Energy-Informed NMF for Polyphonic Piano MIDI Transcription

Ben Kevelson
University of Rochester
bkevelso@ece.rochester.edu

Teghan Murray
University of Rochester
tmurr16@ece.rochester.edu

ABSTRACT

NMF (Nonnegative Matrix Factorization) has many applications in audio signal processing, one of the most prominent of which is pitch detection. Using constrained dictionary elements for highly accurate polyphonic piano pitch detection in tandem with the timing information provided by the associated activation functions, it is possible to generate MIDI transcriptions that contain a wealth of information. Our proposed method uses NMF for pitch and timing of notes and spectral information to determine velocity and correct octave errors. This allows us to create more informative MIDI files. Our method is shown to output accurate pitch information.

1. INTRODUCTION

Automatic Music Transcription (AMT) is a group of tasks that attempts to take a musical audio input and automatically transcribe the notes, often to a piano roll. AMT is a field that encompasses many problems that audio signal processing algorithms attempt to solve, such as pitch, beat, tempo, and onset detection. This is part of what makes automatic transcription difficult; algorithms have to solve all of these problems in tandem. AMT is useful in many cases, including music creation, production, and music search engines [1]. Since it is made up of so many difficult problems, there have been many papers every year for decades that attempt to tackle AMT in some way. Pitch detection is a difficult problem because of the multiple harmonics that are present in each played note. Especially difficult is the task of polyphonic pitch detection, since the harmonics may overlap and interfere with peak detection. Additionally, different instruments have different harmonic contents, which adds to the complexity of the problem. For this reason, we choose to limit our method to piano recordings. Another task present in many pitch detection projects is the correction of octave errors. Since many pitch detection algorithms rely on frequency analysis, differentiating between different octaves of the same note is difficult since their theoretical harmonics are the same.

Music transcription today is dominated by neural networks (NNs) and NMF algorithms. NNs are useful for pattern detection, and can therefore be used to detect harmonics and perform pitch detection. NMF is an elegant solution, since it can perform pitch detection (by training each dictionary element to detect one pitch) and onset detection (in its activation elements) simultaneously. The greatest hurdle in NMF for automatic music transcription is that it is difficult to extract individual notes. Using random initialization along with any piano piece as the training audio will very rarely result in a set of dictionary elements, each of which represents a single piano note. We circumvent this problem by limiting our system to only transcribing one particular piano at a time. Grand pianos will have different harmonic elements than upright pianos, so it is very difficult to find dictionary elements that can generalize to both instruments. Therefore, we limit ourselves to a single piano instrument that our algorithm can learn specific dictionary elements for. This allows us to make our transcription more accurate, while sacrificing generalization. Our proposed method is to take state-of-the-art NMF for polyphonic pitch detection and to create a highly informative MIDI file using spectral information. In Part 2, we discuss our dataset. In Part 3, we discuss our method of polyphonic pitch detection using NMF. In Part 4, we discuss the MIDI transcription algorithms. In Part 5, we discuss our results.

2. DATASET

We could, theoretically, train our method to work for any piano instrument, as our updated method learns the template vectors of each note through the iterative process. This means that we can use an audio recording as our input to transcribe, but doing this alone presents some challenges in terms of results. Any audio signal used as input without a corresponding MIDI file would be hard to compare to the MIDI output, as the audio signal does not have a comparable visual medium (such as a piano roll for MIDI) as well as corresponding numerical values. It is much easier to compare MIDI-generated audio signals to MIDI output, as we can get quantitative results to compare note values, timings, and velocities. For this reason, we generate all input audio signals using Arturia's Analog Lab V plugin, under the 'Japanese Jazz Studio' piano preset.. This allows us to compare the original MIDI roll used to generate the inputs with the calculated MIDI output, rather than comparing audio signals to MIDI. To summarize, we use the audio signal generated from MIDI roll as input to the NMF process, convert the calculated NMF vectors into our MIDI transcription, then compare the generated MIDI transcription to the original MIDI roll used to generate the input.

Energy-Informed NMF for Polyphonic Piano MIDI Transcription

3. NMF METHODS

NMF is the process of estimating two component matrices **W** and **H**, such that their combination **WH** is an approximation of some input matrix **V** (such that **V** ≈ **WH)**. An iteratively-calculated loss function, often metrics such as Euclidean distance and Kullback-Leibler (KL) divergence, is minimized via iterative update to one or both of the component vectors **W** and **H**.

$$W_{ia} \leftarrow W_{ia} \frac{(VH^T)_{ia}}{(WHH^T)_{ia}}$$

$$H_{a\mu} \leftarrow H_{a\mu} \frac{(W^T V)_{a\mu}}{(W^T WH)_{a\mu}}$$

Figure 1. KL Divergence formulas for multiplicative update of **W** and **H** vectors in the NMF algorithm.

As implied by its name, the process includes the stipulation that all three matrices have no negative elements. This property of NMF is especially useful when considering audio applications, as using the magnitude spectrogram of an audio waveform as the input matrix **V** satisfies the inherent non-negativity of the process. This also serves to make the calculated component vectors easier to interpret as physical quantities. When considering the case of a spectrogram of a musical signal (in this case, a piano specifically), the matrix **W** can be construed as a series of templates for each note that is played in the signal, and the matrix **H** can be construed as the activations (durations) of each of these respective templates. This lends itself well to the MIDI transcription component of our work, as the template vectors can be made to be representative of discrete notes and the activation vectors can roughly describe velocity and duration of the note as it would exist in a MIDI format.

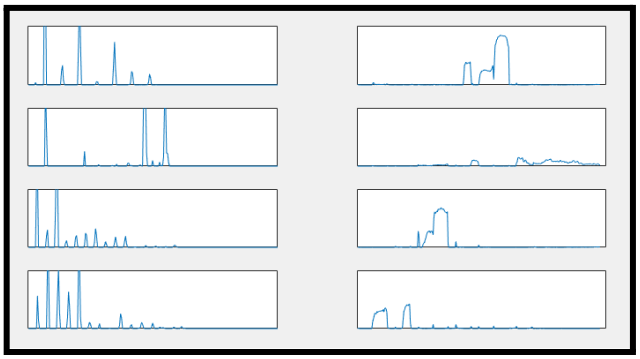

Figure 2. Examples of learned **W** and **H** vectors from NMF process. The **W** vectors (left) represent the harmonic spectrum of each note, and the **H** vectors (right) represent the activations of each learned note.

In the baseline NMF method, the matrices **W** and **H** are initialized randomly, and learn their respective vector formats via the aforementioned iterative update process. However, this baseline method has considerable drawbacks when considering our use case. When the component matrices are unconstrained and initialized randomly, the learned template vectors of **W** may not be representative of a single note – rather, the vectors may become representative of multiple notes, or no note at all. Additionally, the learned activation vectors of **H** may not be representative of the durations of their respective template vectors in **W**. Other errors have been known to arise in this process, such as octave errors (see Section 4.4), but there are methods to modify the baseline NMF process that mitigate these drawbacks.

In [2], a method is described for constraining the vectors of **W** by pre-generating an expected harmonic series for each note, widening the expected regions to account for inharmonicity, and leaving all other values as zero. The advantage of this is that the zero-values regions of the vectors will remain zero during the multiplicative update process, which encourages each of the template vectors of **W** to learn the structure of a single note. This also encourages the activation vectors of **H** to learn a more accurate description of the note durations, thereby making the entire process more accurate. We apply this constrained template vector method in our work. [2] also applies constraints to the activation vectors of **H** using existing information from a corresponding MIDI file for each input. This method further strengthens the accuracy and robustness of the method, but introduces the contingency that the system requires MIDI input for each file. We are unable to use this method in our application, as we want our system to be blind to MIDI input since it is creating MIDI output. Requiring a corresponding MIDI input for each input signal would make the project's purpose entirely redundant. However, we still seek to apply a method for constraining the activation vectors of **H**, but are unable to apply the methods used in [2].

To this end, we propose a method for pre-generating a rough structure for the activation vectors. We use an estimate of note onset and intensity decay based on the original magnitude spectrum input, as opposed to initializing the vectors of **H** randomly. The total energy at each frame of the linear magnitude spectrum is summed, which corresponds to the total number of frames represented in the learned vectors of **H**. We find that the spikes in energy at certain frames correspond to note onsets, but summing this information from the magnitude spectrum leaves us unable to manually discern which template vectors these peaks correspond to. Thus, we initialize *each* activation vector as the summed magnitude spectrum of the input, and allow the NMF process to iteratively learn which peaks correspond to which templates. This will give the vectors a basic structure during the learning process, similar to the methods described in [2], but does not require score information.

As described in Section 5, this also introduces a drawback in the form of increased octave errors, which makes sense since note space an octave apart are being initialized with the same activation vectors, meaning they are more likely to learn errors. However, our octave error correction is able to mitigate this to some extent, as described in Section 4.4.

4. MIDI TRANSCRIPTION

4.1 MIDI Formatting

MIDI is an obvious format to be used for music transcription. Most automatic transcription algorithms output some kind of piano roll, designating onset and offset times and the note being played. MIDI files are convenient because of their small file size and well documented formatting. They are also widely used in music production, especially as music instrument plugins improve in quality. A MIDI file is made up of a series of MIDI messages, each made up of three bytes; the status byte and two data bytes. The status byte indicates what type of message is being relayed (for example, relevant to our work are the NoteOn and NoteOff messages). For the NoteOn and NoteOff messages, the data bytes contain the MIDI note number and the velocity (which represents the relative volume of the MIDI note). The MIDI note number is determined by which dictionary element is being activated. The velocity we determine from spectral information, which is discussed in Section 4.3.

In order to create a MIDI file as our output, we first have to satisfy the requirements of MIDI formatting. All MIDI files begin with a MIDI header file, which contains metadata about the piece. The information relevant to our work are the BPM of the piece, the length of the entire MIDI file, and the ticks per quarter note. We can generate a header file that contains our desired specifications. After the header file, we insert the track information (containing the length of the MIDI file). Afterwards, all MIDI messages are listed. Each MIDI message is preceded by its timing, given in ticks. Within a MIDI file, all timing is presented in ticks, which determine the timing resolution of a MIDI file. The amount of time that a tick represents depends on the BPM of the MIDI file. Each MIDI quarter note is divided into ticks (typical ticks per quarter note values are 240, 480, and 960) [3]. Using the ticks per quarter note information, we can convert from timing (in seconds or samples) to timing in ticks. Finally, we end our MIDI file with an All Notes Off message, which will end any notes that are playing at the end of the piece, followed by the End of Track message present in all MIDI files. With all of this formatting information, we can create a MIDI file from a list of desired MIDI events [4].

4.2 MIDI Transcription

Using the trained dictionary elements, we can use the NMF to output activation functions for any given piano audio (provided it is the same instrument that we used to train the NMF). The simplest way to accomplish MIDI transcription is to set a threshold for the activation functions. When the threshold is passed (when rising) a NoteOn message can be generated, and when it is passed (when falling) a NoteOn message is generated. The MIDI note number is determined by the associated dictionary element, all of which are labeled. To create timing events in a MIDI file, it is necessary to convert the window number (from the NMF output) to seconds (using information from the STFT performed) and then to ticks (using the BPM and ticks per quarter note information provided in the MIDI header file).

We then generate a list of MIDI events. Each event must include the note number, if it is a NoteOn or NoteOff event, the timing between events (in ticks), and the velocity. This is then converted to MIDI format by taking the difference in ticks between events, then creating the event with the appropriate NoteOn or Off status byte message and data bytes.

The activation functions show prominent peaks when activated, and it is visually very easy to see where notes occur and what pitch they're at. However, there is some noise in the elements that aren't being activated. At polyphonic events, the activation functions still appear as peaks, but they tend to have a smaller amplitude. To fix this, we could just lower the threshold, but since there is some noise this will create extra, unwanted notes at the noisy peaks. To account for this we added a simple low pass filter at 20 Hz to get rid of some noise. We also choose to go a step further and perform adaptive thresholding. The threshold at each frame is set to be proportional to the total magnitude of the audio input at that frame. This prevents erroneous notes from occurring, and allows us to generalize the thresholding to any audio file. Raising the threshold at louder sections of audio prevents the noisy peaks from becoming notes. The combination of the LPF and the thresholding means works to prevent incorrect noteOn messages. . The timing and note number can be obtained from the NMF information, but we chose to further increase the amount of information present in our file by incorporating velocity.

4.3 Velocity

To increase the realism of our transcription, we wanted to also include velocity information. Our goal is to create a system that could determine the relative volumes of multiple notes that were played at the same time. To do this, we gathered information from a series of notes played at different velocities. We generated a MIDI file that played several quarter notes at the pitch C4, playing at different velocities (beginning at 120 and descending by a step of 10 to 20). lThis MIDI file was output as audio, then run through the same STFT that is used as the input to the NMF function. We then chose the audio frames immediately after the onsets of the notes. Converting to the linear magnitude spectrum, we visualized the frequency spectrum of the notes. We chose the values of the first 10 peaks occurring at integer

multiples of the fundamental frequency of the input piano note. We summed the values of these peaks for a range of velocity values, then calculated a line of best fit to allow us to approximate the velocity from the value of the sum of peaks. Using the linear magnitude values is useful because it is easier to calculate; the magnitude of the spectra will show only positive peaks (at multiples of the fundamental frequency), while the dB magnitude spectra will show peaks and valleys that are more difficult to detect. We also created lines of best fit for both the linear and dB magnitude spectra to ensure that information wasn't being lost. Testing with the dB magnitude spectrum was no more accurate than the linear magnitude spectrum was, so to reduce computational complexity and improve visibility, we kept the linear spectrum. The peaks of the linear magnitude spectrum and the values of the harmonics can be seen in Figure 3. The values of the peaks were determined using a fitted polynomial interpolation of the frequency spectrum 5 samples before and after the theoretical peak. This allows us to use only the peaks that are relevant to the note of interest. The interpolation is necessary because the frequency resolution of the STFT is not quite fine enough to give the index of the exact integer multiple of the fundamental frequency.

We found that an arctangent function provided the best fit for the relationship between the peak sums and the velocity. From our data, we saw that the curve flattens out at larger dB values. This also aligns with typical audio-to-MIDI values; it is rare that velocities greater than 110 occur. For these reasons, we used an inverse tangent function that flattens out around values of 120. Rounding is also necessary for MIDI file creation. Unfortunately MIDI velocities only have 7 bits of resolution (0-127), so it is necessary to round all calculated values to integers when creating the MIDI file.

4.4 Octave Error Correction

One aspect of pitch detection that is difficult to solve is octave errors. This is a common error that occurs in NMF pitch detection. [5] When solely depending on the NMF, octave errors occur because the dictionary elements for multiple octaves of the same note can be activated. When working on this paper, we encountered several instances where the incorrect harmonic had a greater activation than the correct harmonic. A difference of an octave is not a very large difference to the human ear, and the perception of a "wrong" note in the incorrect octave will not sound dissonant when played back [6]. However, our goal is to create highly accurate MIDI transcription and we have the advantage of having ground truth values to easily compare to.

Our proposed method to solve this problem is to exploit the fact that we are working solely with piano transcription. Piano notes have strong peaks at integer multiples of the fundamental frequency [7]. This means that within a given range of frequencies, higher frequency notes will have fewer peaks. Since we have an accurate guess as to what the note is from the NMF, we can generate a number of frequencies that the pitch could be at. We then detect the number of peaks that actually occur at these frequencies. Counting the number of peaks that actually occur will tell us what octave is actually being played (see Figure 1). We also have the advantage of only needing to select from 2-3 possible octaves, since the NMF is accurate enough to limit the selection.

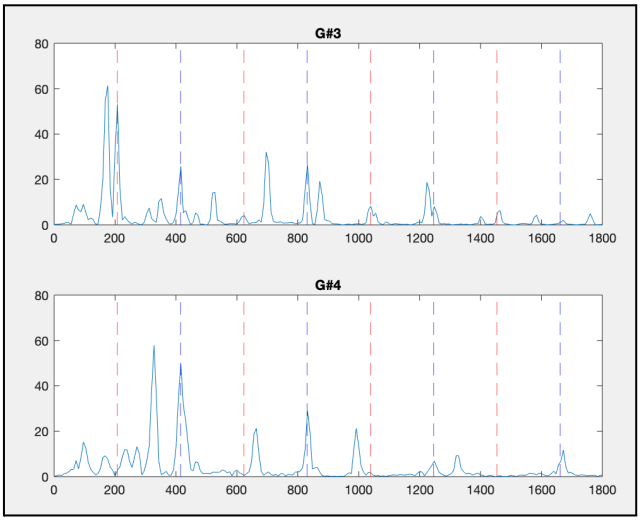

Figure 3. Comparison of peaks between two notes one octave apart. Note that the top spectra has peaks at all multiples of the fundamental (noted by dotted lines), and the bottom has peaks at every other fundamental.

5. RESULTS

Currently, our work has yielded moderately accurate MIDI transcription results. The NMF can be successfully trained to determine a highly accurate set of dictionary elements for a given piano (in our testing, provided by exporting audio from a high-quality MIDI instrument plugin), as well as moderately accurate dictionary elements for raw piano signal input (not generated with MIDI). Our MIDI transcription is capable of accurately determining timing, velocity, and note numbers. However, the timing of the NoteOn and NoteOff messages is still inaccurate. Our thresholding algorithm does not always accurately create new notes when the same note is played twice in a row. Instead it sometimes creates one long note (see Figure 4). The inclusion of our magnitude spectrum initialization for **H** vectors helps to alleviate some issues with the inaccuracy of NoteOn and NoteOff messages, but introduces other inaccuracies, mainly in the form of octave errors. The octave error correction methods described in Section 4.4 are able to mitigate this to some extent, but further work is required to find a tuned balance between these methods that creates output which mitigates all of these drawbacks.

The velocity values are also not exact. This presents another drawback in that it is not exactly accurate. However, since the purpose of our algorithm is to perform piano transcription from audio recordings, this is not necessarily a huge problem. We will be dealing with

much more dynamic audio files (not all notes being played at the same velocity), so the MIDI output will therefore be more dynamic and "human" sounding. The results are also visually deceptive, since our output looks much more messy than our input. This is because our output does not create highly accurate noteOff values. This is a less important issue to us, as discussed in the following section.

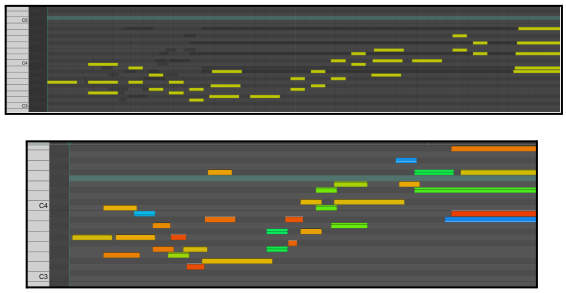

Figure 4. Comparison of ground truths (top) and our current MIDI output (bottom).

6. CONCLUSIONS AND FUTURE WORK

There are some improvements that can be made to increase the robustness of our code. The velocities of notes in chords are smaller than individual notes. Pulling more data from polyphonic chords and velocities could lead to the creation of another model for chord velocities. We are also hoping to improve the onset and offset detection for the timing of our MIDI notes. The onset detection can be improved by creating separate dictionary elements for the piano onsets (which have different spectral energies, due to the impact of the hammer on the piano strings). These separate dictionary elements could give more accurate onset timings. We are less concerned with improving the accuracy of the offset timings. Since we are specifically working to output a MIDI file to be used with a piano instrument plugin, the offset timings are less particular. A piano note that is notated as being 4 beats long and one that is noted as 2 beats long will sound very similar for most piano MIDI instrument plugins. Piano notes do not sustain unless the sustain pedal is depressed, which is a separate MIDI event.

Overall, we have succeeded in our goal of creating a system of polyphonic piano MIDI transcription. We can successfully train an NMF set of dictionary elements to reliably recognize pitches of any given piano, provided we have an audio file of that piano playing a chromatic scale. We can then use those dictionary elements and activation functions to generate a MIDI file that mimics the input audio file.

7. REFERENCES

[1] Benetos, E., Dixon, S., Duan, Z., & Ewert, S. (2019, January). Automatic Music Transcription: an Overview. *IEEE Signal Processing Magazine*, *36*(1).

[2] Ewert, S., & Müller, M. (2012). *Using score-informed constraints for NMF-based source separation*. IEEE Xplore. Retrieved December 5, 2022, from https://www.semanticscholar.org/paper/Using-score-informed-constraints-for-NMF-based-Ewert-M%C3%BCller/baa567da56f5ac88f24a628f147b0cb682a2a918

[3] *Convert MIDI files into MIDI messages*. Convert MIDI Files into MIDI Messages - MATLAB & Simulink. (n.d.). Retrieved December 5, 2022, from https://www.mathworks.com/help/audio/ug/convert-midi-files-into-midi-messages.html

[4] *Official MIDI Specifications*. The MIDI Association - Home. (n.d.). Retrieved 2022, from https://www.midi.org/specifications

[5] Sha, F., & Saul, L. K. (n.d.). *Real-time pitch determination of one or more voices by nonnegative ...* UC San Diego. Retrieved 2022, from https://cseweb.ucsd.edu/~saul/papers/nmf_nips04.pdf

[6] Wagner, B., Mann, D. C., Afroozeh, S., Staubmann, G., & Hoeschele, M. (2019, January 1). *Octave equivalence perception is not linked to vocal mimicry: Budgerigars fail standardized operant tests for octave equivalence*. Brill. Retrieved 2022, from https://brill.com/view/journals/beh/156/5-8/article-p479_4.xml?language=en

[7] Russell, D. A. (n.d.). *Hammer nonlinearity, dynamics and the piano sound*. Penn State College of Engineering. Retrieved 2022, from https://www.acs.psu.edu/drussell/Piano/Dynamics.html