Implementing and Adapting a Downbeat Tracking System for Real-Time Applications

A Thesis

Submitted in Partial Fulfillment of the Requirements for the Degree

of

Master of Science in Emerging Media – Sound and Music Computing

School of Computer Science

Carnegie Mellon University

Che-Yuan Liang

December 2017

Abstract

Downbeat is the first beat of a measure in music, and downbeat tracking is the task of estimating downbeat locations in a musical audio signal. Compared to beat tracking, downbeat tracking is a higher-level task since it detects the boundary between measures. A *real-time* downbeat tracker is a critical component for many interactive music applications. Musicians can naturally synchronize their (human) performances to within tens of milliseconds in real time. However, it is still a challenging task for computers to achieve human-level *accuracy* and *latency* in music synchronization tasks. In the past two years, machine learning based algorithms have achieved great improvements in accuracy over traditional approaches in the MIREX [1] downbeat estimation evaluation exchange. However, most of the algorithms rely on *look ahead* (the future information an algorithm needs) in order to achieve better accuracy, which prevents these systems from being used in real-time applications.

The goal of this thesis is to adapt a state-of-the-art downbeat tracking system from *offline* computation to *real time* by reducing the *look ahead* and applying *prediction*. For true *real-time* systems, downbeats must be anticipated or predicted, because even a system with 10ms latency would impose an unacceptable delay and will still require prediction to deliver real-time output. We will compare systems by using prediction to hide their latency and then evaluating the accuracy of *predicted* downbeats. Notice that adding *look ahead might* improve performance if the compensating prediction does not degrade the performance much. The trade-offs between the amount of reduced *look ahead* and performance, and between the amount of *increased* future prediction and performance are presented in the experimental results.

As a summary, this thesis research consists of the following steps:

1. Re-implement a state-of-the-art downbeat tracking system.
2. Identify the components of the system that prevent online computation.
3. Adapt the system to online computation by modifying the components.
4. Perform *prediction* to achieve real-time output.
5. Conduct experiments to evaluate the trade-off between *prediction* and *look ahead*.

Acknowledgments

Thanks to Prof. Roger B. Dannenberg for being my advisor and for his guidance throughout this endeavor. His constructive feedback and assistance are remarkable. This work wouldn't exist without his encouragement.

Thanks to Prof. Richard M. Stern for being my committee member and his valuable comments. Thanks to Prof. Florian Metze for granting me the access to the LTI cluster. Thanks to my colleagues who have supported and encouraged me along the way.

This work used the Extreme Science and Engineering Discovery Environment (XSEDE) Bridges GPU at the Pittsburgh Supercomputer Center through allocation TG-CIE170024 [2].

This thesis is dedicated to my parents. Thanks to my family for supporting my dream.

Contents

1 Introduction

The *beat* [3] is often defined as the rhythm to which listeners would tap their toes when listening to a piece of music. Music is often organized into groups of 3 or 4 beats called *measures* that share common rhythms (e.g. bass drum on beats 2 and 4). The first beat of each measure is the *downbeat*, which often is the beat where harmonies change, and new phrases begin. In studies of rhythm, tempo, and timing, *tatum* [4] refers to a subdivision of beats that most highly coincides with note onsets. Downbeats, beats and tatums are all *imaginary*[1] time locations that divide the music in the time axis at different levels – downbeats divide music, beats divide downbeats and tatums divide beats.

Downbeat tracking is the task of estimating the downbeat locations in a musical audio signal. The fundamental reason to have a downbeat/beat tracker is that it is almost impossible for humans to perform music in an absolutely steady tempo with no variation. Even if it were possible, composers or performers might deliberately change tempo throughout the piece. Downbeat tracking is a fundamental task for automatic music segmentation [5] or intelligent instruments, and it enables many interesting computations supporting live music production as it allows the computer to model how humans parse the melodic/rhythmic patterns from the musical signal. For example, downbeat tracking can be used to build a real-time sampler [6], that can automatically record audio in measures with varying time length, as an extension of traditional loop pedals that rely on

[1] The "ground truth" of tatum, beat or downbeat locations are defined by humans. Given a piece of music, timings might vary from listening to listening and from person to person, but in most cases people agree with each other, at least to within a time interval that is small compared to note lengths.

humans to follow the metronome. A real-time downbeat-tracking algorithm can create lots of possibilities in interactive performance [7].

Although downbeat/beat tracking is a quite subconscious level task for humans, just like riding a bicycle, it is hard to formulate algorithmically. Over the past two decades, researchers have been trying to formulate beat tracking algorithmically, as will be discussed in Section 2; however, most of these algorithms are not able to perform well in a wide variety of music. Recently, machine-learning-based (ML) methods have achieved great improvement [8] [9] over traditional methods in the MIREX downbeat estimation evaluation exchange [1] and in the original experimental results [8] [9]. In certain genres that have strong rhythmic components and clear clues of downbeats, such as the Ballroom dataset [10] and Pop music (i.e. The Beatles dataset and RWC-Pop), ML-based method can achieve almost 90% in f-score measurement.

A real-time downbeat tracking system should input a raw audio signal stream and output the beat prediction within an acceptable latency (which might even be less than zero to allow time to generate a sound that is synchronized with the beat). However, in order to achieve better accuracy, most systems often require *look ahead* (which will be discussed in Section 3 in detail), limiting the system from being online and real-time. As long as the latency of the system is higher than human perceivable tolerance, which is around 50-70 ms in most evaluation systems (and even this could be considered much too generous), the system is considered not to be a real-time system. Since there are some hard limitations, such as the signal processing analysis window, which make the true real-time response hard to achieve, it seems that a beat tracking system (and humans as well) must predict beats at least slightly ahead of real time to enable real-time applications (i.e. latency slightly less than zero).

As a general rule, the longer the prediction is, the more uncertain the prediction will be. Since the system does not use the information within the prediction interval (which has a duration equal to the *look ahead* of the downbeat tracker), the system cannot respond to any tempo change during this interval. On the other hand, longer prediction ahead lets the system use more temporal context (*look ahead*), which might benefit the performance of the system. Therefore, the goal of this thesis is to discover the trade-off

between the how the amount of *prediction* and *look ahead* affect the overall performance in a specific downbeat tracking system of choice. Since this work is based on a non-real-time beat tracker that is not open-source or available from the developers, a custom implementation is made as part of the thesis.

1.1 Terminology

Since our use of prediction to overcome look ahead does not have a standard terminology in real-time systems, we define some terms here to clarify our approach and main concepts. This terminology will be used consistently throughout the thesis, in which we discuss a system that takes an input stream and generates the output stream within a certain *latency*, which is the result of *look ahead* and *prediction.* The relationship is shown in Figure 1.

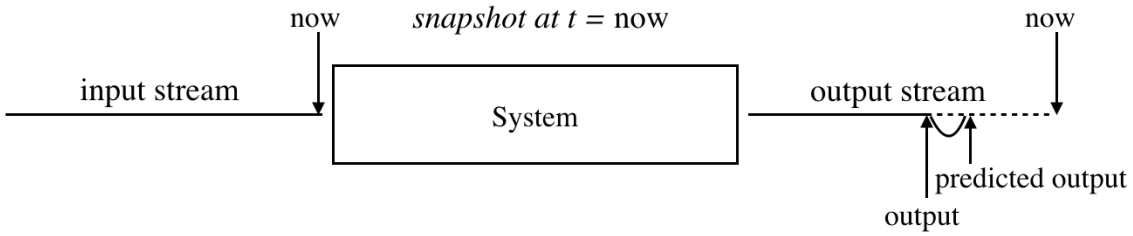

Figure 1 Illustration of the relationship between latency, look ahead, and prediction of a system. The broken line denotes the amount of look ahead, and the curve denotes the amount of prediction.

Definition

- *Latency* is the delay between the input stream and the output stream

  $$latency = look\ ahead - prediction$$

- *Look ahead* is the amount of future information an algorithm/system needs in order to produce the output at certain time instance, such as the future context needed for the algorithms or the analysis window used for signal processing (shown as the broken line in Figure 1).
- *Prediction* is how far into the future output is estimated, given the past output (shown as the curved solid line in Figure 1).

The system can be classified as:

- *Offline system*: A system that has access to all input before producing any output. We could say that an offline system has *arbitrary* or *infinite* latency.
- *Online system*: A system with *finite* latency.
- *Real-time system*: A special case of *online* systems where the latency due to all factors including computation time is less than a certain tolerated threshold depending on applications.

1.2 Scope

In the real world, there are also other causes of latency not addressed in Figure 1. Such as 1) the *low-level latency* caused by the hardware and software buffer and 2) the *scheduling latency* caused by the operating system, which we believe are typically on the order of 10 ms.

This thesis will focus on the latency caused by the *look ahead* (and *prediction*), as it is difficult to conduct evaluation for latency on the scale of few milliseconds, and because our focus is on methods and algorithms rather than operating systems and code optimization. For the same reasons, the experiments will be conducted in offline simulation. Furthermore, the latency caused by look ahead is on the order of seconds, so it is reasonable to ignore the low-level latency and scheduling latency. In fact, we will see later (Section 5.3.2.3) that changing the amount of prediction by a few tens of milliseconds will have negligible impact on accuracy.

1.3 Organization

The rest of this thesis is organized as follows: In Section 2, related work will be briefly summarized. Section 3 will provide an overview of the state-of-the-art downbeat tracking system that works offline. Section 4 will describe the implementation of the original paper in detail and provide benchmark results. Section 5 will describe the look ahead related bottlenecks of the system, assumptions, approaches to adapting the system to real time, and the benchmarks after applying the modification. Section 6 will conclude this thesis and discuss possible future work based on the results.

2 Related Work

In the past two decades, people have been trying to hand design beat tracking algorithms. For example, formulating beat tracking as an optimization problem based on one's assumptions (i.e. "beats should align with onsets and beats should be locally steady in tempo," and the solution that satisfies both constraints is the beat sequence [11]). Standard signal processing techniques [12] [13] (e.g. auto-correlation and comb-filters) have been applied to extract the periodicity as a sub-task of the beat tracking system. Probabilistic models [14] have been designed to encode the musical time signature structure and to fit the long-term signal pattern. In sum, hand-designed algorithms require both excellent understanding in music and engineering domain knowledge in order to design the algorithms.

In recent years, machine learning algorithms (i.e. deep neural networks) have achieved great improvement over the traditional approaches in MIREX downbeat estimation evaluation exchange [1] There are two distinct of series of works that seem to outperform others: Sebastian et al. (BK4 [9] and related work KB1 and KB2 [15]) use recurrent neural networks (RNNs), and Simon et al. (DBDR [8], and related work [16]) use convolutional neural networks (CNNs) to extract features and model temporal context.

Generally speaking, both systems consist of two stages:

1. The first stage of the system uses neural network(s) to compute the likelihood of a beat at any given time, resulting in a one-dimensional time series.
2. The second stage of the system uses a probabilistic model (i.e. a Hidden Markov Model) encoded with musical knowledge to infer a path through hidden states (thus the downbeat locations) from the output of the neural network.

As a comparison, these two systems (BK4 and DBDR) are differed as follows:

1. In the first stage, BK4, directly operates on the magnitude spectrogram using a bi-directional RNN and outputs the beat likelihood sequence, while DBDR first divides the audio into subdivisions (using the tempogram toolbox [17]), which are then analyzed to obtain different features.
2. In the second stage, BK4 uses a bar-pointer model that encodes the phase of the beat within the measure, thus BK4 is tracking beat and downbeat at the same time. On the other hand, DBDR's model does not intend to derive the phase of the beat within a measure, but only the phase of the tatum[2].

In terms of the real-time feasibility, both systems (BK4 and DBDR) make extensive use of future information. First of all, BK4 use a bi-directional RNN to process the full sequence of audio input, and DBDR looks 8 tatums ahead. In addition, DBDR uses the tempogram toolbox [17], which consists of various large signal processing analysis windows (which will be discussed in detail in Section 3), making it challenging to adapt it to real-time application. Secondly, the decoding stage of HMMs also look at the full sequence of observations in order to derive the global optimal path. Since BK4 has a relevantly simple pipeline which is basically data driven, it could be modified to real-time application with less effort than DBDR. For example, KB1 and KB2 (the variations of BK4) use a uni-directional RNN, which reduces the need to look ahead in the first stage, and it only degrades the performance accuracy moderately as shown in MIREX evaluation exchange. Moreover, earlier in 2017, the KB series was modified to create a real-time beat tracker and submitted to the IEEE Signal Processing Cup and awarded as one of the top three contestants, although the accuracy benchmark result is not reported publicly.

Since the deep-learning-based downbeat tracker came out in around 2015, the attempt to adapting these beat tracking algorithms to real-time is fairly few. For example, [18] presents a study on how to speed up the offline processing of downbeat tracking

[2] For example, if a there are 8 tatums in a measure, the phase of tatum goes from 0/8, 1/8, 2/8 … 6/8, 7/8 within a measure.

using the similar CNN architecture as DBDR, and [19] adapts the CNN architecture to a real-time beat tracking application; however, the performance presented in MIREX 2017 is not comparable to the BK series.

This thesis chooses to adapt DBDR for real-time application for the reason that DBDR is arguably the best algorithm as presented in the original paper. Secondly, BK/KB algorithms are already published as open source, so I believe it is a good learning experience to implement a downbeat tracking system from scratch.

3 System Overview

An overview of the original downbeat tracking system will be introduced in this section, including the original paper [8] and tempogram toolbox [17], which plays an important role in this system. This section will focus on the parts of system that are related to real-time downbeat tracking applications and will leave irrelevant details to the references.

The DBDR system consists of three stages: 1) feature extraction (tatum segmentation and feature pre-processing) 2) feature learning with convolutional neural network and 3) Hidden-Markov Model decoding as illustrated in Figure 2. The following paragraphs will describe the system in this order.

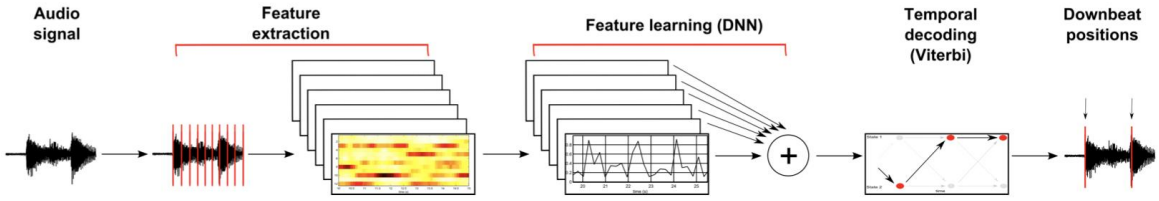

Figure 2 The overview of the DBDR system (taken from the original paper.) The red vertical lines one the left are the tatums that segment the audio.

3.1 Feature extraction

This stage consists of two parts: tatum segmentation and feature pre-processing. Tatum segmentation is a way to perform non-uniform sampling on the pre-processed feature.

3.1.1 Tatum segmentation

There are several reasons to segment the audio with tatums in the first step. First of all, this can reduce the search space and therefore reduce the computation. Secondly,

this can create a *tempo-invariant*[3] feature, since tatums provide a non-uniform sampling on musically meaningful time events. Notice that the tatums are estimations, which must have a high recall rate to the downbeat locations; otherwise, the sample loss cannot be recovered in the later stages of the system.

Tatums are computed using the "tempogram toolbox [20]." The goal of the tempogram toolbox is to synthesize a mid-level representation that captures the periodicity of meaningful musical events. The tempogram is analogous to a spectrogram: at each point along the time axis is a column that expresses periodicity of onset features. Each element of the column represents a different tempo. Given a tempogram, one can search for the strongest tempo at each time point, typically using dynamic programming to enforce continuity constraints so that a smoothly changing "tempo curve" (as a function of time illustrated in Figure 3) can be obtained. Since the tempogram contains phase information, the "tempo curve" can be "synthesized" into a smooth quasi-periodic function called the predominant local pulse (PLP) curve, a one-dimensional time series (a real-valued function of time), and tatums are derived by simply performing peak-picking on this PLP curve. In terms of the look ahead, those steps use large analysis/synthesis windows as illustrated in Figure 4and the following steps:

3.1.1.1 Compute onset detection function (*onset look ahead*)

The original implementation first uses a 1024 sample wide analysis window to perform Short-Time Fourier Transform, which introduces few milliseconds of look ahead. However, a large normalization window (5 s) is used for calculating the onset detection function from the spectrogram to yield a better feature quality.

3.1.1.2 Compute tempogram (*tempogram look ahead*)

The original implementation uses a 6-second-long window to compute the tempogram as described above.

---

[3] Here tempo-invariant means the extracted feature is invariant to tempo change. For example, when a music piece is played in different tempo, the extracted feature should be the same.

3.1.1.3 Compute tempo curve (tatum Viterbi look ahead)

The Viterbi algorithm is used to solve the maximum accumulated path (tempo curve) of the magnitude of tempogram minus the transition cost (penalty) for long distance jumps. Since the Viterbi algorithm is an offline algorithm, this is the major look ahead bottleneck in the system. The original implementation only considers the tempo curve above 60 bpm.

3.1.1.4 Compute overlap-add synthesis (*tempogram look ahead*)

The overlap-add synthesis can be thought of as a inverse process (inverse Fourier transform) of computing the tempogram (Fourier transfrom), but we apply filter (tempo curve) to focused on the phase and magnitude of the tempogram along with the tempo curve. The PLP curve is derived by performing the overlap-add synthesis with a 6-second-wide window. Specifically, for each synthesis time step *t*, a sinusoid with the frequency and phase at tempogram(tempo_curve(t), t), modulated with a synthesis window, is added to the PLP curve. For this system, the resolution of the synthesized PLP curve is 5 milliseconds, which is also the feature rate of the onset detection function.

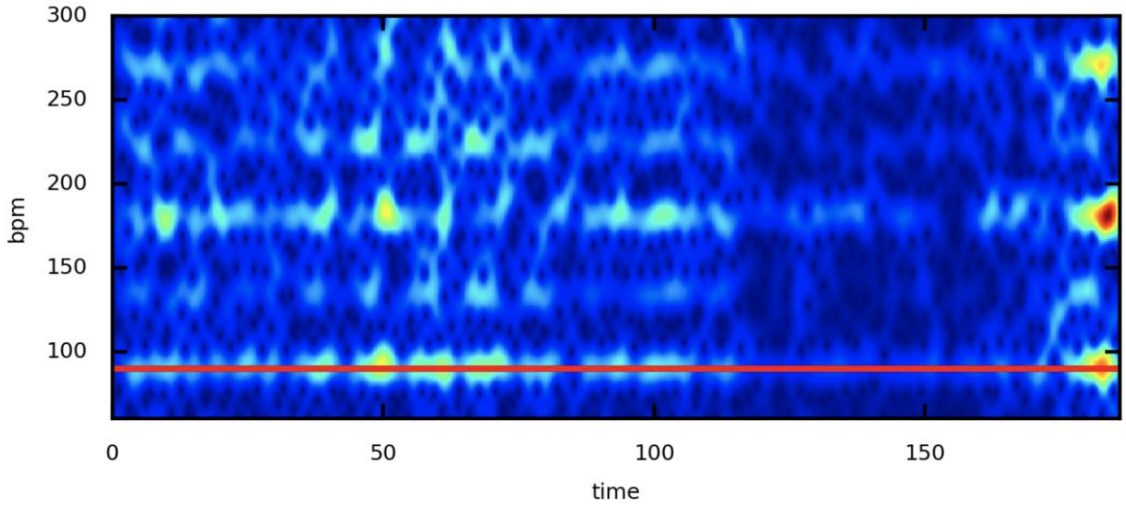

Figure 3 Magnitude of tempogram and decoded tempo curve. The red flat line is the tempo curve, computed by Viterbi algorithm.

3.1.1.5 Peak-picking on PLP curve

The peak-picking algorithm works as the following pseudo-code (Python code is in *pulse.py*):

for each location i:

if k left-side samples (inclusive) are monotonically increasing and k right-side samples (inclusive) are monotonically decreasing, then the location i is a peak

else the location is not a peak

Since the PLP is synthesized with a smoothed sinusoid curve, which has a simple shape of peak and valley without spiky noise, k is set to be 1 for both sides.

As a summary, the look aheads in the tatum computation is illustrated in Figure 4. The onset detection uses a 5-second-long normalization window; the tempogram computation uses a 6-second-long analysis window; and the PLP synthesis stage uses the same windows as tempogram to perform overlap synthesis, resulting in a look ahead of 2.5 + 6 = 8.5 s. Also, the tempo curve is derived by the Viterbi algorithm, so it introduces an infinite (arbitrary) look ahead.

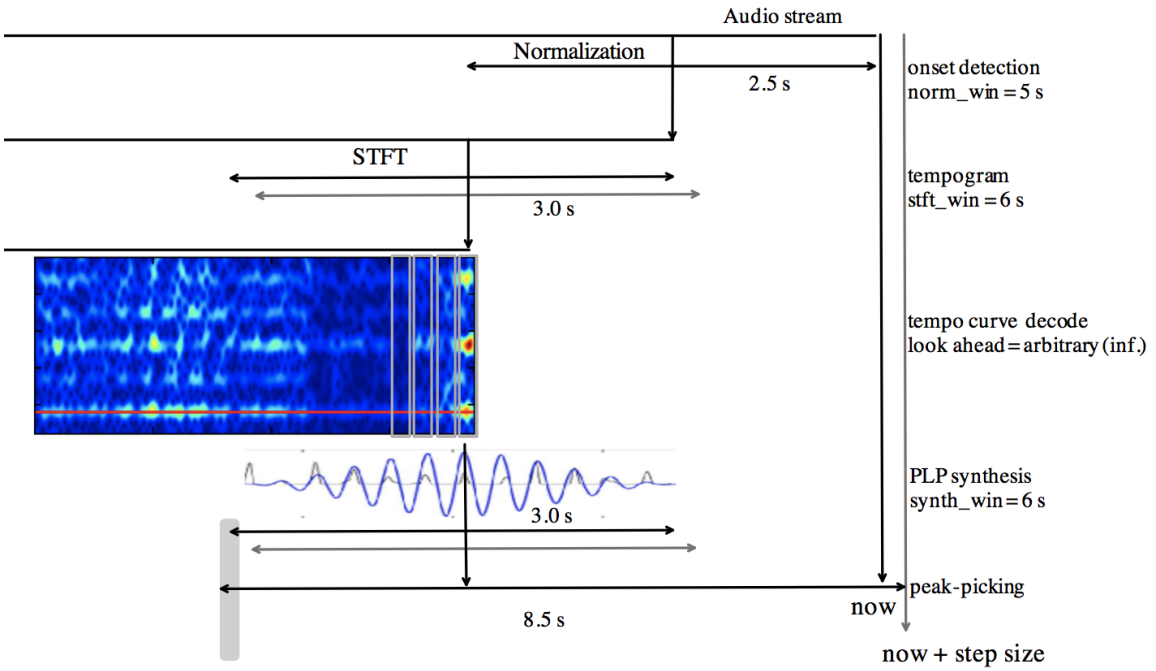

Figure 4 The look ahead diagram for tatum computation.

3.1.2 **Feature pre-processing**

There are three kinds of feature pre-processing methods – harmonic feature, rhythmic feature and melodic feature. Since they are computed similarly, this section will use harmonic feature pre-processing as example. The same concept can be applied to other features.

The harmonic feature is the Chroma-gram which maps (warps) the spectrogram to 12 pitches to capture the harmonic progression of each audio track as shown in Figure 5. This feature is first sampled at the tatums (red line) non-uniformly. For each sample point (tatum), a moving window is used to select a two-dimensional "image," where the x-axis is the time dimension (measured in tatums) and y-axis is the Chroma index. This window captures the temporal context, resulting in a 3D tensor (number of tatums, 12, temporal context width in tatums) for each audio file.

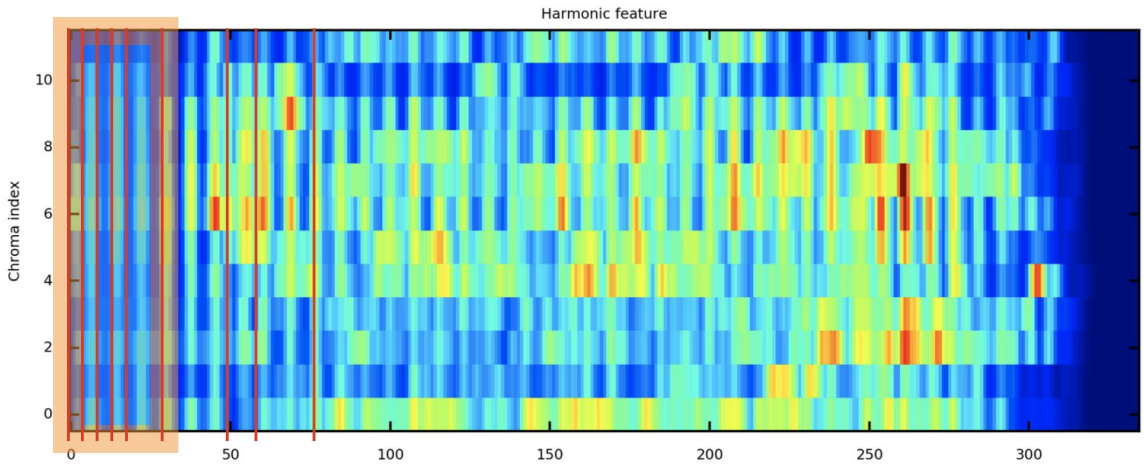

Figure 5 Feature extraction illustration. A moving window (shown at left as a rectangle with a thick outline) is used to capture the temporal context for each tatum.

Specifically, the temporal context for harmonic feature is 9 tatums (i.e. 4 tatums in the future + 4 tatums in the past +1 at the center), and each pair of adjacent tatums are linear interpolated by 5 sample points to increase to time resolution. For an audio with *T* tatums, this will result in a *T* x 12 x *(9* x *5)* tensor.

The same logic for feature extraction can be applied to the rhythmic feature and melodic feature. For reference, the window size, the hop size and the temporal context in detailed in Table 1.

	Harmonic	Rhythmic	Melodic
STFT step size	46.4 ms	23.2 ms	185.8 ms
STFT window	185.7 ms	11.6 ms	11.6 ms
Temporal context	9 tatums	17 tatums	17 tatums

Table 1 The STFT parameters and temporal contexts for each feature.

3.2 Feature learning with neural network

At each sample point (tatum), a 2D feature is sent into the neural network, and the network will output the downbeat likelihood(s). There are two formats of neural network outputs. Let *t to be the* sample point: For harmonic and melodic network, the output at time *t* is a *single value*. And for rhythmic network the output is a *vector of values*, which

contains the 17 tatums of downbeat likelihood centered at time *t*. The multiple estimations at each *t*, will be further averaged into one single value. The original paper claims the rhythmic network can learns better by doing this.

Finally, there will be three 1-D time series of downbeat likelihood for each audio, output from three neural networks independently. In the original paper, these outputs are simply averaged in to a 1-D downbeat likelihood series which is served as the observation sequence of the Hidden-Markov model.

3.3 Hidden Markov Model (HMM) decoding

The final observation sequence is actually the downbeat likelihood for aligned at each tatum (number of tatums is equal to the length of the sequence). Each tatum is assigned to have a hidden-state solved by the Viterbi decoding algorithm. In the original paper, the Viterbi decoding algorithm looks at the whole sequence of the observation sequence to compute the global optimal path. This will be another look ahead bottleneck but could be mitigated by doing Viterbi decoding at each time step as a compromise with the performance as describe in Section 5.

4 Re-implementation

An implementation of the original downbeat tracking system is made as a part of this thesis in order to modify the original system. This implementation follows the original paper as accurately as possible, however, considering the time constraint and missing details in different parts of the original system, some work-around modifications are made. This section will address them in detail. A benchmark result comparing with the original implementation executable program provided by the author is present in the end of this section.

4.1 Dataset

The original paper uses 9 datasets to train and fine-tune the neural networks in a leave-one-out fashion. There are summaries shown below (taken from the original paper).

Name	# Tracks	# DB	Length	Genre
RWC Class [45]	60	10148	5h 24m	Classical
Klapuri subset [15]	$40 - e$	1197	0h 38m	Various
Hainsworth [46]	$222 - e$	6180	3h 19m	Various
RWC Jazz [45]	50	5498	3h 44m	Jazz
RWC Genre [47]	92	11053	6h 22m	Various
Ballroom [48]	$698 - e$	12219	6h 04m	Ballroom dances
Quaero [49]	70	7104	2h 46m	Pop, rap, electro
Beatles [50]	179	13937	8h 01m	Pop
RWC Pop [45]	100	10835	6h 47m	Pop
Total	1511	78171	43h 05m	

Table 2 Datasets overview

In this implementation, the size of dataset is about the same in total length though there are some missing and extra datasets. Three missing datasets are Klapuri, Hainworth and Quaero. These are either not available or do not have public annotations. On the other hand, there are three extra datasets: RWC Royalty-Free, Zweieck, and GTZAN provided from Prof. Dannenberg and Matthew Davies. Both Royalty-Free and Zweieck datasets consist of 15 full tracks, and GTZAN consist of 1000 small clips in various genre cropped to 30 seconds long.

4.2 Rhythmic feature signal pre-processing

The original paper applies μ-law compression [21] to process the STFT spectrogram in the rhythmic feature, however, in the experiment we found that this operation will cause the rhythmic feature to be extremely noisy. By plotting out the input/output as of μ-law compression in input range from -1 to 1, using μ = 1E6 and further plotting out all input range from floating point -1 to 1 to 16-bit integer, none of the results from the compression seems to be correct and they cause the output to be distorted. We find the function is actually serving as an expander as shown in Figure 6, therefore we didn't apply this pre-processing step for the rhythmic feature. One thing to notice is that the original implementation uses MATLAB, and mine use Python libraries (e.q., Scipy, Numpy). This discrepancy might suggest that the numeric value of the original paper and my implementation might be in a different scale.

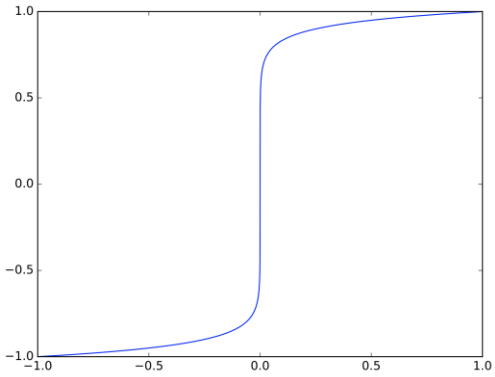

Figure 6 input / output of u-law compression. X-axis is the input, Y-axis is the output from compression algorithm.

4.3 Tatum computation

The tatums are derived by performing peak-picking on the PLP curve computed from the tempogram toolbox, which is publicly available [20]. The naïve way to derive the tempo curve is to directly take the *argmax* path of tempogram at each time. However, this will generate a discontinuous path. In the original downbeat tracking paper, dynamic programing is applied in order to impose a strong continuity constraint to the tempo curve before synthesizing the PLP curve; however, the cost parameter is not detailed. With trial

and error on a subset of 25 audio files randomly selected from all datasets, we find the cost works best with:

$$cost(i,j) = abs(i-j)$$

, where i and j are simply the index of the tempogram matrix. Note that the optimal value actually depends on the numerical values presented in the tempogram matrix.

The result of applying constraints is shown in Table 2, which increases both recall and precision by about 2 percent.

Method / F-measure	Recall	Precision	F-score
Max tempo curve (naive)	95.96	11.85	18.69
Backtracked tempo curve (DP)	97.53	13.54	23.25

Table 3 The f-measure of tatum computed with or without dynamic programing (DP) with downbeat as ground truth with 70 ms tolerance.

4.4 Rhythmic feature extraction

In this implementation, the interpolation ratio of the rhythmic feature is increased from 5 to 21 interpolation points in between two tatums in order to have better temporal resolution. This modification is to deal with the fact that the computed tatums are not perfectly aligned with the peak of rhythmic feature. Although the tatums have a recall rate of 97% within 70 ms of tolerance shown in Table 3, most of them do not hit the peak, in fact, tatums are detected earlier as illustrated in Figure 7. We can see the onset detection function always detects the beginning of the peak by definition, since the PLP curve is synthesized based on this, hence the tatums will also be estimated a bit earlier. In the case of rhythmic feature that have relatively narrow bandwidth, if we don't sample precisely or increase the sample rate, the peak information will lose or distort the feature.

In most cases the largest periodicity of tatums is 500 ms (i.e. 120 tatums per second), which means the temporal resolution is only 100 ms after the linear interpolation. However, most of the bandwidth of rhythmic features is much smaller than this. As long as the tatums are off by the peak by a small fragment, such as 10 ms, the extracted feature will lose the important peak information. Therefore, we increase the

interpolation ratio by 4 times, so that we can have a 25 ms of rhythmic feature resolution or smaller.

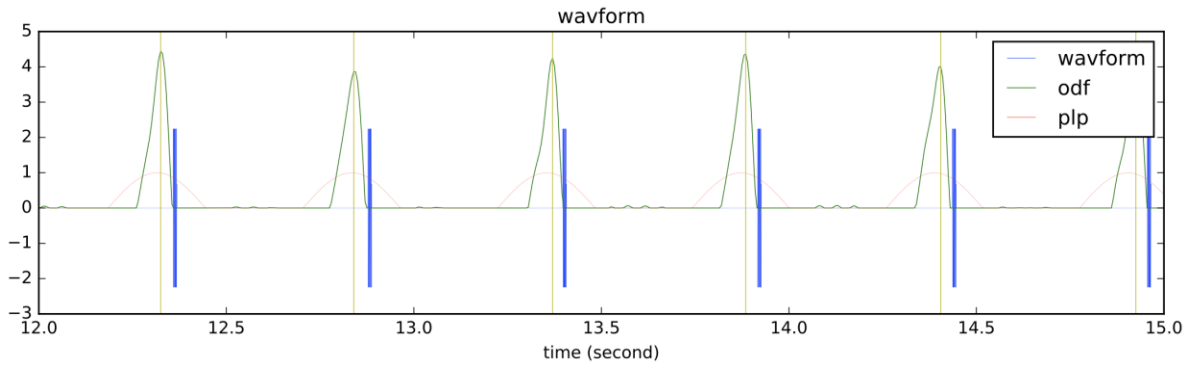

Figure 7 The misalignment of the peak of waveform (waveform), onsets (odf), and peak of PLP (plp).

4.5 Not implementing melodic network

I did not implement the melodic network because it adds extra computation and development time without gaining much performance. The input feature dimension for the melodic network is 304x85, which is about 100 times larger than the original rhythmic network (i.e. 3x85) and about 20 times larger than the modified one in this implementation. However, the information gain from melodic network is limited considering we already have a harmonic network as the Chroma-gram is basically a degenerate version of the melodic feature. Moreover, as addressed in the original paper, the performance gain is only 0.3 percentage points when comparing the system with three networks to the system with 1 melodic plus 2 harmonic networks.

4.6 Rhythmic network architecture

Since the dimension of the rhythmic feature is modified as addressed in sub-section 4, the input dimension is modified from the original 85 (i.e. 17 tatums with 5-point interpolation) to 357 (i.e. 17 tatums with 21-point interpolation). The input dimension for the first layer and the max pooling layer are also scaled accordingly to cover the same amount of information. Specifically, the resulting filter size for the first layer is modified to (3, 174) from (3, 40), and the first max pooling layer is modified to (1, 8) from (1, 2). The detail of network layouts can be found in *model.py* in the source code.

4.7 Neural network training phase (hyper-parameters, data preprocessing…)

There are various details related to the training phase of the convolutional neural network not mentioned in the original paper and its previous works, such as the hyper-parameters and data preprocessing strategy. Since each configuration takes 2-4 hours of training plus modification of code to see the result, this implementation did not perform a complete grid search due to the time and computation constraint. This implementation therefore chooses the parameters empirically by trial and error.

The missing information includes:

1. Batch-size,
2. Dropout rate
3. Learning rate and decay rate
4. Momentum for stochastic gradient descent
5. Shuffle strategy (in-batch or dataset)
6. Regularization strategy
7. Initial distribution
8. Normalization (standardization, mean subtraction) strategy and so on.

For this implementation, those details are listed in the source code *train_harmonic_network.py* and *train_rhyhtmic_network.py* for reference.

4.8 Transition probability of Hidden Markov Model

In the original paper, the transition probability of the HMM is directly assigned, with some empirical tweaking. However, the tweaking step is unclear, so this implementation did not do the further tweaking, and the HMM parameters are assigned by the following rule:

Let the HMM states denoted by “n/k”, where k is number of tatums per measure, and n is the phase of the tatum ($0 < n < k$). For example, the original HMM has k in $K = \{3, 4, 5, 6, 7, 8, 9, 10, 12, 16\}$. For example, for $k = 3$, there are 3 states: “0/3”, “1/3”, “2/3”, for $k = 4$ there are 4 states: “0/4”, “1/4”, “2/4”, “3/4” and so on. So, there will be 10 groups of states in different k.

Let transition (i, j) denote the probability of going from i state to j state, the transition probability is assigned as follow:

1. Assign transition (i, j) = 0.95 if j is the next tatum phase in the same group (same k). For example, from “0/4” to “1/4”, or from “4/5” to “0/5”.
2. Add 0.1 to all transition probability to smooth out the distribution.
3. Normalize the transition matrix to be a right stochastic matrix.

The specific detailed can be found in *hmm.py*.

4.9 Evaluation

The evaluation is conducted as described in the original paper using the beat tracking evaluation toolbox [12]. The dataset is split in a leave-one-out manner (the test set is from a completely different dataset, not used in training). One minor difference is that this evaluation does not discard the first and last few seconds of the audio, which make it slightly stricter than the original evaluation as these regions usually perform badly.

Table 4 shows the benchmark on the *Ballroom* dataset, which consist of 698 audio tracks. The f-score is computed by averaging every f-score of the audio track. For this thesis implementation, the performance is further decomposed into three intermediate steps: “tatum” directly evaluates the f-measure against downbeat annotation; “peak-picking” evaluates the peaks from the averaged output of two neural networks; “HMM” performs Viterbi decoding from the same input used in “peak-picking”.

We can see the HMM performance is about 76% of the original implementation. First of all, when looking at the tatum and peak-peaking performance, we can see the tatum computation is performing as expected. It achieves the recall rate of 97% while maintaining a precision of 15%, which suggest the periodicity of the tatum is about 8 times faster than that of the downbeat. Secondly, we can see that in each of the stages, the recall decreases and the precision increases, and every step increases the f-score by about 20%. This suggests both the neural network and HMM are beneficial to the system; however, there is still room for tweaking in both steps, especially in the neural network stage.

In the original paper's previous work [22], which uses the same HMM model (with different parameters), the performance gain by applying Viterbi decoding to the overall f-score is also around 20%. This suggest the HMM implementation is doing a reasonable job and the problem is most likely coming from the neural nets. In fact, during the training phase, we find the models converges fast in the first 3 epochs and starts to over-fit the training data. Though we have tried different combination of the parameters and pre-processing strategy as described in the previous Section 4.7, this result is the best we can achieve given the missing details from original paper.

	Method	Recall	Precision	F-score
Origin	paper	N/A	N/A	~80.00
	executable	81.11	84.69	**80.23**
Thesis	tatum	97.05	14.74	24.96
	peak-picking	83.16	29.51	41.40
	HMM	79.83	54.00	**60.94**

Table 4 Downbeat detection result on Ballroom dataset. "paper" refers to the result shown in the original paper [8], "executable" refers to the result from the executable provided by the author. "tatum" refers evaluating tatums with downbeat annotations.

5 Adapt to Real-Time

For a real-time downbeat tracking system, the latency of the system has to be reduced to at least zero. To decrease the *latency*, the system can be modified in two directions: one is *reducing the look ahead*, and the other is *increasing the amount of prediction*. Generally speaking, both directions will degrade the accuracy. For example, state-of-the-art beat tracking systems are off-line algorithms that look ahead many seconds for better performance, suggesting that look ahead is important. Prediction cannot react to unseen tempo change, resulting in a loss of accuracy whenever tempo changes (and of course, the fact that tempo is unsteady is the main reason we need downbeat tracking to begin with). Therefore, the goal of this section is to discover the optimal combination of *look ahead* and *prediction* under a given latency constraint (i.e. *latency* < *0*). The assumptions, detail of modifications, and the evaluation results will be covered in this section.

5.1 Total look ahead estimation

The total *look ahead* of the system is estimated as follows. Suppose a system has only two components with *look ahead L1 and L2* respectively. If *L1 and L2* are serial (i.e. the input of *L2* is the output of *L1*), the *total look ahead* is *L1 + L2*. On the other hand, if *L1* and *L2* are parallel (i.e. they both take the same input), the *total look ahead* is determined by *max (L1, L2)*. For example, the total look ahead shown in Figure 8 is:

$$total\ look\ ahead = \max(L0, L1, L2) + \max((L3 + L4), L5)$$

This property implies that when the components are parallel, the components with less look ahead can potentially increase its own look ahead up to the maximum look ahead among all parallel components without changing the *total* look ahead. For example, in CNN + HMM computation, 2 neural networks are parallel with different look aheads (8 tatums ahead for rhythmic network and 4 tatums ahead for harmonic network). In this case, the harmonic network can actually look ahead up to 8 tatums while keeping the same total look ahead.

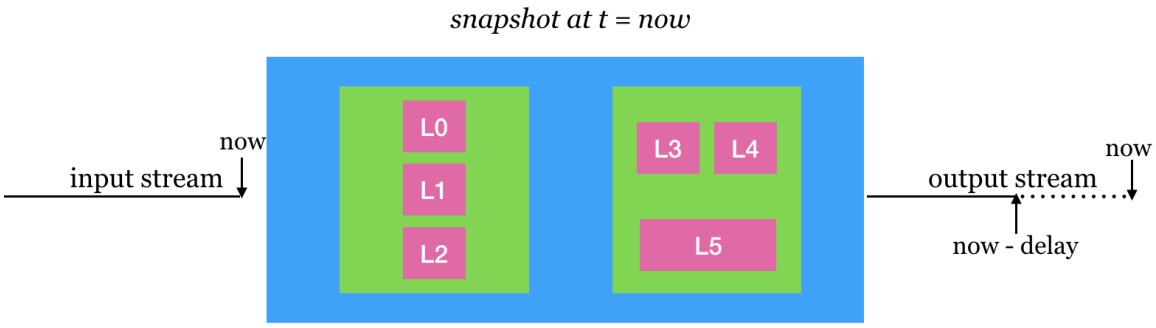

Figure 8 A example of system with various components each has different amount of look ahead. L0, L1, L2 are parallel and L3, L4 are serial. The green blocks are serial.

5.2 Predicting Tatums/Downbeats

Typically, real-time systems are evaluated in terms of meeting or exceeding deadlines for the delivery of results. For musical beat- and downbeat-tracking systems, the tolerable deadline might even be slightly negative to allow for latency in the synthesis of musical output to be synchronized to beats of the input. To achieve real-time performance, it seems necessary to predict tatums/downbeats in the future to make up for the low-level system latency and look ahead found in any downbeat detection system.

This prediction is built upon some assumptions described here. We assume that tatums/beats/downbeats are periodic pulse trains with certain deviation caused by various reasons. On the performance side, the deviation might be coming from the musical expression, e.g. the bass playing earlier to create a certain feeling or style, and the speed of sound causing instruments to be not perfectly synchronized when they are in the distance. In these cases, the deviations are usually repetitive and predictable. On the other hand, the deviation might also be coming from player error, which can be approximated by a Gaussian distribution. On the downbeat tracking algorithm side, the deviation might be coming from the ambiguity of the detection boundary (i.e. the ambiguity of onset locations for soft onsets, which can be up to 100 ms). Or, deviation might be coming from the low time resolution of analysis window, down sampling or quantization error. These distributions are also assumed to be Gaussian.

With the assumptions above (i.e. the stability property of pulses and Gaussian distribution of error), the future locations of tatums seem to be suitably modeled with linear regression as illustrated in Figure 9. We assume that tatums in a “perfect”

performance is equally spaced in time, and therefore a plot of tatum number as a function of time creates a straight line parameterized by the *slope* and the *intercept*. For prediction, the objective is to find a straight line that minimizes the mean squared error of the estimations and observations (i.e. the errors in horizontal direction), where the observations are *n* most recently generated pulse locations.

We expect there will be a trade-off regarding the *number of most recent observations* used to fit the linear regression model. Because we assume that the observations of pulses are noisy, the more observations we use to fit the model, the more accurate the estimation we can approach, however, at the cost of losing responsiveness to tempo change – since the tempo is only locally stable. Therefore, this section will also evaluate the results of future prediction using different number of samples as a control variable.

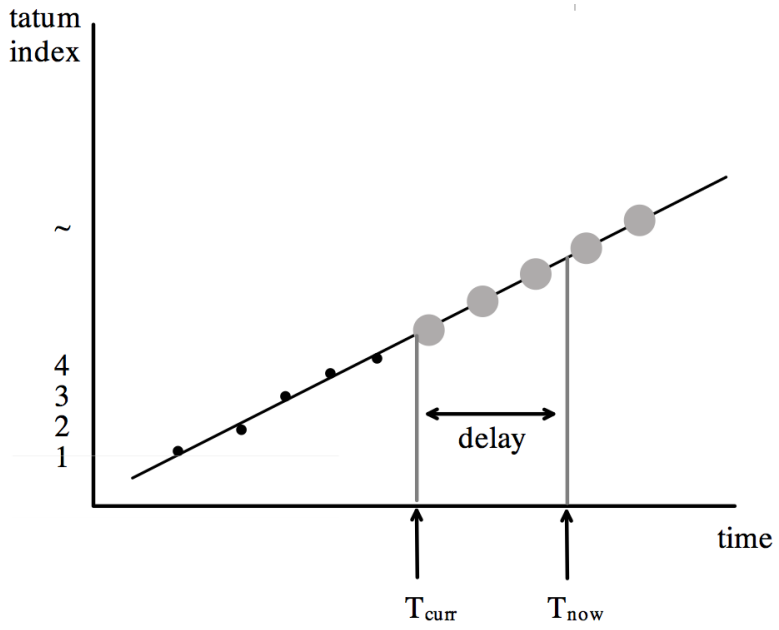

Figure 9 Illustration for tatum prediction with linear regression.

Figure 9 shows a snapshot when the real-world time is pointing at "T_{now}" and the system has generated the output at T_{curr}, therefore the look ahead of the system is $T_{now} - T_{curr}$. The black dots are the seen tatums generated by the algorithm so far, and the grey dots are the tatums locations predicted by linear regression. Note that the T_{now} pointer moves to the right continuously as it is real-world time; however, the processed T_{curr}

pointer moves to the right discontinuously depending on the step size of analysis window or buffers of each component in the system. Therefore, in order to compensate for the step size, the actual compensation must be greater than the worst-case look ahead, which is further illustrated in Figure 10.

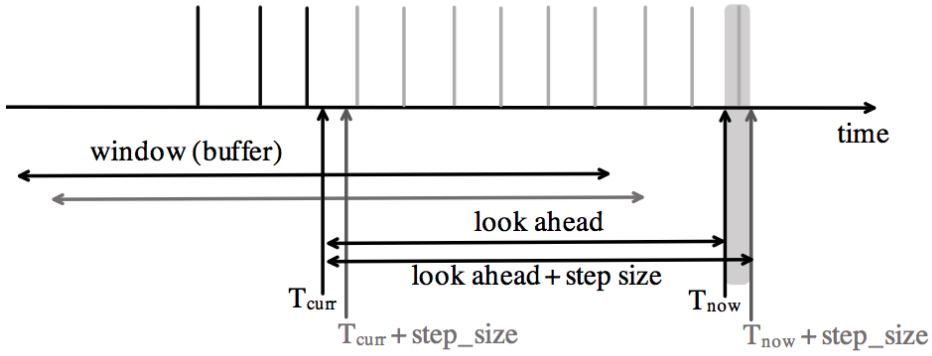

Figure 10 Illustration of extra compensation caused by the discontinuous step size of sliding window (or buffer).

In Figure 10, the black lines are the past estimations of tatums. Grey lines are the predictions; T_{curr} denotes the time instance of latest time frame from the previous stage, while $T_{curr} + step\_size$ denotes the time instance of the next time frame. Since the sliding window is not continuous, the system has to predict the output at t ($T_{now} \leq t < T_{now} + step\ size$) to meet real time requirement.

Because our output is downbeat predictions, it is not enough to simply predict tatums[4]. Recall that the downbeats are computed by decoding the hidden state of each tatum using the original HMM model (Section 4.8), which has a strong assumption that the state transition has higher tendency of staying in its group (tatums per measure). We apply the same assumption to the downbeat prediction logic as illustrated in Figure 11.

[4] Alternatively, we can directly apply linear regression on downbeat locations, but this will give coarser time granularity (time intervals of downbeats are larger), so this design is not evaluated.

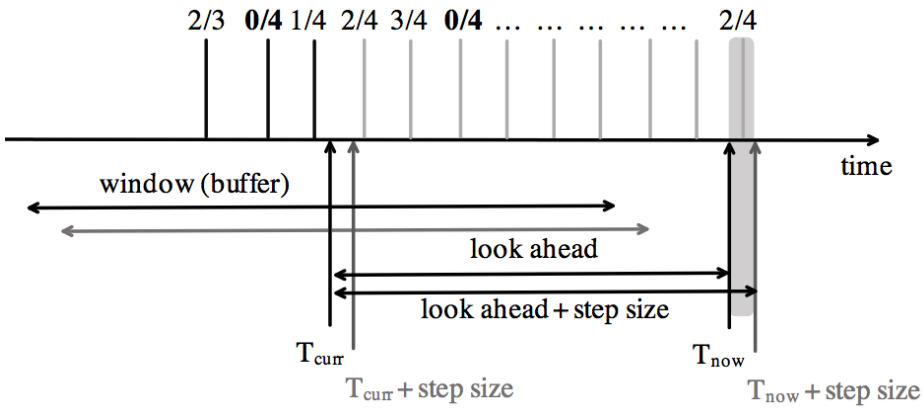

Figure 11 Downbeat prediction logic. The numbers above the vertical lines are the hidden states, where the downbeat states are in bold font. The future states are predicted by the hidden state of latest tatum, in this case it is "1/4".

5.3 Reducing latency

This section will discuss the modifications and the effect of reducing the look ahead and applying prediction to the accuracy of performance individually. Note the symbol **L** denotes look ahead components.

5.3.1 Tatum computation

L1. Reduce Tempo curve look ahead (*tatum Viterbi look ahead*)

The first step to enable online processing is to reduce the look ahead for tempo curve computation defined in Section 3.1.1.3. In this modification, the same Viterbi algorithm will be performed as each column of the tempogram matrix becomes available at each time step. So, the *tatum Viterbi look ahead* can be reduced from infinity to ~0 s.

The time complexity for standard Viterbi decoding is $O(N^2T)$, where N is the number of tempo bins and T is the number of columns of tempogram. In this modification, the computation will be the same in the way we append the max-score matrix (which costs $O(N^2)$ at each step), but the difference is that at each time step we update the back-tracking path (which costs O(T) for each step). Therefore, the computation time for each step is $O(N^2+T)$ which is linear to T (although the total sum for all time steps will be $O(N^2T+T^2)$). Here, T is the total time measured from the beginning of streaming. For online systems, this might cause problems because T

increases without bound, but this could be mitigated simply by performing back-tracking for a constant number of columns starting from the end.

For online computation, the original peak-picking algorithm (Section 3.1.1.5) is also modified. Because the PLP curve is synthesized one piece (grey area) at a time, as shown in Figure 4, instead of the whole sequence, the peak-picking algorithm can only see one chunk of PLP curve and therefore cannot determine if the sample at the right (future) boundary is a peak or not. In this implementation, the sample on the right boundary will be determined in the next time step, so the peak-picking result should be the same as the original one, but this correct result requires that we introduce another step size of look ahead.

L2. Reduce onset detection look ahead (*onset look ahead*)

The original system uses a large normalization window for the onset detection function computation. We attempt to replace this component with an existing state-of-the-art real-time onset detector that uses a uni-directional recurrent neural network [23], with source code available online [24]. This modification will reduce the look ahead from 2.5 s to about 20 ms (the STFT analysis window size) in this stage, which is neglectable in this thesis.

L3. Reduce tempogram analysis/synthesis look ahead (*tempogram look ahead*)

The tempogram toolbox uses a 6 second window by default in order to capture the slow periodicity of tempo, which corresponds to 10 bpm (minimum detectable) in the tempogram. However, in the original downbeat detection system, tempo less than 60 bpm is actually discarded in order to increase the recall rate. Therefore, ideally the window length can be decreased by a large amount without changing the output behavior of the tatum computation.

5.3.2 **Tatum evaluation**

5.3.2.1 Dataset

Since the tatum computation is relatively expensive (the computation time is only about 2 times faster than real time), the experiment is evaluated with a subset containing 10% of

the dataset (Table 5). The result is evaluated by the downbeat annotations, with a 70ms tolerance window.

Dataset	Ballroom	RWC Classical	GTZAN	RWC Genre	RWC Jazz	RWC Pop	RWC Royalty Free	The Beatles	Zweieck	Total
# Tracks	69	6	99	10	5	10	1	17	1	218

Table 5 Randomly selected subsets. Each column contains ~10% of the original datasets.

5.3.2.2 Effect of reducing look ahead (*L1, L2, L3*)

Table 6 shows the evaluation results of reducing the look aheads from tatum computation. First of all, the leftmost column of each group shows the comparison between original tatum computation (*offline*), after reducing the *tatum Viterbi decoding look ahead* and after reducing *onset look ahead* without modifying the *tempogram look ahead*. We can see the performance measures remain the same after reducing *tatum Viterbi decoding look ahead,* and the *look ahead* of the system can be reduced from infinity to about 8.5 s. Secondly, after the reducing the *onset look ahead*, the precision drops by 20%, the recall increases by 4% and the overall F-score drops by 19%. By visualizing the tatums computed by this configuration, we find that the frequency of the tatum becomes 2 to 4 times higher. This change of behavior is also reflected in the f-score and precision degradation. Third, we can see the effect of reducing the tempogram analysis/synthesis window length. The decreasing *tempogram look ahead* only causes slight f-score performance degradation when using the original onset detection algorithm, and causes the f-score drops by 38% from +2 s to +1 s when using the real-time onset detection algorithm.

Method	Offline	*tatum Viterbi look ahead* = 0 s *onset look ahead* = 2.5 s						*tatum Viterbi look ahead* = 0 s *onset look ahead* = 0 s					
tempogram look ahead (s)	+6	+6	+5	+4	+3	**+2**	+1	+6	+5	+4	+3	+2	+1
Avg. Recall	94.8	95.1	95.2	95.0	95.1	**94.9**	92.5	98.5	98.5	98.6	98.6	98.4	67.1
Avg. Precision	12.6	12.7	12.7	12.7	12.6	**12.6**	12.5	10.0	9.9	9.8	9.6	9.4	9.8
Avg. F-score	21.9	22.0	22.0	21.9	21.9	**21.8**	21.7	17.7	17.6	17.5	17.2	16.8	16.1
tatum look ahead (s)	Inf.	+8.5	+7.5	+6.5	+5.5	**+4.5**	+3.5	+6.0	+5.0	+4.0	+3.0	+2.0	+1.0

Table 6 F-measure of the tatum computed from different configuration (without applying prediction). The ground truth is the downbeat annotation, and tolerated window is 70 ms). The tatum look ahead is the total look ahead of tatum computation stage.

5.3.2.3 Effect of applying prediction

To discover the effect of *prediction* and *number of observation (last_n)* used for linear regression. We choose a fixed configuration where *Viterbi look ahead* is 0 s and *tempogram look ahead* is +2 s, with the original onset detection algorithm. We perform grid search on two control variables. The *amount of prediction* is searched from 0 to 10 seconds with a 1 second of interval[5] and the *number of observations* used for linear regression is searched from 2 to 10 seconds with a 1 second interval. The resulting f-score performance is shown in Figure 12[6].

[5] Note that the step size of the prediction is set to be 0.205 s, which is the same step size of the PLP synthesis window, so even when prediction is 0, the prediction logic is still predicting ahead by 0.205 s to compensate the extra delay caused by step size.

[6] Since the resulting degradation patterns for recall and precision look similar and the f-score represents the harmonic combination of recall and precision, we only present the f-score result.

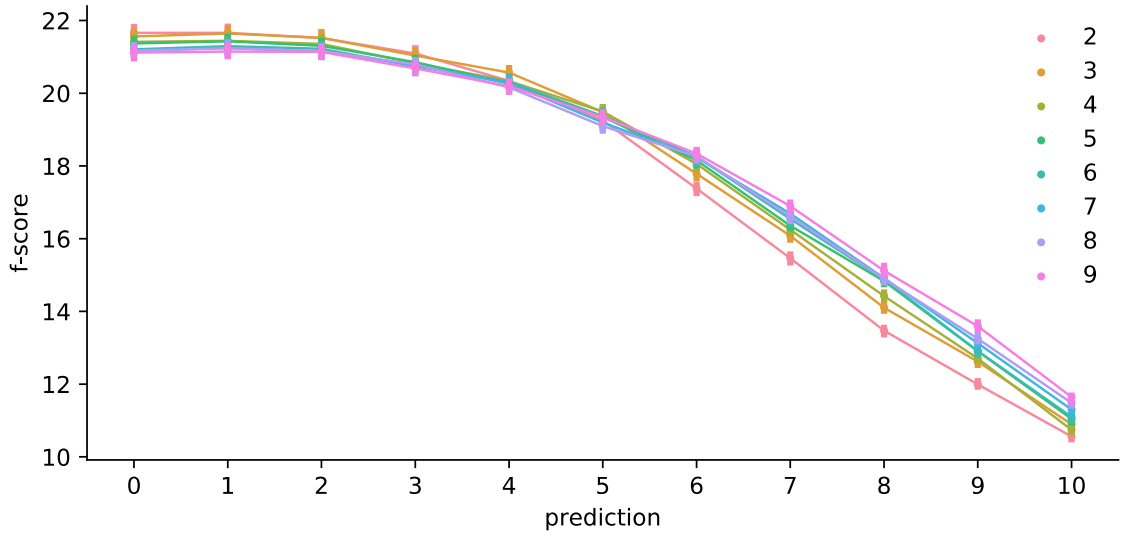

Figure 12 F-score with regard to the amount of prediction. The legend represents for the different number of observation in different color.

First of all, we can look at the sensitivity of performance to the *amount of prediction*. The f-score decreases as the amount of prediction increases for all *numbers of observations (last_n)*. The f-score remains approximately the same until the prediction is greater the 2 s, and drops by more than 10% after the prediction is greater than 5 s. Secondly, we can look at the *number of observations* versus the performance (color). The f-score performance almost behaves the same – fewer observations yield better performance when the amount of prediction is less than 5 s. On the other hand, after the amount of prediction is greater than 5 s, different observations gradually exhibit an inverse relationship. The larger *last_n* starts to yield better performance.

5.3.3 **CNN + HMM computation**

As described in Section 3.1.2, the rhythmic and the harmonic feature looks ahead by 8 tatums and 4 tatums respectively, and HMM Viterbi decoding is an offline algorithm. In this section, we will investigate the sensitivity of look ahead measured in tatums to the performance.

L4. Reduce HMM Viterbi decoding look ahead (*HMM Viterbi look ahead*)

In this modification, we introduce another control variable *HMM Viterbi look ahead*, which is the observation ahead (i.e. the sample points) used in back tracking. This variable is considered since the algorithm is essentially performing peak-picking on the output of neural networks (i.e. determine if a state is a downbeat or not), and looking ahead seems to be helpful to assess if the current observation is the downbeat (as illustrated in Figure 13.) On the other hand, one might expect the same kind of look ahead to be advantageous in tempo curve decoding. However, in this application of Viterbi, past information seems to be sufficient to impose a continuous property right up to the last observation, so look ahead is not used.

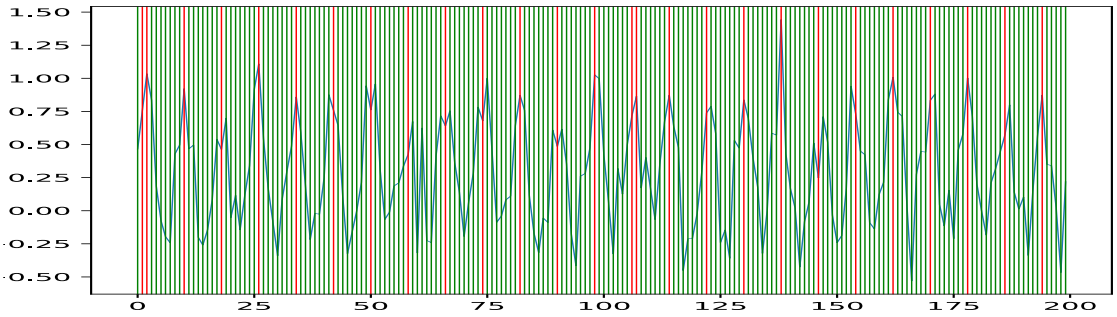

Figure 13 Online Viterbi decoding without looking ahead.

Figure 13 shows a synthesized downbeat likelihood as a function of time. The x-axis is the index of observation; the vertical lines are the decoded hidden states. The red lines are the downbeat states and the green lines are the non-downbeat states. We can see that without looking ahead, the algorithm cannot really determine the peak at the first 3 observations and around 110th observation.

L5. Reduce CNN look ahead

We want to discover the effect of reducing the look ahead of the CNN feature to the performance accuracy, therefore we apply a mask on the future context of the input feature (illustrated in Figure 14) to simulate the missing look ahead. Specifically, we parameterized the amount of mask by percentage. In the experiment, the mask is set to be *m* in {0%, 20%, 40%, 60%, 80%, 100%}, where 0% denotes no mask (i.e. the original case), and 20% denotes zero out the right-most 20% of the right-hand-side image (10 % of the total image) and so on.

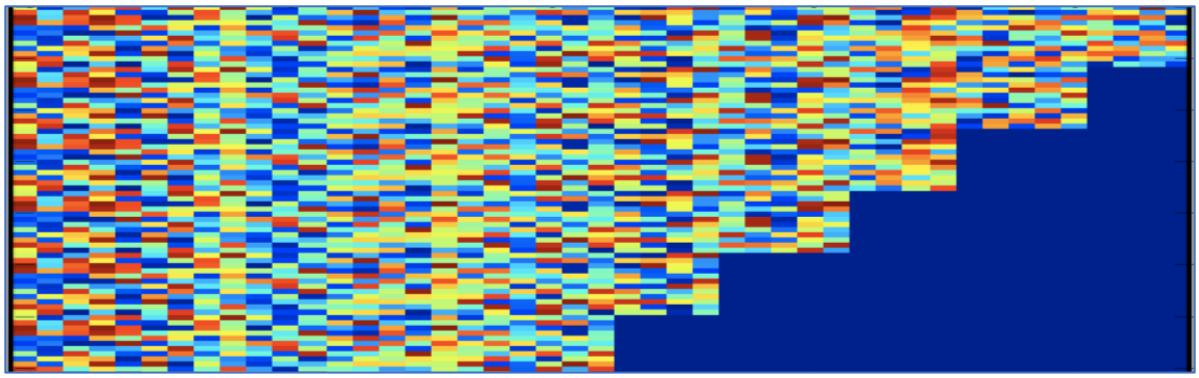

Figure 14 Mask future context from input feature. This illustrates the feature being masked by different amounts. There are 6 rows and the top row is the 2D images without mask, and second row is masked by 20% on the right-hand side, and so on. The bottom row has all of the right-hand side removed.

5.3.4 **CNN + HMM evaluation**

This section evaluates the effect of each modification to the performance accuracy individually. The evaluation setup is the same as what we did in the baseline benchmark (Section 4.9), with the same *offline tatums*, and we apply modifications as follows.

5.3.4.1 Effect of reducing look ahead (*L4, L5*)

Table 7 shows the evaluation results when reducing the look ahead of CNN+HMM computation. First of all, we can look at the sensitivity of *HMM Viterbi looking ahead* to the performance (left-hand-side of the table). The F-score performance degrades by 4% when we reduce the amount of look ahead in Viterbi decoding algorithm from infinity to 5 tatums. And from 5 tatums to 1 tatum, all performance measures are roughly the same. Another gap is between 1 tatum and 0 tatum, the F-score performance decreased by 4% again. This suggests even one step of *HMM Viterbi look ahead* is beneficial to the performance. Secondly, we can look at right-hand-side of the table, which shows the effect of reducing neural network look ahead. Interestingly, the performance degradation is not monotonic when masking out more future information - an optimal trade-off between latency and performance seems to happen at masking out 80% of the future context. In this configuration, the F-score degradation is only around 10% but we get 80% of latency reduction in this stage. Though it's hard to provide explanation of how neural networks work internally, this finding suggests the possibility for this CNN architecture to predict downbeat with lower latency with proper training.

Method	Offline	Reduce HMM *Viterbi look ahead*						Reduce HMM *Viterbi look ahead* Reduce *CNN look ahead (mask)*				
# HMM look ahead in tatum	Inf.	5	4	3	2	1	0	1				
% mask	0%							20%	40%	60%	80%	100%
Avg. Recall	79.8	76.4	75.9	75.9	76.7	**76.5**	72.4	69.1	65.4	30.6	**81.2**	14.0
Avg. Precision	54.0	51.4	51.6	51.6	51.0	**50.8**	49.5	53.0	54.1	57.4	**41.0**	33.6
Avg. F-score	60.9	58.4	58.3	58.3	58.2	**58.1**	55.8	56.7	55.0	36.1	**52.0**	18.0
HMM + CNN. look ahead in tatum	Inf.	5+ 8	4+ 8	3+ 8	2+ 8	1+ 8	0+ 8	1+ 8x80%	1+ 8x60%	1+ 8x40%	1+ 8x20%	1+ 0%

Table 7. F-measure for downbeat computed from different configurations in reducing look ahead. The ground truth is the downbeat annotation, and the tolerated window is 70 ms. HMM+CNN look ahead is the total look ahead in this stage.

5.3.4.2 Effect of applying prediction

We set the configuration to be fixed at HMM Viterbi look ahead = 1 tatum, *CNN feature mask* = 80%, as it seems to be an optimal point for *look ahead* and performance, and do grid search on the amount of *prediction* and the *number of observation* used for linear regression. Note that the step size[7] is also set to be 0.205 second for downbeat prediction logic. The evaluation result is shown in Figure 15.

[7]Though the step size of the tatums is not a fixed interval, we can imagine the prediction logic should still periodically (with fixed step size) probe the current the outputs from the system (even the new output is not generated yet) actively and perform prediction at this rate in order to satisfy real-time output.

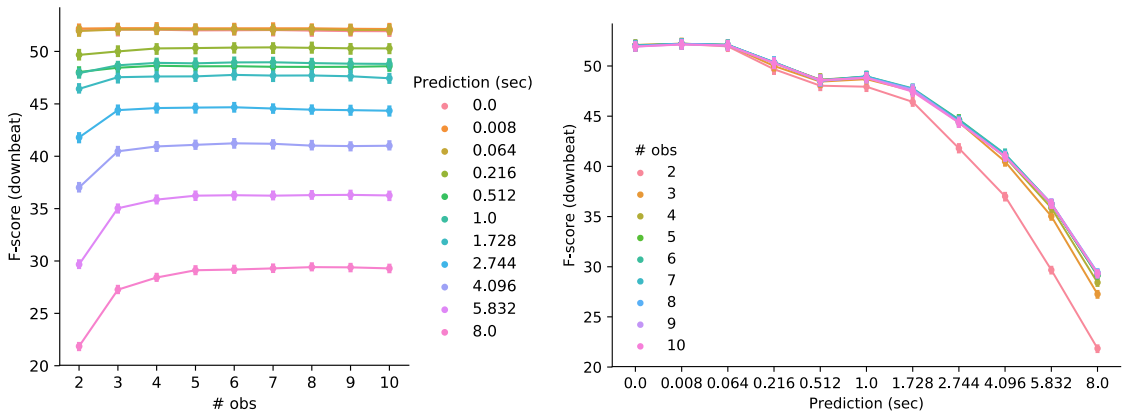

Figure 15 F-score with regard to number of observations (left), and the amount of prediction (right).

First of all, we can look at the sensitivity of performance to the *amount of prediction* (figure on the right). The f-score performance degradation reaches 10% at around 1.728 seconds of prediction and 20% at around 4.096 seconds. Secondly, the sensitivity of *number of observation* with regard to the performance (figure on the left) seems to be small if we only consider the data that perform above 80% of best performance (We can see the lines on the top are almost flat with regard to *number of observation*). When the *number of observation is* between 4 and 10, it seems to have little impact to the performance.

5.4 End-to-end evaluation

This section evaluates the *end-to-end*, online downbeat tracking system with different configurations *(look ahead* and *prediction)*. We use the existing model without retraining the model, and the evaluation is the same as what we did in the last section.

In this end-to-end simulation, the mask that simulates the effect of look ahead is directly computed dynamically, for example, when look ahead is set to 3 s, it might correspond to 5 tatums ahead at t = 1 s, and it might correspond to 6 tatums ahead at t = 2 s (since the tempo speeds up at around t = 2 s). Therefore, we directly describe all look aheads in seconds instead of tatums. Also, the parallel components now can look ahead as much as possible as described in Section 5.1.

In this evaluation, we fixed the online tatum configuration to be {*Tatum Viterbi look ahead* = 0 s, *tempogram look ahead* = 2 s, *onset look ahead* = 2.5 s} which gives

4.5 s of look ahead, and fixed the *hmm_look_ahead (*The *HMM Viterbi look ahead)* to be 1 s. We perform grid search on two control variables: *nn_look_ahead and nn_prediction*, where the *nn_prediction* is the downbeat detection logic defined in Section 5.3.3, and *nn_look_ahead* is the amount of look ahead shared among the parallel components, namely the harmonic network and rhythmic network. Note that the we do not consider the low-level latency and the step-size effects.

Figure 16 shows the ***total latency*** of the system in terms of the f-score performance with different amounts of *nn_look_ahead* rendered in different colors. The *total latency* is defined as:

total *latency (s) = tatum look ahead (4.5 s) + hmm_look_ahead (1 s) + nn_look_ahead – nn_prediction*

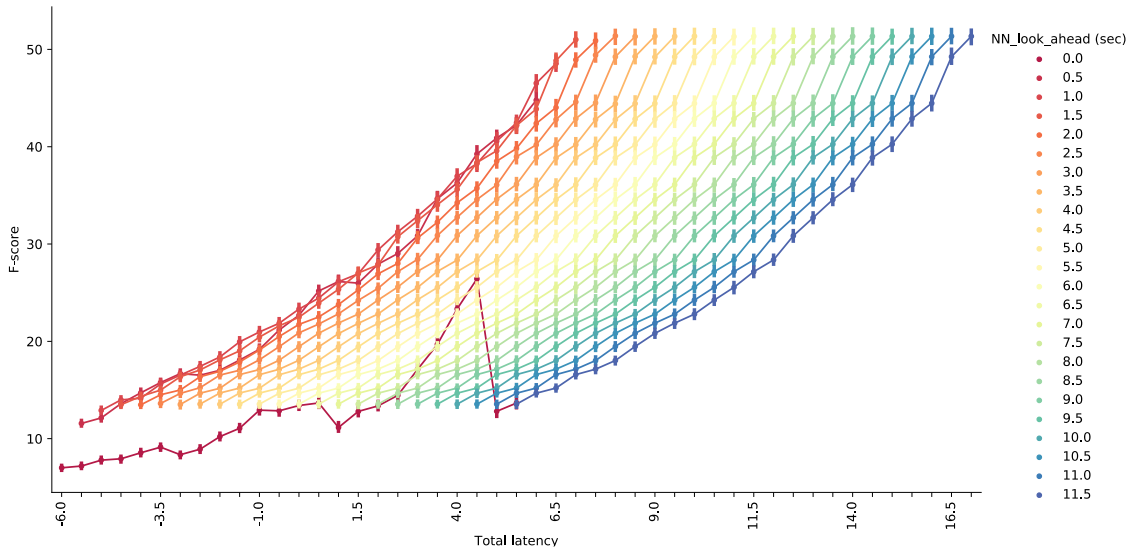

Figure 16 Grid search on nn_look_ahead and nn_prediction versus f-score performance.

First of all, we can see that each line in a different color constructs a curve with degrading f-score performance from top right to bottom left, which is the effect of applying *nn_prediction* (recall that prediction is needed in real-time systems to compensate for look ahead, but prediction reduces the performance as measured by the f-score). Since the degradation patterns after applying prediction are almost the same

among all *nn_look_ahead* (colors), the effect of applying *nn_prediction* can be further simplified to Figure 17 by aggregating *nn_look_ahead* in Figure 16. We can see the f-score to *nn_prediction* plot is a monotonically decreasing curve and the slope of the curve slightly decreases (in magnitude) as *nn_prediction* becomes larger.

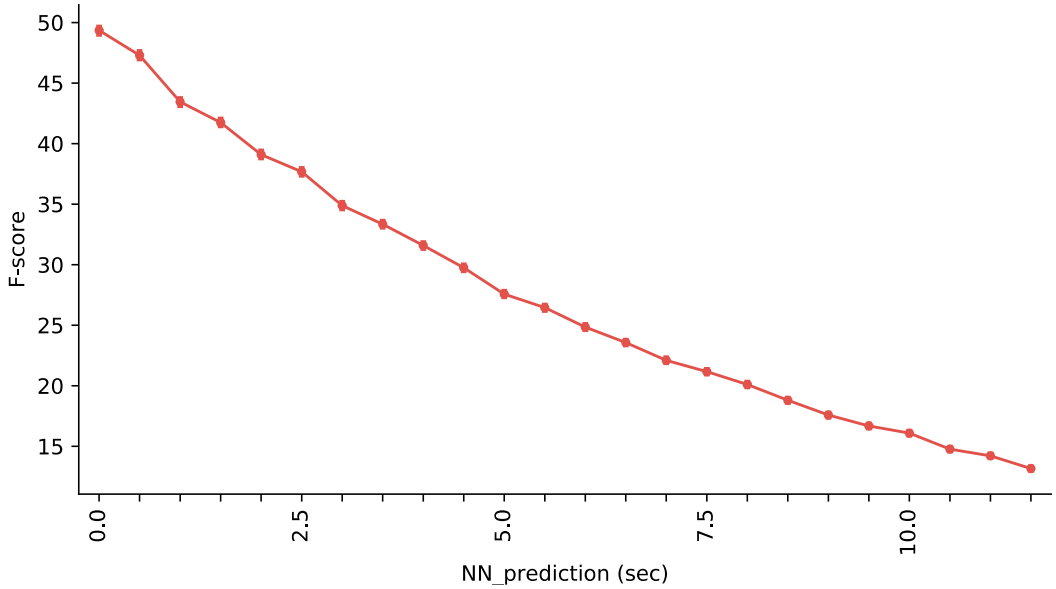

Figure 17 The effect of applying nn_prediction.

Secondly, Figure 18 shows the effect of reducing *nn_look_*ahead to the f-score performance when no *nn_prediction* is applied. We can see the curve has a much smaller slope compared to the f-score degradation caused by applying *nn_prediction* when *nn_look_ahead* is greater than 1 s. However, when *nn_look_ahead* is less than 1 s, the impact of reducing the *nn_look_ahead* suddenly becomes greater than applying *nn_prediction.*

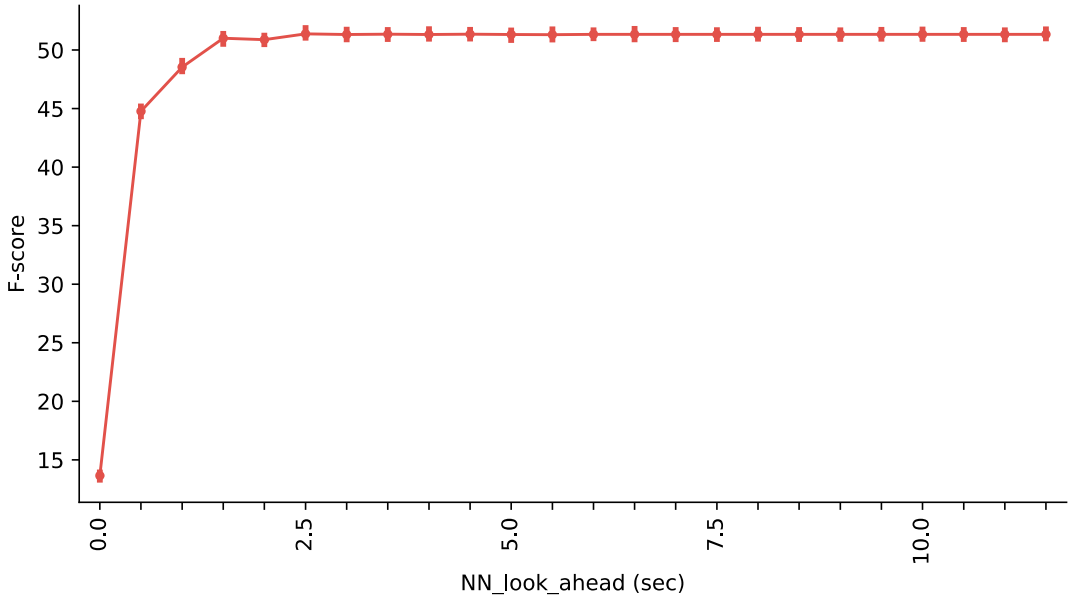

Figure 18 The effect of applying nn_look_ahead.

These observations suggest that in order to reduce the latency with lowest cost of performance degradation, it is better to reduce *nn_look_ahead* to about 1 s, then apply the nn_*prediction*. It costs almost no f-score degradation to reduce *nn_look_ahead* when it is greater than 1 s, but the cost suddenly become greater than *nn_prediction* after *nn_look_ahead* is less than 1 s.

Figure 19 further shows the trade-off between *nn_look_ahead* and *nn*_prediction to f-score performance when the *total latency* is fixed at zero (when the system barely meets *real time)*.

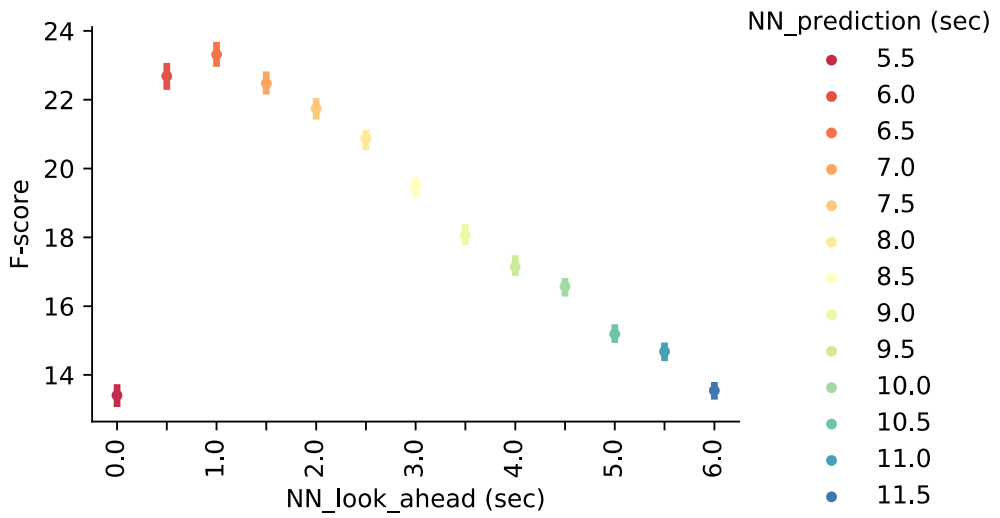

Figure 19 Trade-off between nn_look_ahead and nn_prediction at total latency = 0 s.

First of all, we are interested in the combination of *look* ahead and *prediction* that yields the best performance (the peak of the inverted U shape), which is *nn_look_ahead* = 1.0 s, and *nn_prediction* = 6.5 s (the x-axis denotes the *nn_look_ahead* and the colors denote different *nn_predictions* respectively). The resulting inverse U shape reflects what we observed before. We can see that when *nn_look_ahead* is greater than 1 s (the points on the right-hand-side of the peak), we cannot improve the f-score performance by increasing *nn_look_ahead*; however, in order to satisfy the same *total latency,* we have to apply *nn_prediction* in compensation of the extra *nn_look_ahead*, and this will cause a linear *performance* degradation with a slope of about 10% f-score for every 2.5 s. On the other hand, when the *nn_look_ahead* is less the 1 s (left-hand-side of the peak), we can see the trade-off between reducing *nn_look_ahead (*decrease the f-score performance) and reducing *nn_prediction* (increase the f-score performance*)* is that the f-score degradation caused by reducing *nn_look_ahead* is greater than the f-score increment caused by reducing *nn_prediction.*

This also explains why the performance degrades so much after the *total latency* is less than 6.5 s in Figure 16. The reason is that we have fixed the sum of *tatum look ahead* and *hmm look ahead* to be 5.5 s. So if we want to reduce the *total latency* to be 0 s, after we have reduced the *nn_look_ahead* to 1 s (at this point the *total latency* is 6.5 s and the best f-score among all configurations is about 50%, where *nn_look_ahead* is 1 s and

nn_prediction is 0 s), we have no choice but to apply 6.5 seconds of *nn_prediction* in order to compensate the look ahead, which will degrade the f-score by 25% according to Figure 17. As we can see, the peak f-score is about 24% when *total latency* reaches 0 s, which is close to our explanation (50% - 25% = 25%).

Secondly, we are interested in what is the best performance we can achieve under this real-time constraint comparing to our baseline (Thesis) implementation shown in Table 4, which is around 61% f-score. We can see the best f-score for real-time system Figure 19 is only around 24% f-score, which is around 37% of f-score degradation, making the accuracy unsuitable for real-world applications.

Third, we are interested in directions to improve the real-time accuracy for future work. On one hand, from the individual evaluations in previous sections, we have a sense of the sensitivity (slope) of f-score degradation caused by reducing each *look ahead*. Since we know that the performance degradation caused by *nn_prediction* is approximately linear at 10-15%/2.5 s, we can expect reducing other *look aheads,* if the cost is cheaper than applying *nn_prediction*[8]*,* can increase the f-score performance. For example, in tatum computation, we can reduce 1 second of *tempogram look ahead* almost with no cost, and also apply 2 seconds of tatum prediction with no cost. The *HMM look ahead* seems to have the same slope from 1 tatum to 0 tatum as *nn_prediction*, so it might not be beneficial to reduce *HMM look ahead*. Potentially we can reduce the *tatum look ahead* by 3.5 s, therefore reduce the need of applying 3.5 s of *nn_prediction* and therefore increase 15-20% f-score ideally. On the other hand, though currently the *nn_look_ahead* can already be reduced to about 1 s with only a little accuracy degradation, retraining the model with the masked feature might further push this number to 500 ms or smaller, as we can see, even without retraining, the trade-off between *nn_prediction* and reduce *nn_look_ahead* from 1 s to 500 ms is already close as shown in Figure 19.

Lastly, note that the evaluation of reducing *nn_look_ahead* is simulated by masking the feature without retraining the model for each amount of *nn_look_ahead*. The

[8] *A*ssuming the f-score degradation curve caused by *nn_prediction (slope)* still holds after reducing other look aheads.

performance degradation graph after retraining might behave differently, and this is a limitation of this study. However, we can assume from the original off-line work that look ahead is beneficial, so surely there will be *some* degradation, even if the CNN is retrained for lower look ahead. Furthermore, the degradation we measured from simple masking is rather small and smooth except for the anomalous value at 60% (see Table 7). This all suggests that retraining would achieve only small improvements, so we left this to future work.

6 Conclusion and Future Work

In thesis work we re-implemented a state-of-the-art downbeat tracking system, identified the latency bottleneck of the system, proposed several real-time adaptations for the original system, provided an evaluation and presented the sensitivity of each modification to the accuracy of performance. First of all, the re-implementation part achieves 76% of the original f-score performance on a single dataset. As shown in Section 4.9, the performance gap most likely comes from the training process and hyper-parameter tuning of the neural networks, and a minor gap might come from the details of various parts of the system. Therefore, *we believe that with proper pre-processing of the data, this stage could have a substantial improvement.* Secondly, in Section 5, we find that the latency of the tatum computation can be reduced by 2 seconds without compromising the f-score performance, and *it can potentially achieve real-time computation with about 15% of f-score degradation* (compared to the offline version) when prediction is applied as shown in Figure 12. Third, we see that when masking out the future temporal context in the neural network (without retraining the network after applying each mask), the performance degradation has an optimal sweet spot when masking out 80% of the future context (Section 5.3.4.1). Furthermore, also shows the sweet spot of look ahead is less than 2 s. This suggests the future information beyond 2 s might not be so important to the system and the *possibility of increasing the accuracy by directly retraining this neural network architecture with masked future information*. Fourth, we can see that for downbeat prediction (Figure 15), when the amount of prediction is less than 0.2 s (in addition, the step size is set to be about 200 ms), the prediction is fairly robust, *this is useful to compensate for the low-level latency in the systems*. Fifth, the efficiency of the algorithms should be considered when the system is taking in long streams of audio. In various stages of the system, such as Viterbi decoding, a practical implementation will need to impose an upper bound on the amount of retained data to avoid memory overflow. Also, the Viterbi decoding is an expensive step that, in practice, can be sped up by large amounts using a compiled language. (The current implementation uses an interpreted

version of Python). Sixth, from the experiment results shown in Section 5.3.2, we can see there seems to be substantial *room for improvement* in tatum computation latency (by about 3 s) without compromising the f-score performance, which could potentially increase the accuracy of the real-time downbeat tracking system.

7 Bibliography

[1] MIREX, "2016:Audio_Downbeat_Estimation_Results," [Online]. Available: http://www.music-ir.org/mirex/wiki/2016:Audio_Downbeat_Estimation_Results.

[2] J. Towns and al, "XSEDE: Accelerating Scientific Discovery," in *Computing in Science & Engineering*, 2014.

[3] Wikipedia, "Beat (music)," [Online]. Available: https://en.wikipedia.org/wiki/Beat_(music).

[4] J. Bilmes, "Timing is of the essence: Perceptual and Computational Techniques for Representing Learning and Repro ducing Expressive Timing in Percussive Rhythm," 1993.

[5] Echo Nest, "The Infinite Jukebox," [Online]. Available: http://labs.echonest.com/Uploader/index.html.

[6] SONY CSL Paris, "The Reflexive Looper," [Online]. Available: https://www.youtube.com/watch?v=VOc-ybfJeag&feature=share.

[7] A. Roberts, "Magenta wins "Best Demo" at NIPS 2016!," 2016. [Online]. Available: https://magenta.tensorflow.org/2016/12/16/nips-demo.

[8] S. Durand, J. P. Bello and B. David, "Feature adapted convolutional neural networks for downbeat tracking.," in *2016 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP). IEEE, 2016.*.

[9] S. Böck, F. Krebs and G. Widmer, "Joint Beat and Downbeat Tracking with Recurrent Neural Networks.," in *Proc. of the 17th Int. Society for Music Information*

Retrieval Conf.(ISMIR). 2016..

[10] F. Gouyon, "Ballroom dataset," [Online]. Available: http://mtg.upf.edu/ismir2004/contest/tempoContest/node5.html.

[11] D. P. Ellis, "Beat tracking by dynamic programming," *Journal of New Music Research 36, no. 1 (2007): 51-60.*.

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

[13] S. Dixon, "Evaluation of the audio beat tracking system beatroot.," *Journal of New Music Research 36.1 (2007): 39-50.*.

[14] N. Whiteley, A. T. Cemgil and S. J. Godsill, "Bayesian Modelling of Temporal Structure in Musical Audio," in *ISMIR. 2006*.

[15] F. Krebs, S. Böck, M. Dorfer and G. Widmer, "Downbeat tracking using beat-synchronous features to recurrent neural networks," in *ISMIR*, 2016.

[16] S. Durand, J. P. Bello, B. David and G. Richard, "Downbeat tracking with multiple features and deep neural networks," in *ICASSP*, 2015.

[17] P. Grosche and M. Meinard, "Extracting predominant local pulse information from music recordings," in *IEEE Transactions on Audio, Speech, and Language Processing*, 2011.

[18] A. Maezawa, "Fast and accuracy: Improving a simple beat tracker with a selectively-applied deep beat identification," in *ISMIR*, 2017.

[19] Katsouros, A. Gkiokas and Vassilis, "Convolutional neural networks for real-time beat tracking: A dancing robot application," in *ISMIR*, 2017.

[20] M.-P.-I. Informatik, "Tempogram Toolbox," [Online]. Available: http://resources.mpi-inf.mpg.de/MIR/tempogramtoolbox/.

[21] Wikipedia, "μ-law algorithm - wiki," [Online]. Available: https://en.wikipedia.org/wiki/M-law_algorithm.

[22] S. Durand, J. P. Bello, B. David and G. Richard, "Downbeat tracking with multiple features and deep neural networks," in *ICASSP*, 2015.

[23] S. Böck, A. Arzt, F. Krebs and M. Schedl, "Online Real-time Onset Detection with Recurrent Neural Networks," in *DAFx*, 2012.

[24] B. Sebastian, K. Filip, S. Jan, F. Krebs and W. Gerhard, "madmom: a new Python Audio and Music Signal Processing Library," in *Proceedings of the 24th ACM International Conference on Multimedia*.

[25] N. Degara and e. al., "Reliability-informed beat tracking of musical signals.," in *IEEE Transactions on Audio, Speech, and Language Processing 20.1 (2012): 290-301.*.

[26] M. E. Davies and S. Boeck, "Evaluating the Evaluation Measures for Beat Tracking.," in *ISMIR. 2014.*.

[27] D. J. Levitin, This is your brain on music, Atlantic Books Ltd, 2011.

[28] J. Schluter and S. Bock, "Improved musical onset detection with convolutional neural networks," in *ICASSP*, 2014.

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