function is computed for each input onset signal. As in Chapter 3, it is assumed we are operating at a sample rate of 44.1kHz with a 256 sample hop size (5.8ms), giving us a frame rate of 172Hz for each onset signal.

For beat tracking, the AC function (represented as $\mathbf{R}$ in the following) is typically computed using a summation over a fixed window size as shown in (4.1) for lag τ, window size W, window function $w(m)$, and output frame n. The window function can be used to give greater weight to more recent inputs.

$$\mathbf{R}_n(\tau) = \sum_{m=0}^{W-1} w(m)o(n-m)o(n-m-\tau) \tag{4.1}$$

For a maximum lag of T, this method requires $O(WT)$ multiply-adds per output frame and a buffer of length $W+T$ to hold past input values. Instead of using this method, we can use a leaky integrator to more cheaply compute the AC function.

$$\begin{aligned} \mathbf{R}_n(\tau) &= z_n(\tau) + \alpha\mathbf{R}_{n-1}(\tau) - 0.5z_{n-W}(\tau) && (4.2)\\ z_n(\tau) &= o(n)o(n-\tau) && (4.3)\\ \alpha &= 0.5^{1/W} && (4.4) \end{aligned}$$

Where α is the decay factor for the leaky integrator. This version requires only $O(T)$ multiply-adds per frame and uses the same size buffer for $o(n)$. The window function it implements is shown in Figure 4.3.

In choosing the autocorrelation window size there is a trade-off between stability and responsiveness. A longer window allows the autocorrelation function to be robust to small timing variations, while a smaller window allows for faster adaptation to tempo changes. For each onset signal, two AC functions are computed each using a different window size. The long-window AC is used to make a robust base period estimate, while the short-window AC is used to make more responsive updates to each period hypothesis. The leaky AC easily allows multiplication operations to be shared between ACs with different decay factors since the term $o(n)o(n-\tau)$ is unaffected by a change in this parameter.

The complete output of this autocorrelation stage includes both long and short window AC functions for the ODF and the activation signal of each drum. Also included are long and short "pattern" AC functions which are simply the summation across the corresponding ACs of the ODF and each drum. This pattern AC function is especially good for detecting the meter period since it emphasizes repetition periods that are present across all drums. Each AC function has a maximum lag equivalent to 5 seconds worth of frames, allowing 861 possible periods.

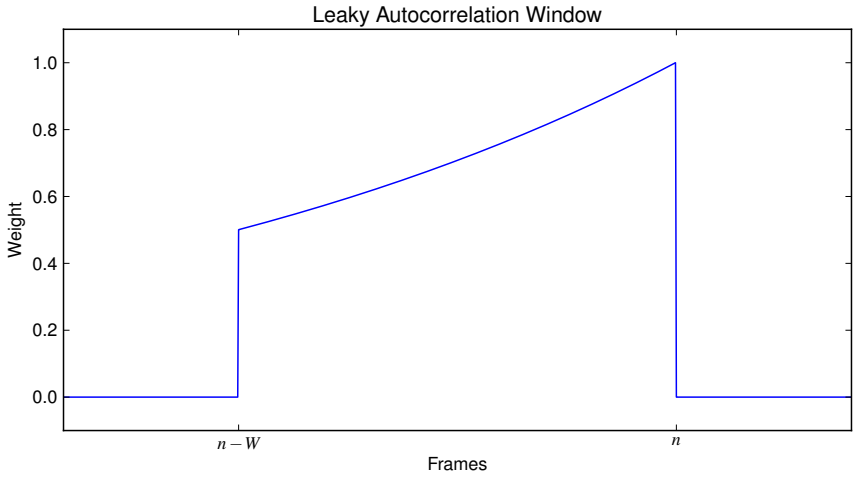

Figure 4.3: *Leaky autocorrelation window with a cliff.*

4.3.2 Base Period Model

In order to make an initial robust estimate of the beat period, a probabilistic graphical model is used similar to the HMM model used by Klapuri [48] shown in Figure 4.4. The Klapuri model attempts joint inference of τ^{A},τ^{B}, and τ^{C}, the tatum, tactus, and meter periods, which creates a very large state space that is infeasible to explicitly search over and requires an approximate beam search. For a max AC lag of 860 frames (about 5 seconds with a 5.8ms hop size), this yields 638 million possible combinations of the three periods. The meter and tatum periods are not quantities that are prone to change more often than every few seconds, so in my base period model, only inference of the tactus period, τ^{B}, is attempted, while possible tatum and meter periods remain as co-factors that influence the likelihood of the beat period, as shown in Figure 4.5.

In the Klapuri model, a single observed vector of comb filter outputs, $\mathbf{R}$, serves as input to the HMM and models the relative strength of the three periods being estimated. In the new model, there can be up to three separate AC functions, $\mathbf{R}^{\mathrm{A}}$,$\mathbf{R}^{\mathrm{B}}$,$\mathbf{R}^{\mathrm{C}}$, used to communicate the relative strength of the three periods separately. In the presented results, the long-window autocorrelation of the ODF is used for both $\mathbf{R}^{\mathrm{A}}$ and $\mathbf{R}^{\mathrm{B}}$, while the long "pattern" AC is used for the meter strength $\mathbf{R}^{\mathrm{C}}$ in this base period model.

The conditional probabilities (represented as directed edges in the model diagram) are modeled similarly to the Klapuri model. In order to make causal estimates of the beat period, τ_n^{B}, from the HMM shown in Figure 4.5, we can use the Viterbi algorithm to compute the final state of the partial path ending at the

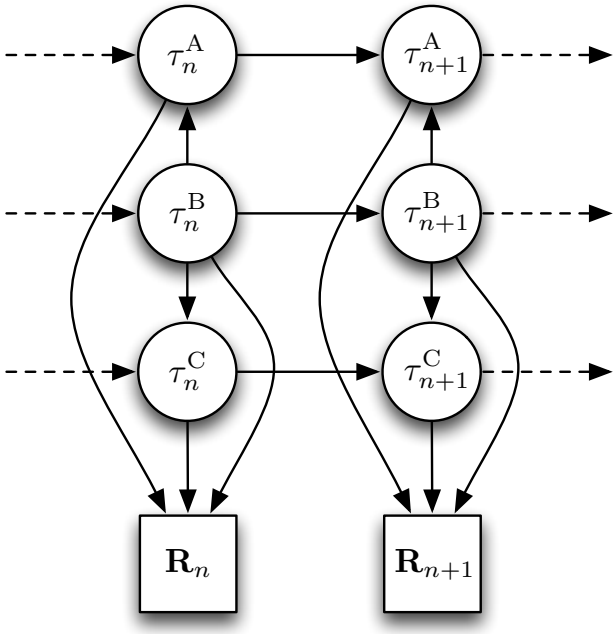

Figure 4.4: *The Klapuri model for period inference.*

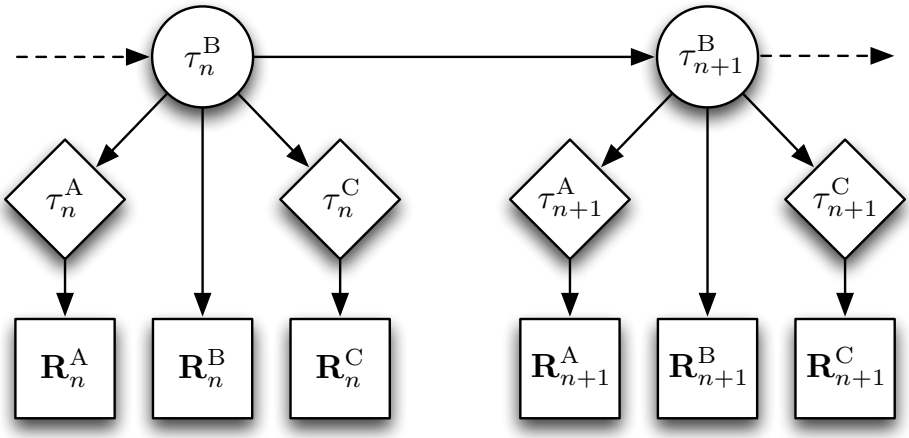

Figure 4.5: *Probabilistic graphical model for base period inference.*

current time [64]. The Viterbi algorithm requires that we compute the likelihood of the observed data under each possible hidden state. The directed edges in Figure 4.5 imply conditional independence between the observed vectors so that the observation likelihood factorizes as:

$$\mathrm{p}(\mathbf{R}^{\mathrm{A}}, \mathbf{R}^{\mathrm{B}}, \mathbf{R}^{\mathrm{C}} | \tau^{\mathrm{B}}) = \mathrm{p}(\mathbf{R}^{\mathrm{A}} | \tau^{\mathrm{B}}) \mathrm{p}(\mathbf{R}^{\mathrm{B}} | \tau^{\mathrm{B}}) \mathrm{p}(\mathbf{R}^{\mathrm{C}} | \tau^{\mathrm{B}}) \tag{4.5}$$

By Bayes' rule and the conditional independence implied by the model, the first and third factors in (4.5) can be computed using:

$$\mathrm{p}(\mathbf{R}^{\mathrm{A}}|\tau^{\mathrm{B}}) = \sum_{\tau^{\mathrm{A}}} \mathrm{p}(\mathbf{R}^{\mathrm{A}}|\tau^{\mathrm{A}})\mathrm{p}(\tau^{\mathrm{A}}|\tau^{\mathrm{B}}) \tag{4.6}$$

$$\mathrm{p}(\mathbf{R}^{\mathrm{C}}|\tau^{\mathrm{B}}) = \sum_{\tau^{\mathrm{C}}} \mathrm{p}(\mathbf{R}^{\mathrm{C}}|\tau^{\mathrm{C}})\mathrm{p}(\tau^{\mathrm{C}}|\tau^{\mathrm{B}}) \tag{4.7}$$

The likelihood of each individual observed vector given the corresponding hidden state is simply the value of the input autocorrelation function at the period of interest.

$$\mathrm{p}(\mathbf{R}^{\mathrm{A}}|\tau^{\mathrm{A}}) \propto \mathbf{R}^{\mathrm{A}}(\tau^{\mathrm{A}}) \tag{4.8}$$

$$\mathrm{p}(\mathbf{R}^{\mathrm{B}}|\tau^{\mathrm{B}}) \propto \mathbf{R}^{\mathrm{B}}(\tau^{\mathrm{B}}) \tag{4.9}$$

$$\mathrm{p}(\mathbf{R}^{\mathrm{C}}|\tau^{\mathrm{C}}) \propto \mathbf{R}^{\mathrm{C}}(\tau^{\mathrm{C}}) \tag{4.10}$$

The remaining conditional probabilities required by (4.6)–(4.7) are modeled as:

$$\mathrm{p}(\tau^{\mathrm{A}}|\tau^{\mathrm{B}}) \propto \mathrm{p}(\tau^{\mathrm{A}}) g_{\mathrm{AB}}\left(\frac{\tau^{\mathrm{B}}}{\tau^{\mathrm{A}}}\right) \tag{4.11}$$

$$\mathrm{p}(\tau^{\mathrm{C}}|\tau^{\mathrm{B}}) \propto \mathrm{p}(\tau^{\mathrm{C}}) g_{\mathrm{BC}}\left(\frac{\tau^{\mathrm{C}}}{\tau^{\mathrm{B}}}\right) \tag{4.12}$$

And the transition probabilities for the beat period hidden states are modeled as:

$$\mathrm{p}(\tau^{\mathrm{B}}_n|\tau^{\mathrm{B}}_{n-1}) = \mathrm{p}(\tau^{\mathrm{B}}_n) f\left(\frac{\tau^{\mathrm{B}}_n}{\tau^{\mathrm{B}}_{n-1}}\right) \tag{4.13}$$

The priors for the tatum, tactus, and meter periods are given by log-normal distributions as suggested by Parncutt [60].

$$\mathrm{p}(\tau^i) = \frac{1}{\tau^i(\ln\sigma^i_{\mathrm{p}})\sqrt{2\pi}} \exp\left[-\frac{1}{2(\ln\sigma^i_{\mathrm{p}})^2}(\ln\tau^i - \ln\mu^i_{\mathrm{p}})^2\right] \tag{4.14}$$

Here I use different parameters compared to those Klapuri uses in order to allow for a wider spread for each period as shown in Figure 4.6. The parameters used are

$$\mu^{\mathrm{A}}_{\mathrm{p}} = 0.18, \qquad \sigma^{\mathrm{A}}_{\mathrm{p}} = 1.47 \tag{4.15}$$

$$\mu^{\mathrm{B}}_{\mathrm{p}} = 0.72, \qquad \sigma^{\mathrm{B}}_{\mathrm{p}} = 1.82 \tag{4.16}$$

$$\mu^{\mathrm{C}}_{\mathrm{p}} = 2.52, \qquad \sigma^{\mathrm{C}}_{\mathrm{p}} = 1.82 \tag{4.17}$$

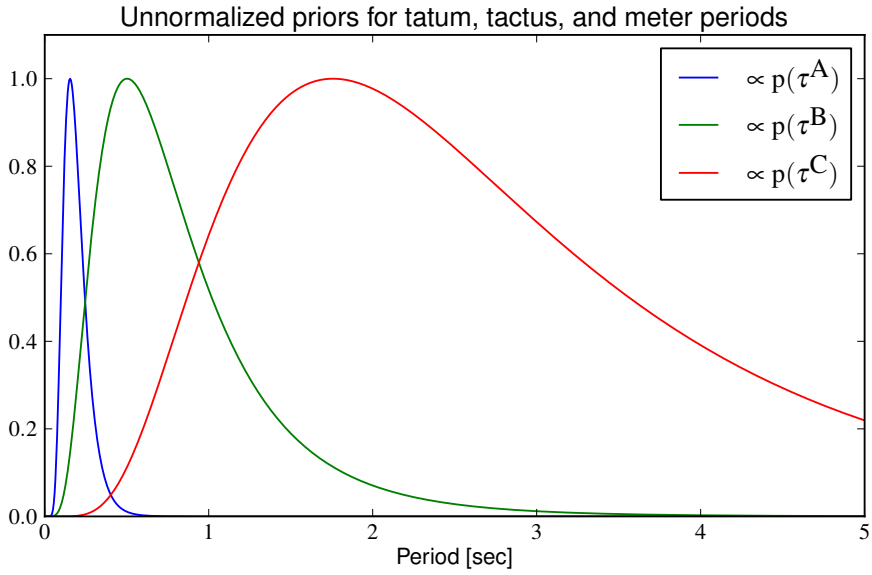

Figure 4.6: *The shape of the prior for each hidden state period.*

The conditional probabilities of simultaneous periods in (4.6)–(4.7) rely on the following Gaussian mixture models which are functions of the ratio between the two periods.

$$g_{\mathrm{AB}}\left(\frac{\tau^{\mathrm{B}}}{\tau^{\mathrm{A}}}\right) = \sum_{m=1}^{M_{\mathrm{AB}}} w_m^{\mathrm{AB}} \frac{1}{\sigma_{\mathrm{AB}}\sqrt{2\pi}} \exp\left[-\frac{1}{2\sigma_{\mathrm{AB}}^2}\left(\frac{\tau^{\mathrm{B}}}{\tau^{\mathrm{A}}} - m\right)^2\right] \tag{4.18}$$

$$g_{\mathrm{BC}}\left(\frac{\tau^{\mathrm{C}}}{\tau^{\mathrm{B}}}\right) = \sum_{m=1}^{M_{\mathrm{BC}}} w_m^{\mathrm{BC}} \frac{1}{\sigma_{\mathrm{BC}}\sqrt{2\pi}} \exp\left[-\frac{1}{2\sigma_{\mathrm{BC}}^2}\left(\frac{\tau^{\mathrm{C}}}{\tau^{\mathrm{B}}} - m\right)^2\right] \tag{4.19}$$

The standard deviations of the Gaussian mixture components, σ_{AB} and σ_{BC}, are set to 0.1 and 0.3, respectively, and the values of the weight parameters, w_m^{AB} and w_m^{BC}, used in the experiments are shown in Figure 4.7. If the beat period, τ^{B}, is equivalent to the quarter note interval, then a ratio of $\tau^{\mathrm{B}}/\tau^{\mathrm{A}} = 3$ means the tatum represents eighth note triplets and $\tau^{\mathrm{B}}/\tau^{\mathrm{A}} = 8$ means the tatum represents 32nd notes. The likelihood of these different tatum ratios are controlled by the weights, w_m^{AB}. The weights w_m^{BC} control the likelihood of different meters.

Lastly, the function needed by the state transition probabilities in (4.13) is defined such that the logarithm of the ratio between the two states is normally distributed.

$$f\left(\frac{\tau_n^{\mathrm{B}}}{\tau_{n-1}^{\mathrm{B}}}\right) = \frac{1}{(\ln\sigma_f)\sqrt{2\pi}} \exp\left[-\frac{1}{2(\ln\sigma_f)^2}(\ln\tau_n^{\mathrm{B}} - \ln\tau_{n-1}^{\mathrm{B}})^2\right] \tag{4.20}$$

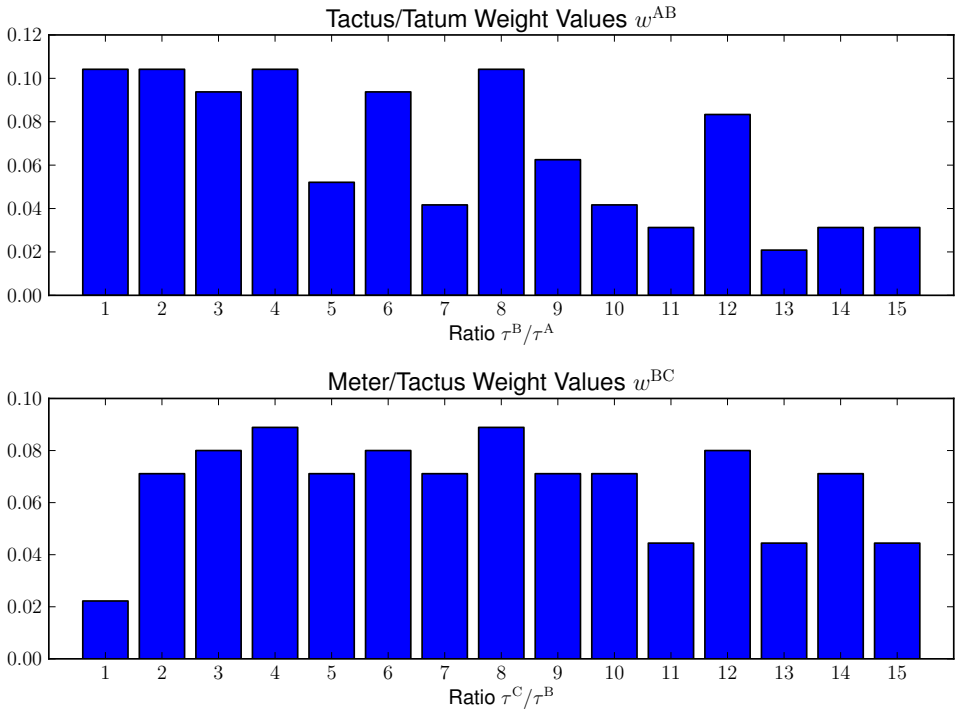

Figure 4.7: *GMM weights for simultaneous period ratios. Top - w_m^{AB} tactus/tatum weights. Bottom - w_m^{BC} meter/tactus weights.*

A value of $\sigma_f = 1.2$ is used, making a period increase of 20% about 40% less likely than a transition to the same state. (i.e., $\mathrm{p}(\tau_n^{\mathrm{B}} = 1.2\tau_{n-1}^{\mathrm{B}}|\tau_{n-1}^{\mathrm{B}}) \approx 0.6\mathrm{p}(\tau_n^{\mathrm{B}} = \tau_{n-1}^{\mathrm{B}}|\tau_{n-1}^{\mathrm{B}})$)

Now we are able to compute the terms $\mathrm{p}(\mathbf{R}^{\mathrm{A}}, \mathbf{R}^{\mathrm{B}}, \mathbf{R}^{\mathrm{C}}|\tau^{\mathrm{B}})$ and $\mathrm{p}(\tau_n^{\mathrm{B}}|\tau_{n-1}^{\mathrm{B}})$ required by the Viterbi algorithm.

Practical Considerations

The length of each input autocorrelation function gives us 861 possible states for each of the three period types. While the fact that we are only inferring the value of τ^{B} makes this process much more feasible than choosing between 861^3 states, we can reduce the amount of computation further. Given the prior distributions on each state, shown in Figure 4.6, it follows that only a subset of each of the 861 states for each period type are likely to ever be achieved. The bounds for each state are restricted to be where 99.9% of the prior probability is contained. For τ^{A}, I compute and search only over states 7–104 (40ms–600ms), for τ^{B}, states 11–512 (63ms–3sec or 20–940 BPM), and for τ^{C}, 67–861 (0.38–5sec). These bounds can be adjusted for special types of music, for example, very slow music with a

tatum period greater than 600ms.

Since this base period model is only seeding the more fine-grained pulse trackers in subsequent steps, there is no reason to make the base period estimate every frame. By reducing the update frequency to around 3 times per second, the computation involved in this step becomes quite insignificant.

4.3.3 Multi-Hypothesis Pulse Tracking

Once a base period is estimated in the previous step, multiple pulse trackers are seeded with periods. Important periods to track include the base period itself and intervals equal to double and half the base period, for reasons that will explained shortly.

Period Tracking

When a single pulse tracker is given a period to track it follows the short-window AC function of the ODF in a small Gaussian-weighted window centered on the initial period, similar to the window used by Davies [21]. Each time the period is updated, the Gaussian window is re-computed and centered at the new period. This allows the period tracker to adjust its estimates slightly from frame to frame while preventing large period transitions, and the shorter autocorrelation decay allows the period estimates to react more quickly to tempo changes. The Gaussian window used is identical to the one shown in (4.20), but a much tighter spread of 1–2% is used ($\sigma_f \in [1.01, 1.02]$). Figure 4.8 shows how changes in the short AC window size, W_{short}, and the value of σ_f affect the stability and responsiveness of the period tracker when a significant tempo transition occurs.

An additional issue that becomes important when tracking short periods is that the tempo resolution in BPM becomes very coarse as the beat period is decreased. For example, with a frame rate of ∼172Hz (5.8ms frames), a period of 40 frames corresponds to 258.4 BPM, while a period of 41 frames corresponds to 252.1 BPM. If two oscillators with these periods are started simultaneously, they will be 180° out of phase in less than 5 seconds. While these high tempos may seem extreme, the period they represent corresponds to eighth notes with a quarter note equal to ∼125 BPM.

An obvious way of dealing with this issue would be to increase the overall frame rate; however, this can significantly increase the computational and memory requirements. Another method would be to interpolate a region of interest in the AC function using Fourier interpolation[1] in order to increase the period resolution. If the BPM is equal to $60 f_r/\tau$ with frame rate f_r and period τ, then the rate of change in BPM per period frame is $\delta\text{BPM}/\delta\tau = -60 f_r/\tau^2$. At

[1]Fourier interpolation involves taking the Discrete Fourier Transform (DFT) of a signal, increasing the frequency range by appending zeros to the DFT, and taking the inverse DFT.

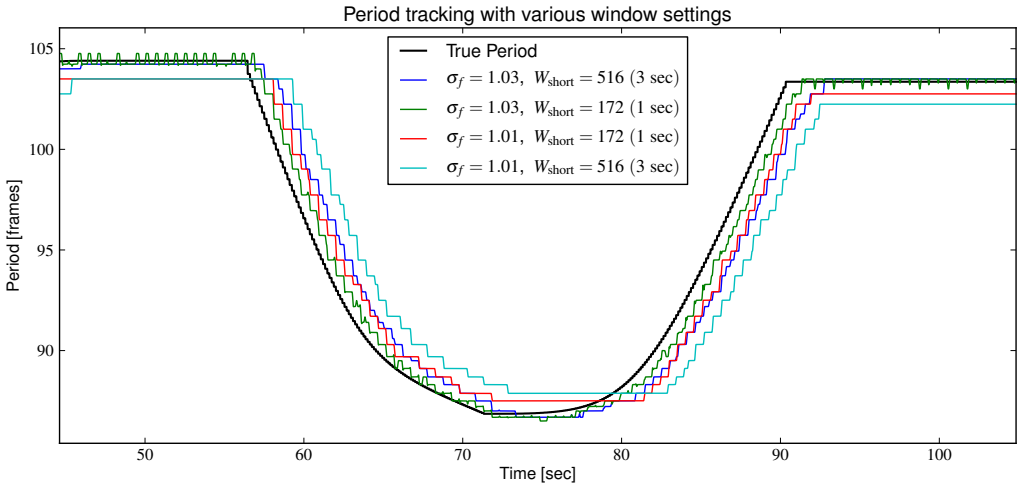

Figure 4.8: *Period tracking is shown using various values of the short AC window size, W_{short} and the spread of the tracking window, σ_f.*

$\tau = 100$, this resolution is about 1 frame/BPM, but at $\tau = 40$ this drops to around 0.15 frames/BPM. In my experiments, I use interpolation to maintain a minimum value of frames/BPM, $\mathcal{I}_{\text{BPM}}$, based on the current period. I use $\mathcal{I}_{\text{BPM}} = 2$ frames/BPM, which results in interpolation being used often (when $\tau < 143$), and $\mathcal{I}_{\text{BPM}} = 0.1$ frames/BPM, which means interpolation is used only when $\tau < 32$.

Phase Tracking

Now that we have fine-grained period estimates for each pulse hypothesis, we can estimate the alignment of the pulse. For each hypothesis, a separate reference pulse is maintained that runs at the detected period. I use the pulse described by Large in [50]:

$$p(n) = 1 + \tanh \gamma(\cos 2\pi\phi(n) - 1) \tag{4.21}$$

Where $\phi(n)$ is the phase of the reference pulse at the current frame and γ adjusts the width of the pulse. The value of $\phi(n)$ is incremented by the reciprocal of the current detected period at each frame. This causes the phase of the pulse to be increased by one after a period worth of frames has elapsed. I use $\gamma = 8$, which creates a pulse like that shown in Figure 4.9. Cross-correlation between this reference pulse and the onset detection function is used to estimate the beat phase. As with autocorrelation, I compute the cross-correlation in an online, recursive manner. Let $p(n)$ and $o(n)$ be the value of the reference pulse and the ODF at

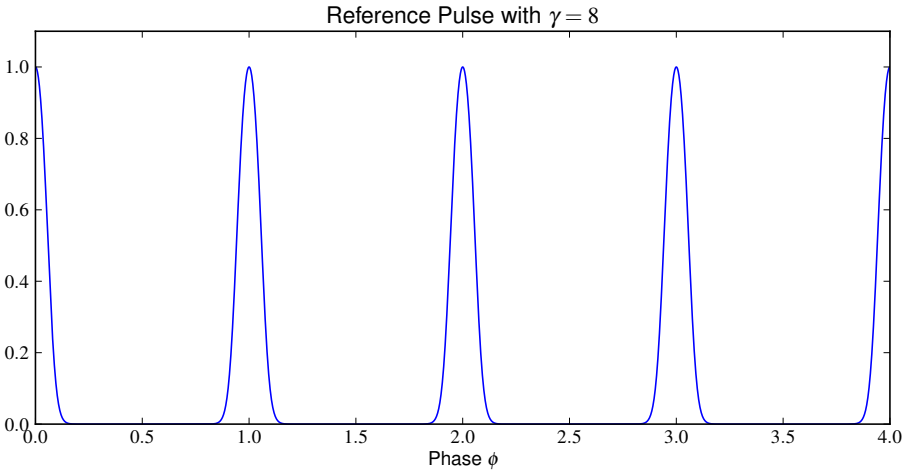

Figure 4.9: *The above reference pulse is used for beat alignment.*

frame n, respectively. The cross-correlation, $\mathbf{C}_n(\varphi)$ is computed as shown below.

$$\mathbf{C}_n(\varphi) = y_n(\varphi) + \alpha \mathbf{C}_{n-1}(\varphi) - 0.5 y_{n-W_c}(\varphi) \quad (4.22)$$

$$y_n(\varphi) = p(n) o(n - \varphi) \quad (4.23)$$

$$\alpha = 0.5^{1/W_c} \quad (4.24)$$

The above implements a decaying window with a cliff at W_c as in Figure 4.3. Updates to $\mathbf{C}_n(\varphi)$ are only performed for values of φ between 0 and the current period.

Instead of setting W_c to be a constant value, W_c is dynamically adjusted to be a multiple of the current period, $W_c = \rho_c \hat{\tau}_p$, so that a constant number of pulse periods are considered in the cross-correlation. I experiment with values of W_c equal to 4 and 8 times the current period ($\rho_c = \{4, 8\}$).

Once the cross-correlation function has been computed, phase decisions are made using another hidden Markov model. This model is considerably simpler than the base period model. The cross-correlation models the observation probabilities given the hidden state, which represents the phase offset in frames of the beat pulse compared with the reference pulse.

The transition probabilities between phase states with $\hat{\tau}_p$-periodic circular distance $\Delta_{\hat{\tau}_p}(\varphi_n, \varphi_{n-1})$ are modeled such that the circular distance as a percentage of the current period is normally distributed.

$$\mathrm{p}(\varphi_n | \varphi_{n-1}) = \frac{1}{\sigma_\varphi \sqrt{2\pi}} \exp\left[-\frac{1}{2\sigma_\varphi^2} \left(\frac{\Delta_{\hat{\tau}_p}(\varphi_n, \varphi_{n-1})}{\hat{\tau}_p} \right)^2 \right] \quad (4.25)$$

$$\Delta_{\hat{\tau}_p}(\varphi_n, \varphi_{n-1}) = \min\left[\delta_{\hat{\tau}_p}(\varphi_n, \varphi_{n-1}), \hat{\tau}_p - \delta_{\hat{\tau}_p}(\varphi_n, \varphi_{n-1}) \right] \quad (4.26)$$

$$\delta_{\hat{\tau}_p}(\varphi_n, \varphi_{n-1}) = |\varphi_n - \varphi_{n-1}| \mod \hat{\tau}_p \quad (4.27)$$

The parameter σ_φ controls the stability of the phase estimates, and in experiments, I use $\sigma_\varphi \in \{0.01, 0.05\}$.

Once the phase is determined, causal predictions of the next beat are made by adding the current beat period to the most recently detected beat location.

Managing Hypotheses

As mentioned above, multiple pulse hypotheses are maintained. Each hypothesis is seeded with a period that forms a ratio with the base period detected in Section 4.3.2. Because double and half period errors are very common in beat tracking, important ratios are the base period itself, half the base period, and double the base period. Other ratios, such as 3:2 and 2:3 can be used in order to attempt to deal with polyrhythms, but here I only evaluate the three previously mentioned ratios.

If the period of any hypothesis strays too far for too long from the intended ratio with the base period, it is corrected back to the intended value. This also allows the base period model to re-seed the pulse hypotheses in the presence of abrupt rhythmic changes.

Initially, the base period hypothesis is the active hypothesis and is responsible for outputting the final beat locations. A transition to an alternative hypothesis is carried out when two criteria are met:

1. The period of the hypothesis has varied less than 5% over the previous two seconds.

2. The average *ambiguity* of the phase estimates of the alternative hypothesis over the previous two seconds is sufficiently less than that of the current hypothesis.

I define *ambiguity* as a normalized circular moment about the current phase estimate ($\hat{\varphi}_n$) of the cross-correlation function.

$$\mathcal{A}_n = \frac{\sum_{l=0}^{\hat{\tau}_p - 1} [\mathbf{C}_n(l)]^j \left[\frac{\Delta_{\hat{\tau}_p}(l, \hat{\varphi}_n)}{\hat{\tau}_p} \right]^k}{\sum_{l=0}^{\hat{\tau}_p - 1} [\mathbf{C}_n(l)]^j} \tag{4.28}$$

Where $\Delta_{\hat{\tau}_p}(\cdot, \cdot)$ is the circular distance as defined in (4.26). I use $j = 2$ and $k = 1$ in the above, which makes the calculation a kind of first moment about the current phase.

This ambiguity measure is meant to quantify how confident the tracker is that its phase estimate is correct. A high ambiguity value can mean there is

a significantly different phase value that explains the cross-correlation function almost equally as well as the current estimate. Or, it could just mean that the tracker is having trouble locking onto the beat. Either way, a hypothesis that outputs high ambiguity values is an undesirable hypothesis.

In criterion #2 above, the average ambiguity of an alternative hypothesis must be less than a certain percentage of the average ambiguity of the current hypothesis. In my experiments, the alternative-hypothesis ambiguity must be less than 0.3 times the current-hypothesis ambiguity when transitioning to a hypothesis with half the period of the current hypothesis and 0.6 for a transition to a period double that of the current hypothesis. These values tend to favor the faster half period hypothesis when it is able to steadily track beats, due to the fact that the closer the pulse period gets to the tatum, the less phase ambiguity exists.

Halving or doubling the beat period in order to find the most easily tracked interval may seem undesirable because the output tempo is then being doubled or halved. However, the beat tracker is aware of the ratio the current hypothesis forms with the base period. For example, if the base period represents quarter notes, the tracker knows that the half period hypothesis is tracking eighth notes, and determining quarter note alignment of the eighth note grid is quite easy using the drum pattern analysis framework presented in Chapter 5.

4.3.4 Meter and Tatum Inference

Estimating the meter and tatum periods, τ_{C} and τ_{A}, is accomplished using the same graphical model shown in Figure 4.5. The two periods are estimated independently given the beat period of the current pulse hypothesis, τ_{B}, and the corresponding autocorrelation function, $\mathbf{R}_{\mathrm{A}}$ or $\mathbf{R}_{\mathrm{C}}$.

$$\mathrm{p}(\tau_{\mathrm{A}}|\tau_{\mathrm{B}},\mathbf{R}_{\mathrm{A}}) \propto \mathrm{p}(\tau_{\mathrm{A}},\mathbf{R}_{\mathrm{A}}|\tau_{\mathrm{B}}) = \mathrm{p}(\mathbf{R}_{\mathrm{A}}|\tau_{\mathrm{A}})\mathrm{p}(\tau_{\mathrm{A}}|\tau_{\mathrm{B}}) \quad (4.29)$$

$$\mathrm{p}(\tau_{\mathrm{C}}|\tau_{\mathrm{B}},\mathbf{R}_{\mathrm{C}}) \propto \mathrm{p}(\tau_{\mathrm{C}},\mathbf{R}_{\mathrm{C}}|\tau_{\mathrm{B}}) = \mathrm{p}(\mathbf{R}_{\mathrm{C}}|\tau_{\mathrm{C}})\mathrm{p}(\tau_{\mathrm{C}}|\tau_{\mathrm{B}}) \quad (4.30)$$

The terms required in the above can be found in (4.8)–(4.12).

These estimates need to be computed no more often than once or twice a second since transitions occurring more often than this would be unbecoming a drummer.

Instead of just choosing the values for τ_{A} and τ_{C} that maximize the probabilities above, we can compute the maximum probability in a window that corresponds to each meter and tatum ratio of interest. This gives us an idea about the relative strengths of each meter or tatum ratio across time and can help to make more robust long term estimates. I use windows with widths equal to 30% of the beat period for meter inference and 10% of the target tatum period for tatum inference.

After detecting the meter, aligning the beat grid with measure boundaries (downbeat detection) is done using the drum pattern analysis system in Chapter 5.

4.4 Evaluation

4.4.1 Test Set

Because there is a lack of drums-only datasets suitable for beat tracking evaluation, I constructed my own. The data was recorded using Roland V-Drums[2]. The output from this electronic drum set is recorded to a midi representation which is then converted to an audio signal using the multi-sampled drum library Superior Drummer 2.0[3].

The dataset consists of 84 drum tracks lasting 20–40 seconds each, totalling about 45 minutes of drum music. Each track was performed to a click track in order to simplify the annotation process. The tempos vary between 77 and 180 BPM, and the styles include rock (including double and half time feels), funk, jazz, metal, blast beats, tribal rhythms, and sections of drum roll fills at binary and ternary subdivisions of the beat. As far as the meter goes, there are 35 tracks in 4/4 time, 26 in 6/8, 7 in 5/4, 5 in 7/4, 4 in 3/4, 4 in 7/8, and 3 in 5/8.

For each genre and time signature, the rhythmic complexity varies from very basic to difficult enough that an experienced musician could potentially have trouble tracking the beat.

4.4.2 Beat Tracking Evaluation Method

There have a been significant attempts to standardize the way beat tracking algorithms are evaluated [55][22]. To evaluate my beat tracker, I used the "continuity-based" evaluation method covered in [22]. This method takes a sequence of input beat locations β_b and compares them with a sequence of ground-truth beat annotations a_j. The intervals between two successive input beats and two successive ground-truth beats are respectively defined as:

$$\delta_j = \beta_j - \beta_{j-1} \tag{4.31}$$

$$\tau_j = a_j - a_{j-1} \tag{4.32}$$

In order for an input beat β_b to be labeled as correct, three conditions must hold:

$$1. \quad a_j - \theta_\varphi \tau_j < \beta_b < a_j + \theta_\varphi \tau_j \tag{4.33}$$

$$2. \quad a_{j-1} - \theta_\varphi \tau_{j-1} < \beta_{b-1} < a_{j-1} + \theta_\varphi \tau_{j-1} \tag{4.34}$$

$$3. \quad (1 - \theta_\tau)\tau_j < \delta_b < (1 + \theta_\tau)\tau_j \tag{4.35}$$

The first condition requires that the input beat in question lies within some tolerance window around a ground truth beat. The width is a percentage of the

[2] http://rolandus.com

[3] http://toontrack.com

current ground-truth beat period defined by θ_φ. The second condition requires that the previous input beat lies near enough the previous ground truth beat. The third puts requirements on the interval between the last two input beats, so that the interval must be within θ_τ percent of the true beat period.

In my evaluation, I use $\theta_\varphi = 0.25$ and $\theta_\tau = 0.175$. The standard approach is to use 0.175 for both, but I've softened the alignment constraint due to the accuracy of the phase of the beat annotations in the test set (especially at higher tempos).

Once we have established the correctness of the input beats, the overall beat accuracy for a track is evaluated in terms of the maximum number of contiguous correct beats (labeled "Cont." in the results) and the total number of correct beats (labeled "Total" in the results). These values are then divided by the total number of ground truth beats to give an accuracy percentage. In addition to comparing the input beats to a beat sequence at the correct metrical level (CML), the input sequence is also compared with ground-truth beat sequences at double and half the actual tempo, and the best result of the three metrical levels is recorded for both the contiguous and total results. This "allowed metrical level" result is labeled "AML" in the results.

The period estimates coming from the active hypothesis are evaluated as well. The period at each frame is deemed correct if it is within 5% of the true beat period. I report AML values for the period results as well, and the overall accuracy is presented in terms of the percentage of correct frames.

For meter estimation, I report the percentage of correct frames for both CML (correct meter) and AML (double or half the correct meter allowed as well).

4.4.3 Results

The beat tracking accuracy was evaluated for 63 variations of the parameter set. Parameters varied {and their values} include:

- Hypo. $\in$ {"Base","Alt"}: Whether the base period hypothesis is used or alternative hypotheses are considered based on their ambiguity (pg. 41).
- $W_{\text{long}} \in \{6\,\text{sec}, 12\,\text{sec}\}$ - The autocorrelation window width used to determine the base period (pg. 32).
- $W_{\text{short}} \in \{1\,\text{sec}, 3\,\text{sec}\}$ - The autocorrelation window width used to determine the period for each hypothesis (pg. 32).
- $\sigma_f^H \in \{1.01, 1.02\}$ - The spread of the Gaussian window used for hypothesis period tracking (σ_f in Figure 4.8 on pg. 39).
- $\mathcal{I}_{\text{BPM}} \in \{0.1, 2\}$ - The interpolation target in terms of the minimum allowed autocorrelation resolution in frames/BPM for the period tracker (pg. 38).

- $\rho_c \in \{4, 8\}$ - The number of beat periods to include in the cross-correlation window size W_c (pg. 40).
- $\sigma_\varphi \in \{0.01, 0.05\}$ - The spread of the phase tracking transition window (pg. 40).

First we'll look at the best and worst performers in each result category. The top portion of Table 4.1 shows the parameters used in each of the nine trials giving the best and worst results. The lower portion shows the accuracy results for the categories discussed in Section 4.4.2. The best result in each category is boxed and shown in bold, and the worst result is boxed and shown in italic.

The most important point to make here is the difference in CML and AML results, depending on whether the tracker considered following less ambiguous, alternative hypotheses. The best CML beat accuracy always goes to a tracker that only uses the base period hypothesis, while the best AML accuracy always goes to a tracker that follows the least ambiguous hypothesis. This makes sense because the ambiguity measure tends to favor less ambiguous subdivisions of the beat, which frequently takes the pulse out of the "correct metrical level". However, subdividing the beat pays off greatly when evaluating with the AML criterion as can be seen in the "AML Total %" column for trials A and C. This is the intended behavior of the ambiguity measure, especially since the drum pattern analysis work presented in Chapter 5 can accurately align a subdivision of the beat with the actual beat grid. Trial G illustrates what happens when each parameter is set to the more responsive option: all of the estimates become unstable and performance degrades considerably.

To give a better overall picture of the effect of parameter variations on performance numbers, I will present a series of stem plots that show the sorted accuracy results for each category. Each data point will be colored according the value of the parameter that is being examined. We will look at the alternative hypothesis parameter, "Hypo.", first.

The data shown in Figure 4.10 shows an obvious distinction between CML and AML performance and the two values of Hypo. Nowhere is it more apparent than in the AML Total results in the lower right plot, in which every single tracker considering alternative beat period hypotheses outperformed every tracker considering only the base period. The CML Total results clearly favor the base-period-only trackers, which is expected.

The effect of autocorrelation window size on period detection accuracy is examined in Figure 4.12 and Figure 4.13. It is clear that the longer window sizes yield significantly better results for period detection; however, this is at the expense of responsiveness to tempo changes, which I do not attempt to quantify here.

For σ_f^H and σ_φ, the tracking results are slightly better for the more stable (smaller spread) values of the parameters, $\sigma_f^H = 1.01$ and $\sigma_\varphi = 0.01$, but the

Trial	Hypo.	W_{long}	W_{short}	σ_f^H	$\mathcal{I}_{\text{BPM}}$	ρ_c	σ_φ
A	Base	12 sec	3 sec	1.01	0.5	4	0.01
B	Alt	6 sec	3 sec	1.01	0.5	8	0.05
C	Alt	12 sec	3 sec	1.01	10	4	0.01
D	Base	12 sec	3 sec	1.01	10	4	0.01
E	Alt	12 sec	3 sec	1.01	0.5	4	0.01
F	Base	12 sec	3 sec	1.02	10	4	0.05
G	Alt	6 sec	1 sec	1.02	0.5	4	0.05
H	Base	6 sec	1 sec	1.02	0.5	4	0.05
I	Alt	12 sec	3 sec	1.01	10	4	0.05

Trial	CML Cont. %	CML Total %	AML Cont. %	AML Total %	CML Period %	AML Period %	CML Meter %	AML Meter %
A	**51.6**	**56.7**	56.1	62.0	67.6	83.6	51.8	63.0
B	47.6	52.2	**63.0**	70.8	54.0	82.5	47.9	64.0
C	45.9	51.3	60.3	**71.8**	52.0	83.4	45.7	64.8
D	50.5	56.2	54.6	61.2	**67.6**	83.4	51.7	62.3
E	45.7	50.5	62.0	71.7	50.7	**84.0**	45.7	**66.0**
F	46.7	55.3	51.3	60.8	66.9	82.8	**52.1**	63.2
G	*29.1*	*46.5*	*35.3*	62.8	*47.6*	*73.4*	44.7	58.5
H	34.1	51.3	37.3	*56.3*	61.2	76.7	47.8	*58.3*
I	42.9	48.7	56.9	68.8	49.7	83.2	*44.2*	64.3

Table 4.1: *Best and worst trials for each result category. Top - parameters used in each trial. Bottom - accuracy numbers. Best results are shown in boxed bold. Worst results are shown in boxed italic.*

effect isn't as striking as that of the Hypo parameter. For ρ_c, the results tend to favor the shorter window ($\rho_c = 4$); however, the best AML Cont. results are achieved with a tracker using $\rho_c = 8$.

Overall, the strongest conclusion that can be made is that following an alternative beat period hypothesis in the presence of uncertainty can greatly improve tracking results when it is acceptable to output beats at a different metrical level than the annotated beats. The main benefit seems to come when subdividing the original beat hypothesis to have a period that is half of the detected base period. The next chapter will cover the pattern analysis techniques that, among other things, can be used to align a beat grid with higher-level metrical structures, including measure boundaries, quarter notes, half notes, etc.

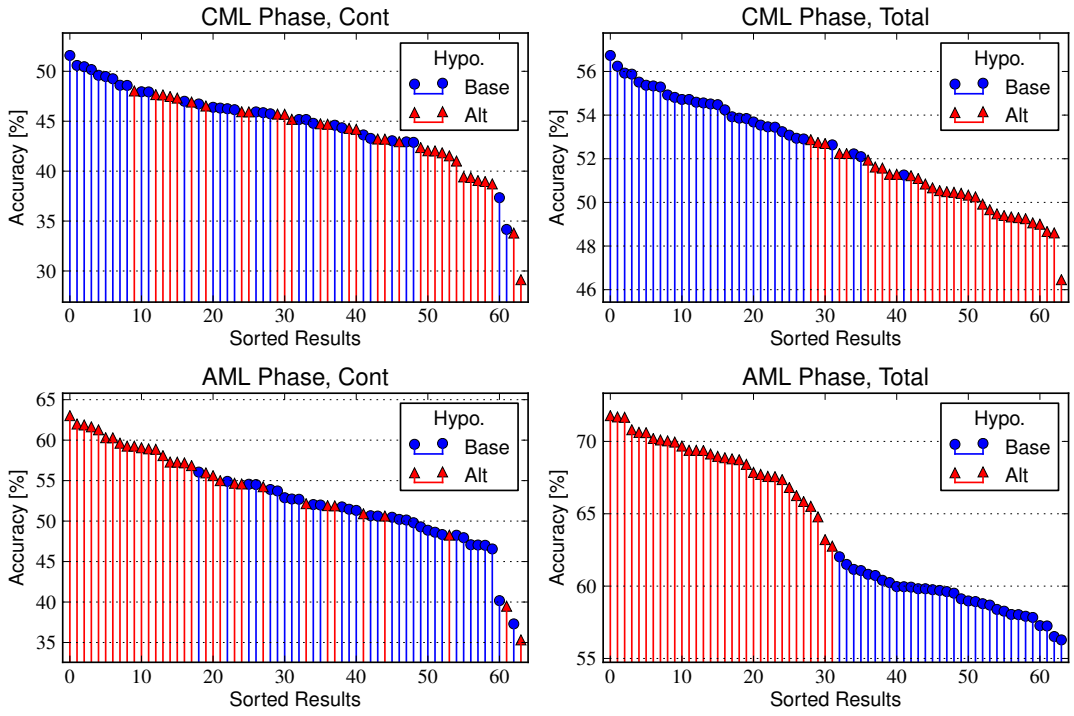

Figure 4.10: *Sorted beat location accuracy results, colored according to whether alternative hypotheses were considered (Alt) or only the base hypothesis was used (Base).*

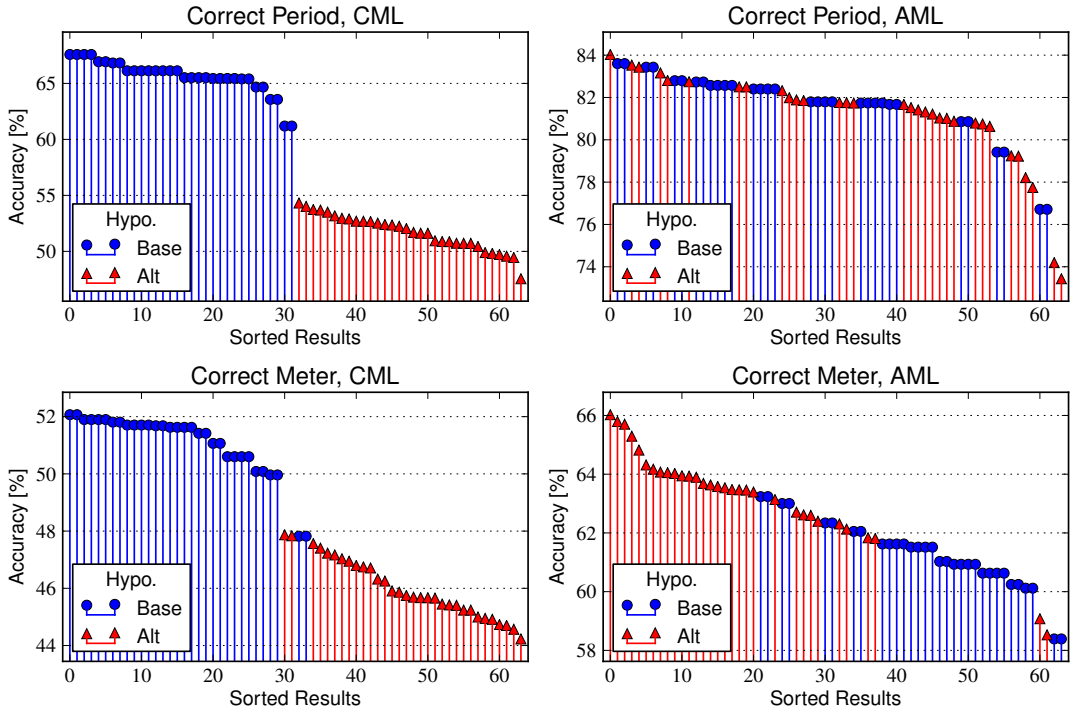

Figure 4.11: *Sorted beat period and meter accuracy results, colored according to whether alternative hypothesis were considered (Alt) or only the base hypothesis was used (Base).*

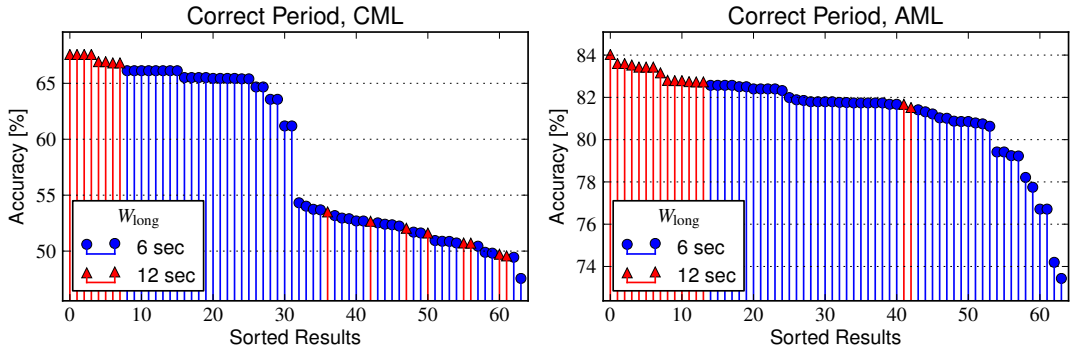

Figure 4.12: *Sorted beat period accuracy results, colored according to the length of the long autocorrelation window used to detect the base period, W_{long}.*

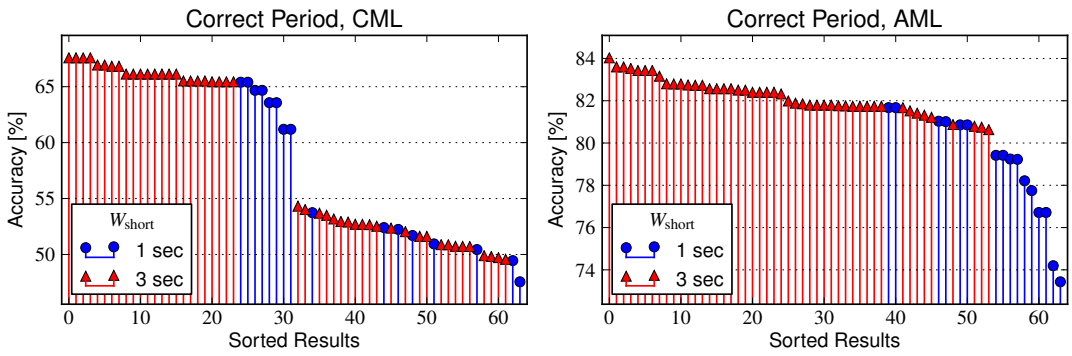

Figure 4.13: *Sorted beat period accuracy results, colored according to the length of the short autocorrelation window used to detect the base period, W_{short}.*

Chapter 5

Analyzing Drum Patterns

5.1 Introduction

So far, I have covered techniques that allow the detection of drum events and the estimation of a metrical grid on which these events lie. Now I will present an approach that allows higher-level analysis of drum patterns, which can be defined as repeating sequences of drum events. Variation is possible from one repetition of a pattern to the next, but common elements remain that define the overall style or feel of the pattern. A single repetition of a drum pattern typically consists of an integer number of beats that define the meter of the pattern and the current meter of the piece of music (though simultaneous polymeter between instruments is possible but uncommon). Figure 4.1 in Section 4.1 illustrates these metrical concepts.

Analyzing rhythms at the measure level is a difficult problem due to the high level of ambiguity present. For example, when trying to decide which beat is the first beat in the measure, a frequently encountered problem is half measure ambiguity. Figure 5.1 illustrates this problem with a basic rock pattern. We know which beat is which because we have the sheet music in front of us; however, if we

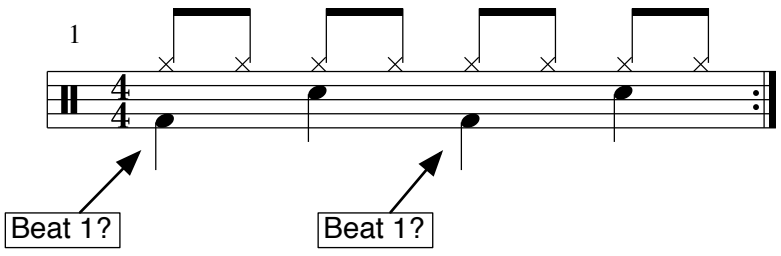

Figure 5.1: *Two possible measure alignment hypotheses. From lowest to highest, the notes shown represent the bass drum, snare drum, and hi-hat, respectively.*

start listening to such a rhythm at an arbitrary point in time, there's nothing to distinguish beat one and beat three. When a rhythm is actually performed, however, there can be many contextual clues that help a listener create an improved hypothesis about where each measure begins. The bass drum and hi-hat may be played slightly louder on beat one versus beat three, or there may have been some sort of drum fill or embellishment during the end of the previous measure to set the table for beat one of the current measure. Writing a set of rules to describe every possible contextual cue is a common approach to improving drum pattern analysis; however, this process can be quite tedious, especially when presented with many genres or styles of rhythm. Most existing approaches to measure alignment use some sort of heuristic rule-based method and forego machine learning techniques altogether. These will be covered in the next section.

My alternative approach uses deep learning techniques to create a domain-specific model of rhythmic information. Deep neural networks have shown promise in many discriminative tasks, such as written digit recognition [43] and acoustic modeling [57][56][18][19]. To demonstrate the effectiveness of this model, I focus on a specific discriminative task, the aforementioned problem of beat-measure alignment; however, this approach can be applied to other drum pattern analysis tasks such as style classification, meter detection, or transition/novelty detection. Section 5.3 serves as an introduction to common techniques used in deep learning, and covers the specific methods I use to analyze drum patterns.

5.2 Previous Work on Downbeat Detection

As mentioned above, previous work on beat-measure alignment has focused on simple heuristic rules. In [48], Klapuri presents a beat tracker that determines beat-measure alignment by cross-correlating multi-band onset patterns with two different rock-based pattern templates. In [32], Goto addresses beat alignment by detecting chord change locations and by alignment with 8 different drum pattern templates. Approaches that use hand-written patterns like these work well for the typical pop song but are ineffective when presented with exotic rhythm styles or odd meters.

The more recent approach by Peeters [62], which improves on the downbeat detection results of Klapuri, uses Latent Discriminant Analysis to automatically learn "beat templates" from spectral and pitch features. The work of Jehan [46] uses a Support Vector Machine (SVM) to classify beats based on recent spectral features. Hockman extends this approach by adding a bass frequency detector to stabilize the estimates and specializes the SVM training for hardcore, jungle, and drum and bass music [45].

Rather than analyzing a complete piece of polyphonic music, my approach focuses on detecting the downbeat from a sequence of drum onsets. This problem

is much simpler in terms of recognizing salient events since the drums are isolated; however, additional musical cues such as chord changes are not present to provide additional context. In addition, my goal is to keep the model completely general and avoid specializing for any single type of music. Deep learning techniques are useful in this regard because they allow us to encode a large amount of musical knowledge into the *distributed state-space* [7] of a Restricted Boltzmann Machine (RBM), which I introduce in the next section.

5.3 Deep Learning

5.3.1 The Restricted Boltzmann Machine

The primary tool driving current approaches to deep learning is the restricted Boltzmann machine (RBM) [44] and variations on the RBM. RBMs are stochastic auto-encoders, and multiple RBMs can be composed to form deep belief networks (DBNs), which are probabilistic multi-layer neural networks [7]. The RBM, as shown in Figure 5.2, is a two layer probabilistic graphical model with undirected connections between visible layer units, v_i, and hidden layer units, h_j. The "restricted" part of the name points to the fact that there are no connections between units in the same layer. This allows the conditional distribution of the units of one layer given all the units of the other layer to be completely factorial, i.e.,

$$\mathrm{p}(\mathbf{v}|\mathbf{h}) = \prod_i \mathrm{p}(v_i|\mathbf{h}) \tag{5.1}$$

$$\mathrm{p}(\mathbf{h}|\mathbf{v}) = \prod_j \mathrm{p}(h_j|\mathbf{v}) \tag{5.2}$$

The RBM is a probabilistic energy-based model, meaning the probability of a specific configuration of the visible and hidden units is proportional to the negative exponentiation of an energy function, $E(\mathbf{v}, \mathbf{h})$

$$\mathrm{p}(\mathbf{v}, \mathbf{h}) = \frac{e^{-E(\mathbf{v},\mathbf{h})}}{Z} \tag{5.3}$$

Where $Z = \sum_{\mathbf{v},\mathbf{h}} \exp(-E(\mathbf{v}, \mathbf{h}))$ is a normalizing constant referred to as the *partition function*. Note that because Z is difficult to compute, it is typically intractable to compute the joint distribution $\mathrm{p}(\mathbf{v}, \mathbf{h})$.

For binary-valued visible and hidden units, the energy function, $E(\mathbf{v}, \mathbf{h})$, can be written as:

$$E(\mathbf{v}, \mathbf{h}) = -\mathbf{a}^{\mathrm{T}}\mathbf{v} - \mathbf{b}^{\mathrm{T}}\mathbf{h} - \mathbf{v}^{\mathrm{T}}W\mathbf{h} \tag{5.4}$$

Where $\mathbf{a}$ and $\mathbf{b}$ are vectors containing the visible and hidden unit biases, respectively, and W is the weight matrix that connects the two layers.

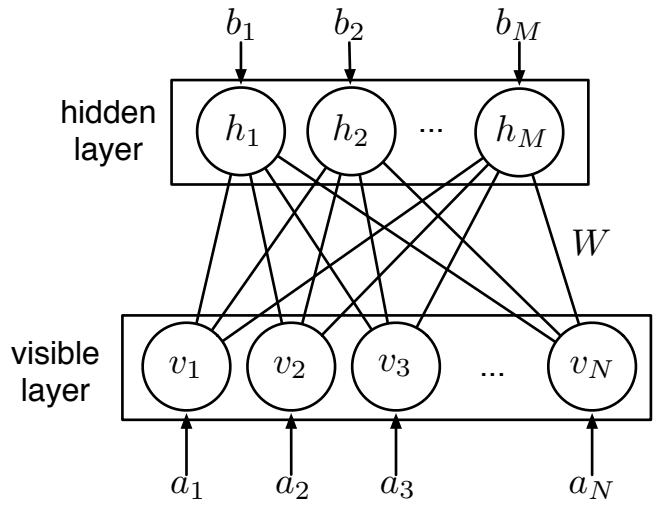

Figure 5.2: *A restricted Boltzmann machine with N visible units and M hidden units.*

The goal in training an RBM is to maximize the likelihood of the training data under the model, $\mathrm{p}(\mathbf{v})$. The actual log-likelihood gradient is difficult to compute because it involves the intractable partition function Z; however, stochastic estimates of the gradient can be made by drawing Gibbs samples from the joint distribution $\mathrm{p}(\mathbf{v}, \mathbf{h})$ using the factorial conditional distributions in (5.5),(5.6).

$$\mathrm{p}(v_i = 1|\mathbf{h}) = \bar{\sigma}(a_i + \textstyle\sum_j W_{ij} h_j) \tag{5.5}$$
$$\mathrm{p}(h_j = 1|\mathbf{v}) = \bar{\sigma}(b_j + \textstyle\sum_i W_{ij} v_i) \tag{5.6}$$

Where $\bar{\sigma}(x)$ is the logistic sigmoid function:

$$\bar{\sigma}(x) = \frac{1}{1 + e^{-x}} \tag{5.7}$$

The Gibbs sampling Markov chain can take quite a long time to produce actual samples from the joint distribution, so in practice the chain is started at a training example and run for a small number of iterations. Using this estimate of the log-likelihood gradient, we are instead minimizing a quantity referred to as the *contrastive divergence* [44, 7]. Contrastive divergence updates for the RBM parameters are shown below:

$$\Delta W_{ij} \propto \langle v_i h_j \rangle_0 - \langle v_i h_j \rangle_k \tag{5.8}$$
$$\Delta a_i \propto \langle v_i \rangle_0 - \langle v_i \rangle_k \tag{5.9}$$
$$\Delta b_j \propto \langle h_j \rangle_0 - \langle h_j \rangle_k \tag{5.10}$$

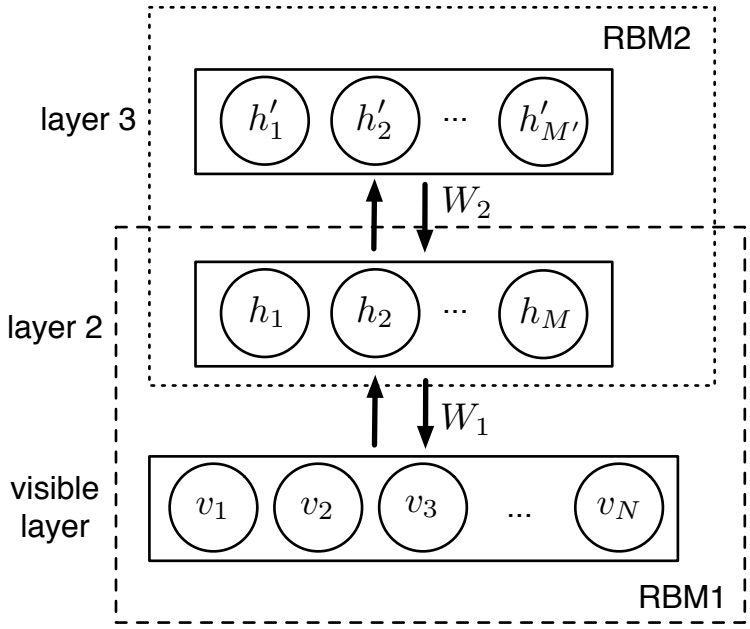

Figure 5.3: *A 3-layer deep belief network comprised of 2 RBMs*

Where $\langle \cdot \rangle_k$ denotes the value of the quantity after k iterations of Gibbs sampling, and for $k = 0$, v_i is simply the training data and h_j is a sample from (5.6) given the training data. Typically, these updates are performed using multiple training examples at a time by averaging over the updates produced by each example. This helps to smooth the learning signal and also helps take advantage of the computational efficiency of larger matrix operations. As $k \to \infty$, these updates approach maximum likelihood learning.

5.3.2 Stacking RBMs

A deep belief network is formed when multiple RBMs are stacked on top of each other as shown in Figure 5.3. After training a first-level RBM using the training data, we can perform a deterministic up-pass by setting the hidden units to their real-valued activation probabilities using (5.6) for each visible training vector. This is the same as what is done in the up-pass in a deterministic logistic neural network. These deterministic hidden unit values are then used as the visible data in a subsequent higher-level RBM, which is also trained using contrastive divergence learning. This RBM stacking continues until the network reaches the desired depth. This greedy layer-by-layer training approach is a useful procedure for learning a set of non-linear features in an unsupervised manner [36], and

it has been shown to be a beneficial pre-training procedure when followed by discriminative backpropagation [43].

5.3.3 The Conditional Restricted Boltzmann Machine

The conditional restricted Boltzmann machine (CRBM) takes the RBM a step further by adding directed connections between additional visible units, y_i, and the existing visible and hidden units, as shown in Figure 5.4. These additional units can represent any type of additional information, including visible data from the recent past. This allows the CRBM to be an effective generative model of time sequence data [72], and this fact is what motivated my use of the CRBM to model drum patterns.

The directed connections, which are represented by weight matrices A and B, replace the bias terms, $\mathbf{a}$ and $\mathbf{b}$ in (5.5),(5.6), with dynamic bias terms, $\hat{\mathbf{a}}$ and $\hat{\mathbf{b}}$.

$$\hat{\mathbf{a}} = \mathbf{a} + A\mathbf{y} \tag{5.11}$$
$$\hat{\mathbf{b}} = \mathbf{b} + B\mathbf{y} \tag{5.12}$$

Where $\mathbf{y}$ is a vector containing the conditioning data. This modified RBM models the distribution $\mathrm{p}(\mathbf{v}, \mathbf{h}|\mathbf{y})$, and the learning rules in (5.8)–(5.10) are unchanged except for the addition of the dynamic bias terms to the sampling expressions. The learning rules for the conditional weight matrices also have a familiar form:

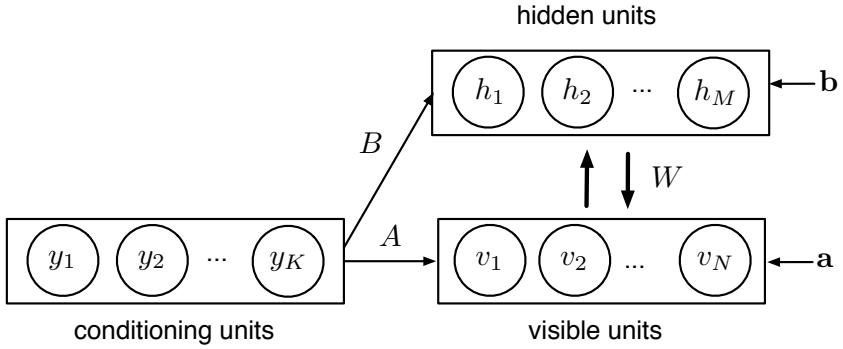

Figure 5.4: *A conditional restricted Boltzmann machine.*

$$\Delta A_{ij} \propto \langle v_i y_j \rangle_0 - \langle v_i y_j \rangle_k \tag{5.13}$$
$$\Delta B_{ij} \propto \langle h_i y_j \rangle_0 - \langle h_i y_j \rangle_k \tag{5.14}$$

Note that the y_j above are simply the training values and are not stochastically sampled in any way.

5.4 Modeling and Analyzing Drum Patterns

5.4.1 Bounded Linear Units

Drum patterns are not simply a series of ones and zeros, onset or no onset. Most drum patterns contain an appreciable sonic difference between accented and unaccented notes on every drum or cymbal, and it is these differences which give drum patterns their character. In order to effectively model drum patterns using the CRBM, we must modify the binary-valued visible units to be real-valued.

There are many options for getting real-valued visible activations out of RBMs; in fact, it has been shown that every distribution in the exponential family is a viable candidate [74]. A popular choice is the Gaussian distribution due to its familiarity and ubiquity; however, the unboundedness of Gaussian samples does not translate well to the space of dynamic levels possible within a drum pattern.

In order to model the bounded nature of playing dynamics, I use a modified version of the *rectified linear units* (RLUs) described in [58]. RLUs are constructed from a series of binary units with identical inputs but with fixed, increasing bias offsets. If the bias offsets are chosen appropriately, the expected value and variance of the number of active units out of these N tied binary units with common input x is:

$$\mathrm{E}[v|x] = \log(1+e^{x}) - \log(1+e^{x-N}) \tag{5.15}$$

$$\mathrm{Var}(v|x) = \bar{\sigma}(x) - \bar{\sigma}(x-N) \tag{5.16}$$

As can be seen in Figure 5.5, (5.15) yields a sigmoidal curve that saturates when $x > N$, bottoms out when $x < 0$, and is approximately linear in between. In the linear region, the variance is equal to one, so the value of N is chosen to achieve the desired level of noisiness in the samples, and the training data can be rescaled to the interval $[0, N]$. In this work, I have chosen $N = 20$, so that a value of 20 represents the loudest possible drum strike, while zero represents the absence of a drum strike. To sample from these *bounded linear units* (BLUs), instead of actually sampling from N binary units with stepped offsets, I approximate their total output with:

$$\mathrm{p}(v|x) \sim \Big[\mathcal{N}\big(\mathrm{E}[v|x], \mathrm{Var}(v|x)\big)\Big]_0^N \tag{5.17}$$

where $\mathcal{N}(\cdot)$ is a normal distribution with mean and variance provided by (5.15) and (5.16), and $\big[\,\cdot\,\big]_0^N$ snaps values outside of the interval $[0, N]$ to the boundaries of the interval. Because these BLUs are constructed from logistic binary units, all of the RBM learning rules from Section 5.3 are still valid; the only thing that changes is how we sample from the visible BLUs.

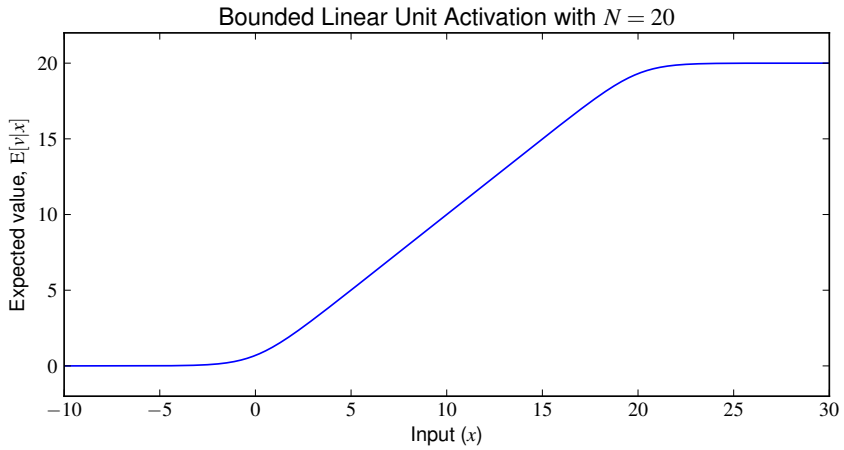

Figure 5.5: *Bounded linear unit activation function*

5.4.2 Label Units

If bounded linear units give us a way to get drum onset information into the network, label units are how we get information out of the network. A standard approach to multi-class classification with neural networks is to use a group of softmax output units, which assigns a value to each of its units (each with input x_i) based on the softmax activation function shown in (5.18). This activation function is convenient for classification because the activation values of the group sum to one, which allows the output values to be interpreted as posterior class probabilities given the input data.

$$\mathcal{S}_{\max}(x_i, \mathbf{x}) = \frac{e^{x_i}}{\sum_j e^{x_j}} \tag{5.18}$$

In the realm of RBMs and deep learning, a different approach can be used which entails providing the ground truth class labels as part of the visible data during training. This approach has been shown to be more effective than using a separate softmax output layer in certain cases [43], and it indeed achieves better results for this application. Instead of adding the label units to a separate output layer, I augment the visible layer in the top-level RBM of a deep belief network with a group of softmax label units, as shown in Figure 5.6. This allows us to train the top-level RBM using the label units as visible data, by turning on only the correct label unit during training. Once this labeled RBM has been trained, we can assign a posterior probability to each of the label units given the data by computing the *free energy*, $\mathcal{F}(\mathbf{v}, l)$, of the RBM for each possible label configuration and taking

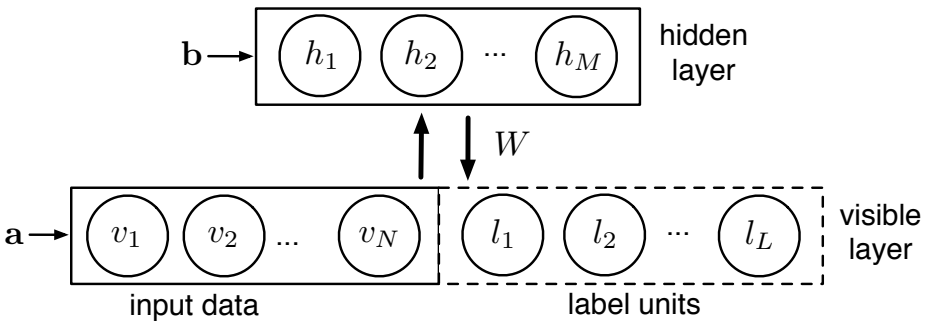

Figure 5.6: *An RBM with an added group of visible label units.*

the softmax (see [44]):

$$\begin{aligned}\mathcal{F}(\mathbf{v}, l) &= -\log \sum_{\mathbf{h}} e^{-E(\mathbf{v}^{(l)}, \mathbf{h})} \\ &= -\sum_i v_i^{(l)} a_i - \sum_j \log\left(1 + \exp\left(x_j^{(l)}\right)\right) \qquad (5.19)\end{aligned}$$

$$\mathrm{p}(l|\mathbf{v}) = \frac{e^{-\mathcal{F}(\mathbf{v},l)}}{\sum_k e^{-\mathcal{F}(\mathbf{v},k)}} \qquad (5.20)$$

Where $x_j^{(l)} = b_j + \sum_i W_{ij} v_i^{(l)}$, and $v_i^{(l)}$ denotes the visible data but with only unit l of the label unit group activated. The term $E(\mathbf{v}^{(l)}, \mathbf{h})$ is the energy function from eq. (5.4). This calculation is tractable due to the typically small number of label units being evaluated.

5.4.3 Modeling Drum Patterns

In this drum pattern analysis network, the first layer is always pre-trained as a CRBM. This CRBM models the current drum beat or subdivision using one BLU visible unit per drum or cymbal. In my experiments, I use a minimal three-drum setup: bass drum, snare drum, and hi-hat, but this can be expanded to work with any percussion setup. The conditioning units, y_j, of the CRBM contain drum activations from the recent past. In my experiments, $\mathbf{y}$ is fed with drum activations from the most recent two measures (or 32 subdivisions given a 4/4 time signature with sixteenth note subdivisions).

Subsequent layers use binary visible units instead of BLUs. Intermediate layers of the DBN can be made up of either additional CRBMs or standard RBMs, and the final layer must have visible label units to represent the classifier output. Using an intermediate-layer CRBM allows the layer to take into account past hidden unit activations of the layer below it, which allows it to learn higher-level

time dependencies. In doing so, it increases the past time context that the top-level layer sees, since the past hidden unit activations of a first-level CRBM have been conditioned on the past relative to themselves. In order to make a fair comparison between DBNs that use different numbers of CRBM layers, we must make sure that the top layer always has access to the same amount of visible first-layer data from the past.

In the experiments, I train the label units to detect the current sixteenth note beat subdivision within the current 4/4 measure. In the next section, I give details on the configuration and training of the various DBNs that I test for this task.

5.5 Training the System

5.5.1 Training Data

The dataset consists of 173 twelve-measure sequences comprising a total of 33,216 beat subdivisions, each of which contains bass drum, snare drum, and hi-hat activations. The data was collected using electronic Roland V-drums[1], quantized to exact sixteenth note subdivisions, and converted to a subdivision-synchronous drum activation matrix.

The sequences were intended to span a sizeable, but by no means complete, collection of popular rhythmic styles. There is a strong rock bias, with many beats featuring a prominent back beat[2]; however, also included are more syncopated styles such as funk and drum 'n' bass as well as the Brazilian styles samba and bossa nova. I use three-fold cross-validation to evaluate each network configuration, with a 33%,57%,10% split for test, training, and validation data, respectively.

5.5.2 Network Configurations

I test four network configurations each consisting of around 90,000 parameters. Although this may seem like quite a few parameters for such a small dataset, these configurations produced better results than versions with less parameters. For each of the four network architectures, I tested multiple hidden unit configurations and have chosen to present only those which performed best on the test data for each architecture. They are as follows:

1. *Labeled-CRBM (3 layers)*
   3 visible data units + 16 visible label units, 800 hidden units, and 32 past subdivisions of context (96 conditioning units)

[1] `http://rolandus.com`

[2] snare hits on beats 2 and 4 in a 4/4 measure

2. *CRBM → Labeled-RBM (4 layers)*
   CRBM with 600 hidden units, LRBM with 50 hidden units. The CRBM again has a context of 32 subdivisions.

3. *CRBM → Labeled-CRBM (4 layers)*
   With 200 and 50 hidden units respectively. Each CRBM has a context of 16.

4. *CRBM → CRBM → Labeled-RBM (5 layers)*
   With 200, 25, and 25 hidden units respectively. Both CRBMs have a context of 16 subdivisions.

5.5.3 Network Training

Pre-Training

Each non-labeled RBM was pre-trained using contrastive divergence with $k = 1$ (CD-1) for 300 sweeps through the training data (300 epochs). Updates were performed using mini-batch gradient descent with a mini-batch size of 100 training examples. The order of the training data was randomized before training and after every 100 epochs in order to smooth the learning.

Top-level labeled layers were pre-trained with the correct visible label unit switched on and the other label units switched off. I pre-trained each labeled layer using CD with $k = 1$ for 150 epochs and then k was linearly increased from 1 to 10 for an additional 150 epochs.

For all pre-training, the learning rate was initialized to 10^{-4} and then slowly adjusted during training so that the average weight update magnitude was on the order of 10^{-3} times the average weight parameter magnitude. As suggested by Taylor [72], CRBM autoregressive weights (A in Figure 5.4 and eq. (5.13)) used a learning rate 1/10 of the above dynamic learning rate. To prevent the network from simply learning to duplicate drum activations from the previous measure, I only used autoregressive weights connecting the previous 4 subdivisions in first level CRBMs and connecting only the previous subdivision in second level CRBMs. We can think of these short-term autoregressive weights as encoding what is physically possible for a drummer to play, whereas the weights connecting the past to the hidden units (B) encode the drummer's memory of what has been recently played and his intentions for the current subdivision. During the last 30 epochs, the actual learning rate was forced to decay to 1/10 of the dynamic learning rate. After the first 15 epochs, a momentum term of 0.9 was gradually introduced in order to speed up the learning [42]. To prevent any of the weights from becoming excessively large, an L2 weight-decay constraint of 0.01 was used. Additional information on practical considerations for training restricted Boltzmann machine can be found in [42].

Fine-Tuning

In pre-training, the label units in the top-level RBM are provided as part of the visible data; however during fine-tuning of the entire network, they are decoupled from the rest of the input units of the RBM and become the output units. This can be visualized as swinging the label units from the bottom to the top as shown in Figure 5.7. In doing so, the weights W are split into matrices W' (for the input–hidden connections) and R (for the hidden–output connections). Also, the original biases $\mathbf{a}$ become $\mathbf{a}'$ and $\mathbf{c}$. The network can be discriminatively fine-

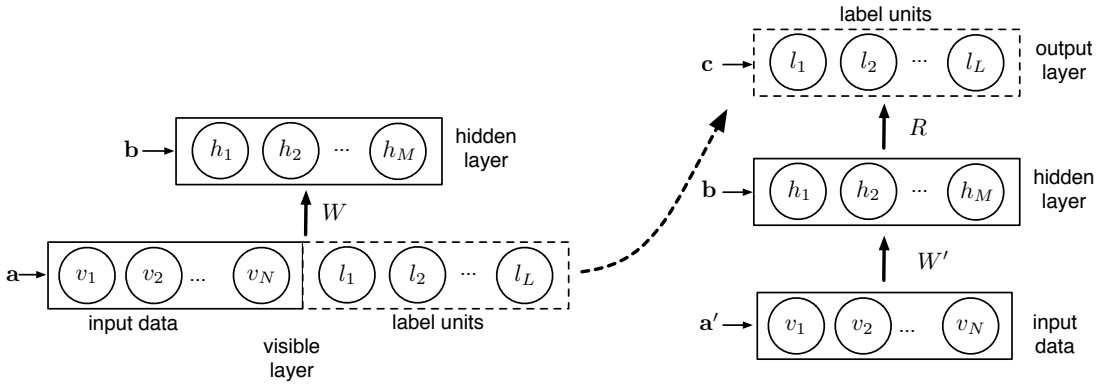

Figure 5.7: *In the top-level RBM, the label units are decoupled from the rest of the input units for backpropagation.*

tuned by backpropagating the cross-entropy label unit error to the logistic units in lower layers. Below the top level, this is identical to traditional logistic backpropagation; however, the unconventional free-energy-based output activations of the label units (eq. (5.20)) require different updates.

The cross-entropy error, H, is defined as.

$$\mathrm{H}(\mathbf{t}, \mathbf{a}) = -\sum_i t_i \log o_i \tag{5.21}$$

Where t_i is the target value of output unit i and o_i is the actual output activation. Since the target output for the label probabilities is 1 for the correct label and 0 for the other labels, H simplifies to:

$$\mathrm{H}(l = l^*, \mathbf{v}) = -\log \mathrm{p}(l = l^* | \mathbf{v}) \tag{5.22}$$

Where l^* is the correct label. Therefore, minimizing H is equivalent to maximizing the log-probability of the correct label.

The gradient updates for W', R, $\mathbf{b}$, and $\mathbf{c}$ used to minimize H are computed as follows. Let $\xi_d(\mathbf{v})$ be the label probability error of label unit d given correct

label l^*.

$$\xi_d(\mathbf{v}) = \begin{cases} 1 - \mathrm{p}(l = d|\mathbf{v}), & d = l^* \\ 0 - \mathrm{p}(l = d|\mathbf{v}), & d \neq l^* \end{cases} \tag{5.23}$$

Let $h_j^d(\mathbf{v})$ be the real-valued activation of hidden unit j given that only label unit d is turned on and the input data is $\mathbf{v}$.

$$h_j^d(\mathbf{v}) = \bar{\sigma}\left(b_j + \sum_i v_i W_{ij} + R_{dj}\right) \tag{5.24}$$

The gradient-based learning rules are then:

$$\Delta W_{ij} \propto v_i \sum_d \xi_d(\mathbf{v}) h_j^d(\mathbf{v}) \tag{5.25}$$

$$\Delta R_{dj} \propto \xi_d(\mathbf{v}) h_j^d(\mathbf{v}) \tag{5.26}$$

$$\Delta b_j \propto \sum_d \xi_d(\mathbf{v}) h_j^d(\mathbf{v}) \tag{5.27}$$

$$\Delta c_d \propto \xi_d(\mathbf{v}) \tag{5.28}$$

In order to backpropagate the error to lower levels, the gradient of the cost function with respect to the input units is required.

$$\frac{\partial \mathrm{H}(l = l^*, \mathbf{v})}{\partial v_i} = -\sum_d \left(\xi_d(\mathbf{v}) \left[a_i' + \sum_j h_j^d(\mathbf{v}) W_{ij}\right]\right) \tag{5.29}$$

Using the above, backpropagation can be continued to lower levels as usual [11].

As in the layer-wise pre-training, mini-batch updates with a batch size of 100 were used during fine-tuning. The learning rate was dynamically adjusted but a smaller target magnitude ratio of 5×10^{-5} was used. The update momentum was the same as during pre-training and an L2 weight decay of 0.001 was used. During backpropagation, the parameter set that produced the lowest cross-entropy error on the validation set up to that point in the training was remembered. As soon as the validation error began to increase, the learning rate was slowly reduced to 10% of the dynamically adjusted value over a period of 30 epochs. At this point, the training was halted, and the remembered minimum validation error parameters were used as the final model parameters. This *early stopping* training procedure typically resulted in 250–500 epochs of backprogation being performed. I also experimented with using a set number of training epochs (500). This produced very similar but slightly worse results (with longer training time) compared to the early stopping procedure.

Implementation

Neural network training can be extremely computationally intensive, so it is very important to pay attention to execution speed and efficiency since this could mean the difference between hours and weeks of training time. Training relies heavily on multiplying large matrices, which can be done considerably faster using highly data-parallel graphics processing units (GPUs). In fact, I believe the advent of general-purpose GPU computing has played a large part in the resurgence of neural network research. I use Gnumpy [73], a Python module which provides Numpy-like[3] bindings for matrix operations on Nvidia GPUs. Modules such as Gnumpy allow much faster execution while retaining the programmer productivity of high-level scripting languages such as Python and Matlab. In this case, it results in much faster overall application development and experiment turnaround. Using an Nvidia Tesla C2050 GPU, training the LCRBM model (#1, 3-layers) took an average of around 20 minutes, while the CRBM-CRBM-LRBM (#4, 5-layers) took around 30 minutes. The typical split between pre-training time and backpropagation time was around 60%/40%.

5.5.4 HMM Filtering

In addition to simply classifying each subdivision individually, we can take into account additional sequential context by providing the label probabilities as posterior state probabilities in a hidden Markov model (HMM) [64]. In order to maximize coherence between successive beat subdivision estimates, I assign a high probability of a transition to the next sequential beat and give an equal division of the remaining probability to other transitions. Since the system is designed for live use, I use a strictly causal version of the Viterbi algorithm to estimate the current beat subdivision.

5.6 Downbeat Detection Results

5.6.1 Classifying Subdivisions

Here I present the classification results for beat-measure alignment, as well as half note and quarter note alignment. The test data contains 16 beat subdivisions per 4/4 measure, so 16 separate label units are used in the training.

In order to justify the large number of parameters used in these networks, Figure 5.8 shows beat labeling performance for a single-level labeled CRBM network as the number of hidden units is increased. Accuracy tops out around 80% and doesn't improve significantly beyond 800 hidden units. This 800 hidden unit

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

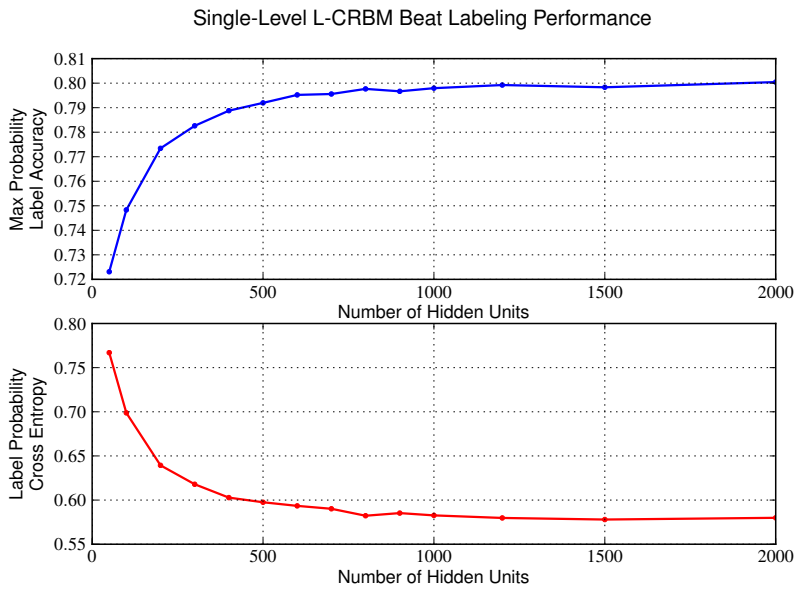

Figure 5.8: *Measure alignment accuracy for a single-level LCRBM model as the number of hidden units is increased.*

network has around 90,000 parameters, so I set this as the equalizer with which to compare different models.

In the results that follow, beat labeling accuracy is shown for the four models described in Section 5.5.2. As a baseline, results from a 2-template cross-correlation approach are shown for comparison. The two templates used are adapted for use with drums from the band-wise templates used by Klapuri in [48]. Figure 5.9 shows measure alignment (downbeat detection) results produced using the maximum likelihood beat labels produced by each model. All four of the CDBN-based models shown significant improvement over the 2-template correlation, with each one achieving around 80% accuracy. The average test label probability cross-entropy for each model is shown in the bottom of Figure 5.9 (smaller is better). For both label accuracy and cross-entropy, the model using two stacked CRBMs achieves the best results, though only by a slight margin. A much larger data set is needed to adequately evaluate the trade-offs in these networks.

Example label probabilities for each model are shown in Figure 5.10, where the horizontal axis represent time in beat subdivisions, and the columns of each matrix hold the estimated label probabilities for the current subdivision. Ideal (ground-truth) output appears as periodic diagonal bands proceeding sequentially through each measure, as shown in the bottom matrix. The normalized 2-template correlation exhibits wildly varying output, although some structure is present.

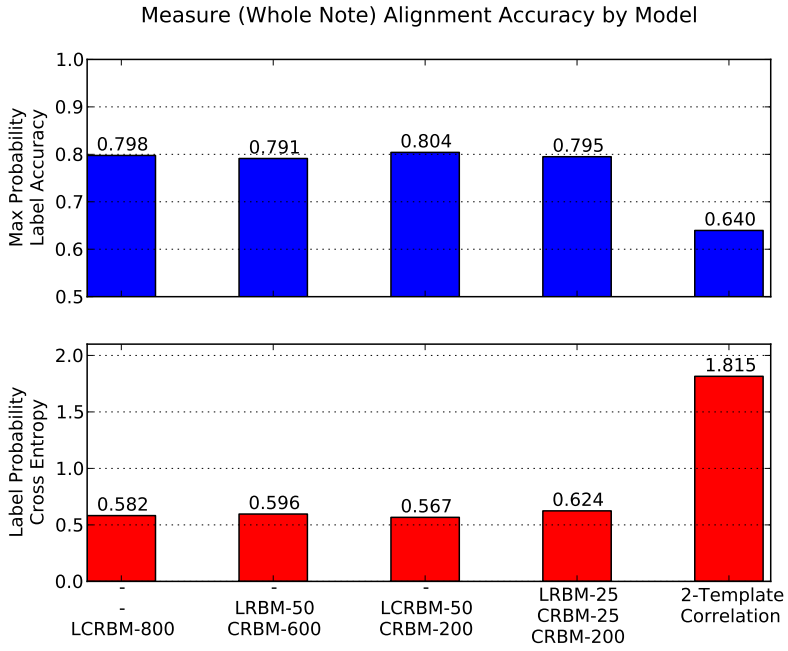

Figure 5.9: *Measure alignment accuracy for four CDBN models with ∼90,000 parameters each compared to a two-template correlation approach.*

Each of the CDBN-based models have difficulty toward the middle of the example sequence of beats, with the two lower models possibly showing slightly less ambiguity.

Figures 5.11–5.12 show results for quarter note and half note alignment, respectively. Separate models were not trained to predict these alternative granularities of alignment. Instead, the probabilities for the original beat labels were summed according to which alignment they correspond to given the target alignment period. For example, if there are 16 subdivisions in a measure, subdivisions 1, 5, 9, and 13 all correspond to the same alignment on a quarter note grid (mod 4 invariance).

In Chapter 4, pg. 42, it is mentioned that halving the beat period to track eighth notes in order to reduce beat alignment ambiguity was not a problem due to the fact that correct quarter note alignment could ultimately be computed using techniques from this chapter. Figure 5.11 shows that quarter note alignment can be estimated with around 96% accuracy, which is a significant improvement compared to the template correlation method. In addition, the cross-entropy of the output probabilities is significantly better indicating the probability assigned to the correct labels is much closer to one. This improved result is important when using HMM filtering to enhance the accuracy of the predicted labels, as is shown next.

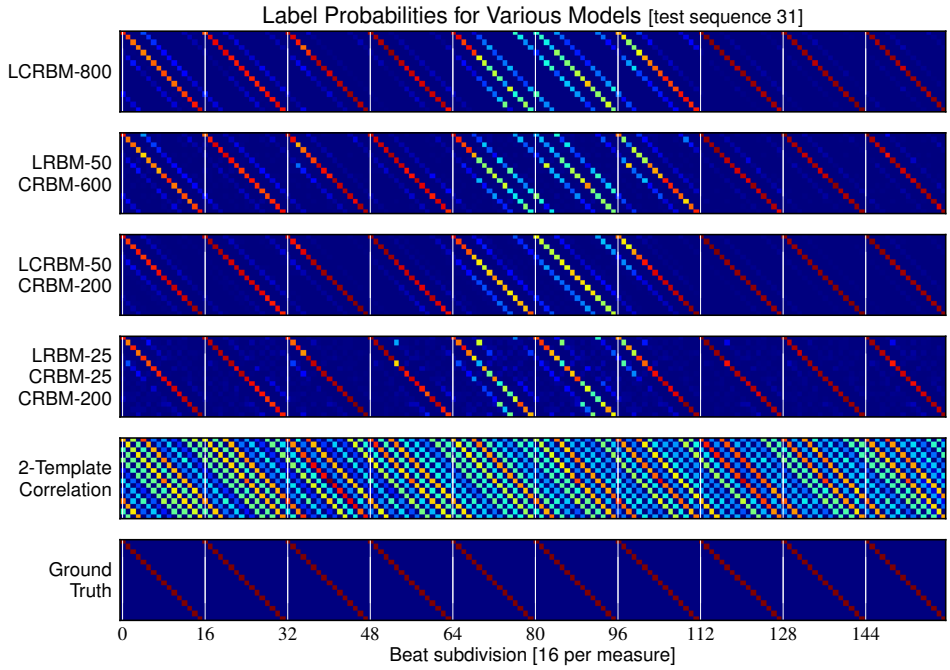

Figure 5.10: *Example posterior subdivision probabilities from the four models and the ground truth labels. The columns in each matrix show the posterior probability of each label for a particular beat subdivision. Vertical white lines indicate measure boundaries.*

5.6.2 Using HMM Filtering

Now I will present the classification results when using the Viterbi algorithm to compute the most likely partial path end-state at each point in time. I was concerned there would be a tendency for the high sequential state transition probability to increase the number of classification errors in the presence of half-note offset ambiguities; however, the HMM filtering only seemed to help classification on average.

As shown in Figure 5.13, increasing the sequential transition probability increases the overall beat classification accuracy; however, in a real-world application, this probability must be kept below 1.0 in order to allow for incorrect beat tracking or beats which are purposely omitted by the drummer. Therefore, this parameter should be set with the end use case in mind.

When looking at each curve presented in Figure 5.13, we can make a connection between the subdivision label cross-entropy of a model (shown in Figure 5.9) and the improvement seen in the HMM curves. For example, the fourth model containing 3 levels of RBM has the highest label cross-entropy and lags behind the

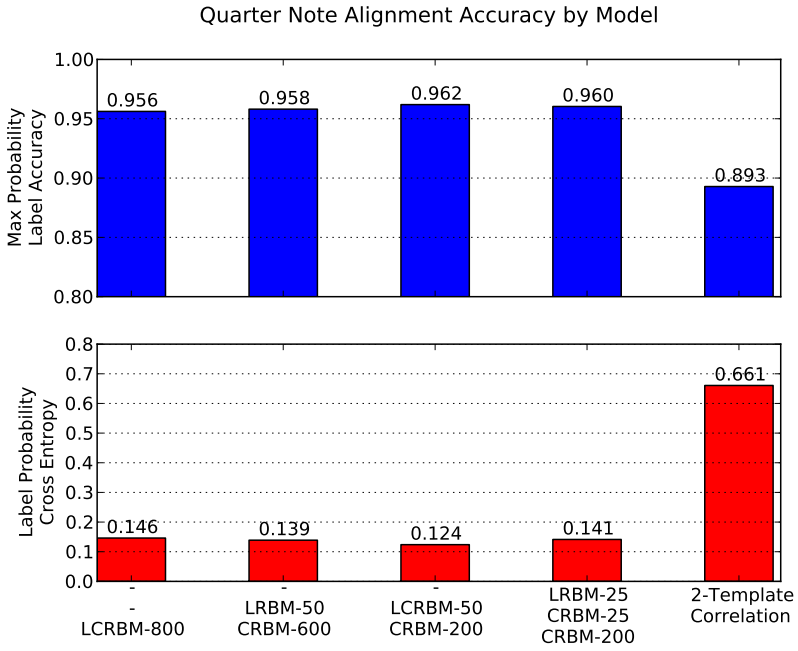

Figure 5.11: *Measure alignment accuracy for four CDBN models with ∼90,000 parameters each compared to a two-template correlation approach.*

pack in Figure 5.13 despite the fact that it did not start with the lowest unfiltered accuracy. This shows that increasing the certainty with which independent label estimates are made (decreasing the cross-entropy) improves the HMM-filtered labeling accuracy, even if it doesn't improve the unfiltered accuracy. Example HMM-filtered results are shown in Figure 5.14.

5.7 Discussion of Results

The presented results show that a neural network based on the conditional restricted Boltzmann machine is a promising model with which to analyze drum patterns. This model achieves a significant improvement in measure alignment performance compared to the standard template correlation approach. However, the results do not point to an optimal network configuration for this task, as it is likely that this will be highly dependent on the diversity and complexity of the drum patterns in the dataset. A much larger dataset will be needed to make any more conclusive judgments in this regard.

There was a significant gap between training set accuracy ($\sim 95\%$) and test set accuracy ($\sim 80\%$) after backpropagation for all models. The training/test accuracy after pre-training but before backprogation was much more even (around

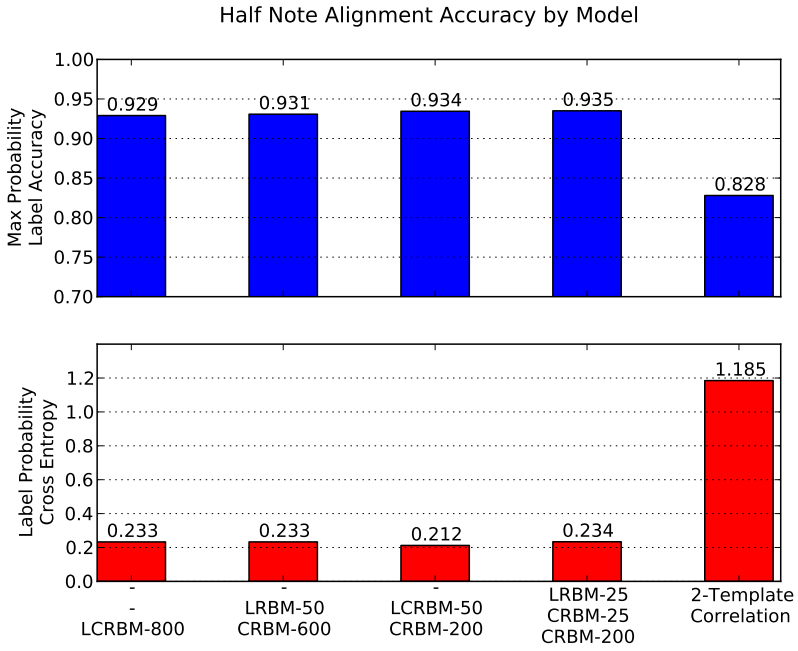

Figure 5.12: *Measure alignment accuracy for four CDBN models with ∼90,000 parameters each compared to a two-template correlation approach.*

50% for both). This suggests the regularization provided by the stochastic RBM training was quite effective at preventing overfitting, and that incorporating additional regularization techniques into the backpropagation phase would improve test accuracy. Using a larger training set can help to prevent overfitting at the expense of longer training time, but when additional labeled training is not available, this is not an option.

A network-wide stochastic generative training procedure similar to that used in RBM pre-training can be used to fine-tune the entire network as a whole [44][43]. Though this approach is much more computationally demanding than simple backpropagation, the level of regularization it achieves can greatly improve test accuracy. There is also the option of combining generative updates with backpropagation in a hybrid fine-tuning scheme which has the potential to improve generalization over standard discriminative backpropagation.

Another approach to regularization is the use of "bottleneck" features [57], which restrict the number of hidden units in certain layers to a small number. This forces the network to learn a compact distributed representation of the input data that only retains the aspects most important for classification. This compact encoding of the input data not only prevents overfitting to noise in the training data, but also reduces the number of weights required in subsequent levels, which provides additional protection against overfitting and can significantly speed up

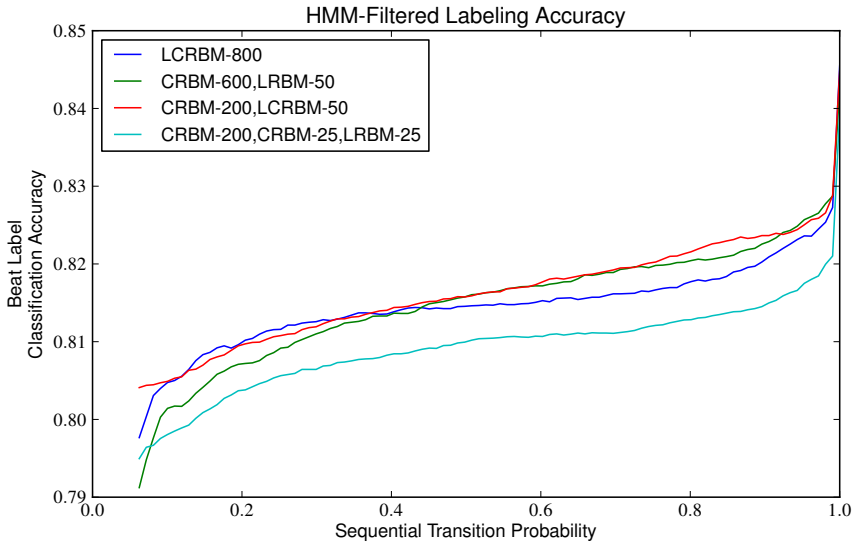

Figure 5.13: *HMM-filtered classification accuracy with increasing sequential transition probability*

learning.

Because the distributed representations learned by deep belief networks allow for a very large state-space to be spanned, it is conceivable that a similarly constructed model could be used to effectively analyze a very large array of rhythmic styles and genres, avoiding the need for models that are specialized for a single specific genre.

A final regularization technique worth mentioning is the "dropout" procedure that has recently shown promise in training a very deep convolutional object recognition network [49]. This technique involves disabling randomly chosen hidden units during training, thereby forcing the network to learn robust features that do not depend on the presence of other features. The use of dropout in the above mentioned object recognition network has produced results that significantly surpass the previous state-of-the-art.

Although I do not objectively evaluate the use of these models for generating drum patterns, it is important to note that because the CRBM is inherently a generative model, these networks are especially well-suited to serve as stochastic drum machines. Even a single labeled-CRBM works well for this purpose, and turning on the label unit of the desired subdivision during Gibbs sampling helps increase the metric stability of the generated patterns.

Overall, this generatively pre-trained deep model model has significant potential to be of use in many music information retrieval and computer music tasks. In addition to exploring the regularization techniques mentioned above, future work

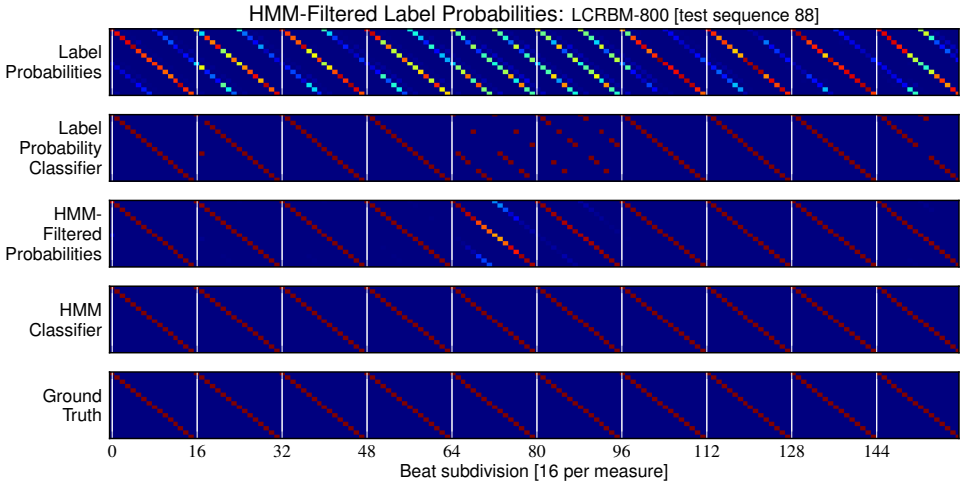

Figure 5.14: *Row 1: Example posterior probabilities. Row 2: Posterior classifications. Row 3: HMM-filtered classifications. Row 4: Ground truth labels. Vertical white lines indicate measure boundaries. Sequential transition probability is set to 85%.*

should explore the ability of the model to discriminate between rhythmic styles, discern between different time signatures, or to detect rhythmic transitions or fills. It would also be interesting to do a more in-depth evaluation of the generative abilities of the model.

Chapter 6

Discussion

The previous four chapters have detailed each of the components of the drum understanding system shown in Figure 6.1. In this last chapter, I cover practical considerations that must be addressed when implementing and integrating the components of such a system. Then I will conclude with a summary of the contributions and future directions for this work.

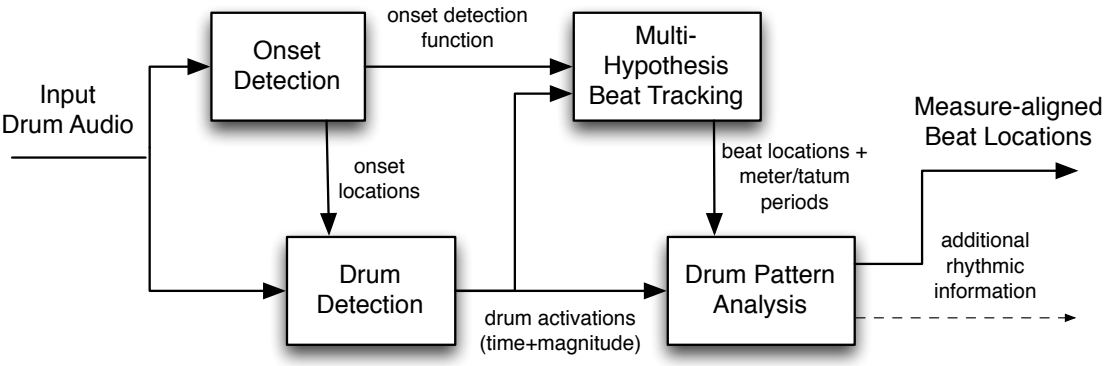

Figure 6.1: *The components of the drum understanding system covered in the preceding four chapters.*

6.1 Practical Considerations

6.1.1 Computation

The implementations used to evaluate the presented techniques were all written in Python[1], making heavy use of the Numpy[2] and Scipy[3] numerical modules for things like FFTs, matrix multiplications, and other array processing primitives. Because the functions available in these modules are implemented in C, the performance is more than acceptable for prototyping and testing in most cases. However, training the deep neural network used in drum pattern recognition did require the use of an additional module, Gnumpy [73], to move the computation from the CPU to a highly parallel GPU. High efficiency GPU implementations have been very important to the progress of neural network research in recent years [42][57][49].

After training, efficient offline analysis can be accomplished by batching up many frames of audio and sending them to a GPU, multi-core system, or computing cluster for processing. In a real-time performance scenario (after training), the computational needs change quite a bit. The most significant bit of synchronous computation is the FFT used by the onset and drum detectors. A 1024-length FFT is computed every 5.8ms (172Hz), giving on the order of 10^6–10^7 floating point operations (flops) per second (using $1.5N\log_2(N) \approx 1.5 \times 10^4$ flops/FFT with $N = 1024$ as an estimate [29]).

When onsets are detected, the drum detector decomposes a 1440-element spectral slice onto around 10–80 drum templates. The 100 or so NVD iterations needed for the decomposition require around 10^7 matrix multiplication flops per detected onset.

In the beat tracker, the most significant chunk of computation is the multi-hypothesis pulse tracking which needs around 10^5–10^6 flops per hypothesis per second to make beat estimates. Adding Fourier interpolation to increase period tracking resolution can drastically increase the amount of computation depending on the level of interpolation.

The pattern analysis models that were evaluated all had around 90,000 multiplicative weights, each of which requires one multiply-add in a feed-forward up-pass. At 120 BPM, there are 8 sixteenth note subdivisions per second that need to be evaluated. This only requires around 10^6 flops per second.

Of course, these computational requirements are only estimates and the actual execution time can vary widely with changes to the CPU or memory hierarchy or when using different libraries to compute key routines such as FFT, matrix multiply, or the exponential function. Especially for real-time execution, it is

[1] `http://www.python.org`
[2] `http://numpy.scipy.org`
[3] `http://www.scipy.org`

important to make sure that the worst-case execution time of each piece of the system is well understood, so that every processing deadline can be met. However, the above estimates show that as long as reasonably efficient implementations are used, real-time execution is completely feasible given that current multi-core CPUs are capable of executing well over 10^{10} flops per second.

6.1.2 Scheduling

In an implementation designed for real-world use in a musical performance environment, it is unlikely that a Python implementation will be able to meet the real-time needs of the application. The compiled C modules used by Python can be plenty efficient; however, the Python interpreter, which is responsible for running each piece of C code, must also perform maintenance tasks like garbage collection, which can take an unbounded amount time, thereby seriously degrading real-time quality of service. In addition, Python has very limited support for multi-thread execution. For these reasons, a fully compiled implementation in an efficient language like C/C++ is strongly indicated. Alternatively, this system could serve as a compiled plug-in in an audio programming environment like Max/MSP[4] or in a digital audio sequencer like Ableton Live[5].

Since the tasks executed by the system occur at multiple rates, special scheduling considerations need to be taken for the different task granularities. The finest-granularity tasks occur at the frame rate (every 5.8ms). These include the onset detector, buffering of drum detector spectral slices, and the computation required by period and phase trackers used by each hypothesis of the beat tracker. Depending on the use case, these tasks have rather lax deadlines since the beat tracker places beat predictions on an output queue one beat period in advance. Therefore, the overall system output can be delayed by a hundred or so milliseconds (depending on the tempo) and still output a beat prediction on time.

Some tasks occur much less frequently than the frame rate and could be scheduled on separate lower-priority threads in order to maintain more uniform processor loading. For example, the drum detector waits for an onset to occur and then waits 67ms for the spectral slice buffer to be filled. It then runs NVD iterations until convergence, which could take significantly longer than the frame-synchronous tasks. Since the beat tracker expects drum-wise activations as input, the drum detector thread must be promoted to high priority (by blocking the dependent tasks) if it is not finished when its output is required.

Base period inference and meter/tatum inference used by the beat tracker occur only a couple times per second, and no parts of the system require their output by a specific deadline.They can be scheduled on a thread with lower priority

[4] `http://www.cycling74.com`

[5] `http://www.ableton.com`

than the drum detector. The pattern analyzer network, when used for measure alignment, runs once per tatum subdivision and its output would be required during every predicted beat.

The primary timing requirement is that beat predictions are made on time, so timing constraints are much less strict than in cases where audio output is being produced in a frame-synchronous manner.

6.2 Contributions

Happily there were many positive results reported in the preceding chapters. My hope is that these techniques will be improved upon in future work by others and eventually adapted for use in real-world systems. Here is a summary of the key contributions.

The approach to onset detection presented in Chapter 2 used a multi-band detection function along with a causal dynamic threshold that is robust to abrupt changes in dynamics. Overall, the onset detector achieves a detection recall of 94.1% and a precision of 93.9% over 8022 drum onsets played at varying dynamic levels.

Chapter 3 presented a gamma mixture model soft clustering procedure that learns multiple spectral templates for each drum and chooses the optimal number of templates automatically. Use of the gamma distribution allows clustering using the perceptual Itakura-Saito distance, as opposed to the Euclidean distance used by a Gaussian mixture model. In addition, this gamma-based clustering procedure eliminates the expensive and potentially unstable covariance matrix computation required by the oft-used Gaussian mixture model. I also introduced "tail" templates, which model a portion of the decay of a drum sound. They are shown to be quite effective at reducing false drum detections. The use of multiple templates per drum along with tail templates improves detection accuracy from 83.3% to 90.3%.

The causal beat tracking work from Chapter 4 aims to balance stability and responsiveness through the use of period estimation models operating at different time granularities. An ambiguity measure was developed in order to delegate between multiple period hypotheses. Many configurations of the beat tracking parameters were tested, and every configuration using multiple period hypotheses outperformed all single-period-hypothesis configurations in terms of the allowed metrical levels (AML) overall beat tracking accuracy. The best multi-hypothesis version achieved 71.8% accuracy while the best single-hypothesis version was evaluated at 62.0% total AML accuracy.

The last chapter, Chapter 5, focused on a deep neural network model for analyzing drum patterns. This model was constructed from stochastically trained conditional restricted Boltzmann machines (CRBMs) and fine-tuned to make es-

timates on beat-measure alignment. A new type of visible unit, the bounded linear unit, was used to represent drum loudness. Compared to the standard template correlation approach to measure alignment, which was tested at 64.0% accuracy, a model composed of two stacked CRBMs eliminated almost half of the errors, producing 80.4% accuracy. When used for quarter note alignment, the same model was 95.6% accurate versus the 89.3% figure of the 2-template correlation (again eliminating about half of the errors). It was also shown that the results from these CRBM-based models can be improved using an HMM to filter the alignment estimates across time.

6.3 Future Directions

Because music information retrieval is a relatively new field, there are a paucity of standardized datasets that can be used to evaluate techniques in a standardized way. There are a few existing beat tracking and tempo tracking datasets that have made the rounds, but these consist mostly of recorded polyphonic pop music or ballroom dance tracks and cannot be used to evaluate the drum-specific techniques I have presented here. An important step that would aid progress in the field of automated music understanding would be the standardization of evaluation datasets as is done in the speech recognition and computer vision communities. There have been efforts in this regard (e.g., the MIREX[6] event and some efforts to standardize evaluation [22].); however, support for more specific tasks like drum detection is still lacking. This still does not reduce the importance of doing an objective large-scale comparison of these techniques.

Beyond further comparative testing, it would be interesting to apply the CRBM-based drum pattern model to other rhythm analysis problems such as style classification, perceptual tempo estimation, and fill detection. I would also say that this model has the potential to be very useful in any application that requires the analysis of features across time, which is the vast majority of music information retrieval and audio processing tasks. Using the trained drum pattern model in a generative manner as a stochastic drum machine for use in interactive performance is also a compelling application.

Lastly, an immediate direction for this work would be to incorporate these techniques into a real-world implementation that can be used in a live performance or practice scenario.

[6]The music information retrieval evaluation exchange, `http://music-ir.org/mirex`

Bibliography

[1] P Allen and R B Dannenberg. "Tracking Musical Beats in Real Time". In: *International Computer Music Conference* (1990).

[2] A Banerjee, S Merugu, and IS Dhillon. "Clustering with Bregman Divergences". In: *Journal of Machine Learning Research* (2005).

[3] E Battenberg. "Improvements to Percussive Component Extraction Using Non-Negative Matrix Factorization and Support Vector Machines". M.S. thesis. University of California, Berkeley, Dec. 2008.

[4] E Battenberg, V Huang, and D Wessel. "Toward live drum separation using probabilistic spectral clustering based on the Itakura-Saito divergence". In: *Audio Engineering Society Conference: 45th International Conference: Applications of Time-Frequency Processing in Audio*. 2012.

[5] JP Bello, L Daudet, S A Abdallah, C Duxbury, M Davies, and MB Sandler. "A tutorial on onset detection in music signals". In: *IEEE Transactions on Speech and Audio Processing* 13.5 (2005), p. 1035.

[6] JP Bello, C Duxbury, M Davies, and M Sandler. "On the use of phase and energy for musical onset detection in the complex domain". In: *IEEE Signal Processing Letters* 11.6 (2004), pp. 553–556.

[7] Y Bengio. "Learning deep architectures for AI". In: *Foundations and Trends in Machine Learning* (2009).

[8] N Bertin, C Févotte, and R Badeau. "A tempering approach for Itakura-Saito non-negative matrix factorization. With application to music transcription". In: *ICASSP 2009*. 2009, pp. 1545–1548.

[9] J A Bilmes. *A gentle tutorial of the EM algorithm and its application to parameter estimation for Gaussian mixture and hidden Markov models*. Tech. rep. TR-97-021. International Computer Science Institute, 1998.

[10] J A Bilmes. "Techniques to foster drum machine expressivity". In: *Proceedings of the International Computer Music Conference* (1993).

[11] C M Bishop. *Neural networks for pattern recognition*. Oxford University Press, 1995.

[12] S Böck, F Krebs, and M Schedl. "Evaluating the online capabilities of onset detection methods". In: *International Society for Music Information Retrieval Conference*. 2012.

[13] S Böck, A Arzt, F Krebs, and M Schedl. "Online realtime onset detection with recurrent neural networks". In: *International Conference on Digital Audio Effects (DAFx-12)*. 2012.

[14] CA Bouman. *Cluster: An Unsupervised Algorithm for Modeling Gaussian Mixtures*. `https://engineering.purdue.edu/~bouman/software/cluster/`. 1998.

[15] A Cichocki, R Zdunek, and S Amaria. "Csiszar's divergences for non-negative matrix factorization: Family of new algorithms". In: *International Conference on Independent Component Analysis and Blind Signal Separation*. 2006.

[16] N Collins. "Drumtrack: Beat induction from an acoustic drum kit with synchronised scheduling". In: *International Computer Music Conference*. 2005.

[17] N M Collins. "Towards autonomous agents for live computer music: Realtime machine listening and interactive music systems". PhD thesis. University of Cambridge, 2006.

[18] G E Dahl, D Yu, and L Deng. "Large vocabulary continuous speech recognition with context-dependent DBN-HMMs". In: *Proc. ICASSP* (2011).

[19] G E Dahl, M Ranzato, A Mohamed, and G E Hinton. "Phone recognition with the mean-covariance restricted Boltzmann machine". In: *Advances in Neural Information Processing Systems* 23 (2010), pp. 469–477.

[20] R B Dannenberg. "Following an improvisation in real time". In: *International Computer Music Conference*. 1987, pp. 241–248.

[21] M E P Davies and M D Plumbley. "Context-dependent beat tracking of musical audio". In: *Audio, Speech, and Language Processing, IEEE Transactions on* 15.3 (2007), pp. 1009–1020.

[22] M E P Davies, N Degara, and M D Plumbley. "Evaluation methods for musical audio beat tracking algorithms". In: *Queen Mary University, Centre for Digital Music, Tech. Rep. C4DM-TR-09-06* (2009).

[23] S Dixon. "Automatic extraction of tempo and beat from expressive performances". In: *Journal of New Music Research* (2001).

[24] D P W Ellis. "Beat tracking by dynamic programming". In: *Journal of New Music Research* 36.1 (2007), pp. 51–60.

[25] F Eyben, S Böck, B. Schuller, and A. Graves. "Universal onset detection with bidirectional long-short term memory neural networks". In: *Proc. of ISMIR* (2010).

[26] C Févotte, N Bertin, and JL Durrieu. "Nonnegative matrix factorization with the Itakura-Saito divergence: With application to music analysis". In: *Neural Computation* 21.3 (2009), pp. 793–830.

[27] W T Fitch and A J Rosenfeld. "Perception and production of syncopated rhythms". In: *Music Perception* 25.1 (2007), pp. 43–58.

[28] D FitzGerald, R Lawlor, and E Coyle. "Prior subspace analysis for drum transcription". In: *114th Convention of the Audio Engineering Society*. Dublin Institute of Technology, 2003.

[29] G Garcia. "Optimal filter partition for efficient convolution with short input/output delay". In: *113th Convention of the Audio Engineering Society* (2002).

[30] O Gillet and G Richard. "Automatic transcription of drum loops". In: *Acoustics, Speech, and Signal Processing, (ICASSP '04). IEEE*. 2004, iv–269–iv–272 vol.4.

[31] O Gillet and G Richard. "Transcription and separation of drum signals from polyphonic music". In: *IEEE Transactions on Audio* (2008).

[32] M Goto. "An audio-based real-time beat tracking system for music with or without drum-sounds". In: *Journal of New Music Research* 30.2 (2001), pp. 159–171.

[33] M Goto and Y Muraoka. "Music understanding at the beat level: Real-time beat tracking for audio signals". In: *Computational Auditory Scene Analysis* (1998), pp. 157–176.

[34] F Gouyon and P Herrera. "Exploration of techniques for automatic labeling of audio drum tracks instruments". In: *Proceedings of MOSART: Workshop on Current Directions in Computer Music*. 2001.

[35] S W Hainsworth. "Techniques for the automated analysis of musical audio". PhD thesis. University of Cambridge, 2003.

[36] P Hamel and D Eck. "Learning features from music audio with deep belief networks". In: *Proc. of the 11th International Society for Music Information Retrieval Conference (ISMIR 2010)* (2010), pp. 339–344.

[37] S Handel. *Listening: An introduction to the perception of auditory events.* The MIT Press, 1993.

[38] M Helén and T Virtanen. "Separation of drums from polyphonic music using non-negative matrix factorization and support vector machine". In: *Proc. EUSIPCO* (2005).

[39] R Hennequin, B David, and R Badeau. "Beta-Divergence as a Subclass of Bregman Divergence". In: *Signal Processing Letters, IEEE* 18.2 (2011), pp. 83–86.

[40] P Herrera, A Yeterian, and F Gouyon. "Automatic classification of drum sounds: a comparison of feature selection methods and classification techniques". In: *Music and Artificial Intelligence* (2002), pp. 69–80.

[41] P Herrera, A Dehamel, and F Gouyon. "Automatic labeling of unpitched percussion sounds". In: *114th Convention of the Audio Engineering Society*. 2003.

[42] G E Hinton. *A practical guide to training restricted boltzmann machines*. Tech. rep. 2010-003. University of Toronto, 2010.

[43] G E Hinton. "To recognize shapes, first learn to generate images". In: *Progress in brain research* 165 (2007), pp. 535–547.

[44] G E Hinton and S Osindero. "A fast learning algorithm for deep belief nets". In: *Neural Computation* (2006).

[45] J A Hockman, M E P Davies, and I Fujinaga. "One in the jungle: Down detection in hardcore, jungle, and drum and bass". In: *International Society for Music Information Retrieval Conference*. 2012.

[46] T Jehan. "Creating music by listening". PhD thesis. Massachusetts Institute of Technology, 2005.

[47] A Klapuri. "Sound onset detection by applying psychoacoustic knowledge". In: *ICASSP* (1999).

[48] AP Klapuri, AJ Eronen, and JT Astola. "Analysis of the meter of acoustic musical signals". In: *IEEE Transactions on Speech and Audio Processing* 14.1 (2006), p. 342.

[49] A. Krizhevsky, I Sutskever, and G E Hinton. "ImageNet classification with deep convolutional neural networks". In: *Advances in Neural Information Processing Systems* (2012).

[50] EW Large. "Beat tracking with a nonlinear oscillator". In: *Proceedings of the IJCAI'95 Workshop on Artificial Intelligence and Music* (1995), pp. 24–31.

[51] J Laroche. "Efficient tempo and beat tracking in audio recordings". In: *Journal of the Audio Engineering Society* 51 (Apr. 2003).

[52] D Lee and H Seung. "Algorithms for non-negative matrix factorization". In: *Advances in Neural Information Processing Systems* (2001).

[53] F Lerdahl and R Jackendoff. *A Generative Theory of Tonal Music*. MIT Press, 1996.

[54] G Madison. "Experiencing groove induced by music: consistency and phenomenology". In: *Music Perception* 24.2 (2006), pp. 201–208.

[55] M F Mckinney, D Moelants, M E P Davies, and A Klapuri. "Evaluation of Audio Beat Tracking and Music Tempo Extraction Algorithms". In: *Journal of New Music Research* 36.1 (Feb. 2007), pp. 1–16.

[56] A Mohamed, G Dahl, and G Hinton. "Acoustic Modeling using Deep Belief Networks". In: *Audio, Speech, and Language Processing, IEEE Transactions on* 99 (2011), p. 1.

[57] A Mohamed and G E Dahl. "Deep belief networks for phone recognition". In: *NIPS Workshop on Deep Learning for Speech Recognition and Related Applications* (2009).

[58] V. Nair and G E Hinton. "Rectified linear units improve restricted boltzmann machines". In: *Proc. 27th International Conference on Machine Learning* (2010).

[59] Alan V Oppenheim and Ronald W Schafer. *Discrete-time signal processing*. Prentice Hall, 1989.

[60] R Parncutt. "A perceptual model of pulse salience and metrical accent in musical rhythms". In: *Music Perception* (1994), pp. 409–464.

[61] J Paulus and T Virtanen. "Drum transcription with non-negative spectrogram factorisation". In: *Proc. of 13th European Signal Processing Conference (EUSIPCO2005)*. 2005.

[62] G Peeters and H Papadopoulos. "Simultaneous beat and downbeat-tracking using a probabilistic framework: theory and large-scale evaluation". In: *Audio, Speech, and Language Processing, IEEE Transactions on* 19.6 (2011), pp. 1754–1769.

[63] M D Plumbley, S A Abdallah, J P Bello, M E Davies, G Monti, and M B Sandler. "Automatic Music Transcription and Audio Source Separation". In: *Cybernetics and Systems* 33.6 (Sept. 2002), pp. 603–627.

[64] L R Rabiner. "A tutorial on hidden Markov models and selected applications in speech recognition". In: *Proceedings of the IEEE* 77.2 (1989), pp. 257–286.

[65] L R Rabiner and B H Juang. *Fundamentals of Speech Recognition*. Prentice Hall. 1993.

[66] DA Reynolds. "Approaches and applications of audio diarization". In: *Acoustics* (2005).

[67] DA Reynolds, TF Quatieri, and RB Dunn. "Speaker verification using adapted Gaussian mixture models". In: *Digital signal processing* 10.1-3 (2000), pp. 19–41.

[68] J Rissanen. "A universal prior for integers and estimation by minimum description length". In: *The Annals of statistics* (1983).

[69] A Robertson. "Interactive real-time musical systems". PhD thesis. Queen Mary University of London, 2009.

[70] ED Scheirer. "Tempo and beat analysis of acoustic musical signals". In: *The Journal of the Acoustical Society of America* 103 (1998), p. 588.

[71] K Tanghe, S Degroeve, and B De Baets. "An algorithm for detecting and labeling drum events in polyphonic music". In: *Proc of First Annual Music Information Retrieval Evaluation eXchange (MIREX)* (2005).

[72] G Taylor and G E Hinton. "Two Distributed-State Models For Generating High-Dimensional Time Series". In: *Journal of Machine Learning Research* (2011).

[73] T Tieleman. *Gnumpy: an easy way to use GPU boards in Python.* Tech. rep. 2010-002. University of Toronto, 2010.

[74] M Welling and M Rosen-Zvi. "Exponential family harmoniums with an application to information retrieval". In: *Advances in Neural Information Processing Systems* (2005).

[75] J Yoo, M Kim, K Kang, and S Choi. "Nonnegative matrix partial cofactorization for drum source separation". In: *Acoustics Speech and Signal Processing (ICASSP), IEEE International Conference on* (2010), pp. 1942–1945.

[76] K Yoshii, M Goto, and H G Okuno. "Drum Sound Recognition for Polyphonic Audio Signals by Adaptation and Matching of Spectrogram Templates With Harmonic Structure Suppression". In: *Audio, Speech, and Language Processing, IEEE Transactions on* 15.1 (2007), pp. 333–345.