# Early Investigations Into Real-Time Beatbox Resynthesis Using Non-Negative Matrix Factorization

**Noah Miller**
University of Rochester
nmill15@ece.rochester.edu

**Joe Bumpus**
University of Rochester
jbumpus2@ur.rochester.edu

## ABSTRACT

Beatboxing is a vocal performance technique in which a vocalist imitates percussive and melodic instrument sounds. This project aims to synthesize a sampled drum track based on a live percussive beatboxing performance using Non-Negative Matrix Factorization (NMF) in an interactive Python script. Given the spectral dictionary of any user's personal percussive elements, pre-loaded drum samples are triggered according to the decomposed activation matrix of the performance. The activation matrix directly returned from our NMF algorithm resulted in inaccurate sample triggering as it was not able to reliably separate drum sounds into distinct singular elements. To resolve this issue, we devised a system to compare local maxima across activation matrix elements. This returned activations corresponding to only one singular dictionary element and allowed for accurate sample triggering. We propose methods and considerations for real-time implementation of this algorithm.

## 1. INTRODUCTION

Beatboxing can be anything from a personal pastime to a live musical artform; a staple of the casual, impromptu musical jam or the driving rhythmic force of an a cappella performance. From a technical standpoint, beatboxing is an increasingly popular contemporary singing style where the vocalist imitates percussive drum and pitched musical instrument sounds. In this project, we focus specifically on the percussive elements of beatboxing and attempt to turn this purely vocal art into a digital sample controller using traditional source separation techniques.

Non-Negative Matrix Factorization (NMF) is a source separation method in which a spectrogram is decomposed into a dictionary matrix (representing the spectral pattern of each element) and an activation matrix (representing the active time of each element) [1]. Here, an input stream of human beatboxing is parsed through an NMF algorithm in order to acquire the activation matrix, which is then used to synthesize a corresponding audio track by triggering preloaded drum samples. This concept could prove extremely useful for musicians and producers with limited time or resources, facilitating quick form musical idea creation in a way that is far more natural, intuitive, and inspiring than manual drum programming in typical digital audio workstations.

In the following sections, we will briefly explain NMF and detail the methodology behind designing our interactive Python script. We will then discuss our experimental results before drawing conclusions on both the functionality and the user experience of our working solution. Finally, we will discuss the limitations of this current solution and considerations for implementing a similar system in real-time.

## 2. BACKGROUND: NON-NEGATIVE MATRIX FACTORIZATION

Given a matrix of non-negative data—a magnitude spectrum in our case, the set of which can be packed as columns into a non-negative $n \times m$ matrix $V$, where $n$ is the total number of spectra and $m$ is the number of their frequencies—NMF is able to summarize the rows of $V$ in the rows of an $r \times m$ matrix $H$ and the columns of $V$ in the columns of an $n \times r$ matrix $W$ [2]. This factorization is of the form

$$V \approx WH \tag{1}$$

where $W$ is the dictionary matrix showing the spectral content of each element, and $H$ is the activation matrix showing the temporal activations of the spectral vectors. The parameter $r$ sets the rank of the approximation and controls the power of summarization. Analytically choosing appropriate values for $r$ makes it possible to extract the major elements of the structure of $V$ [2].

There are multiple ways to measure the approximation, notably Euclidean distance and Kullback-Leibler (KL) divergence. KL divergence was used for the purpose of this project, which corresponds to the following iterative multiplicative update rules:

$$W_{ia} \leftarrow W_{ia} \frac{(VH^T)_{ia}}{(WHH^T)_{ia}} \tag{2}$$

$$H_{i\mu} \leftarrow H_{i\mu} \frac{(W^TV)_{a\mu}}{(W^TWH)_{a\mu}} \tag{3}$$

These update rules preserve the non-negativity of W and H and also constrain the columns of W to sum to unity [3].

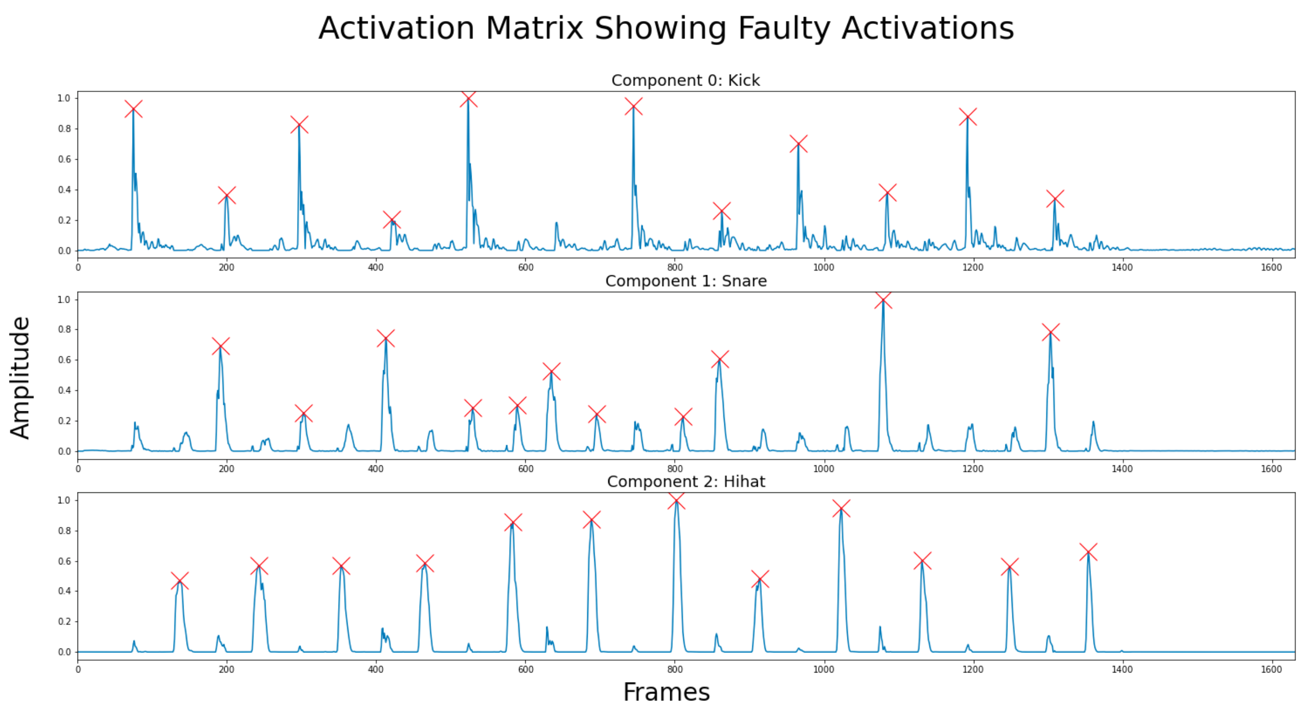

**Figure 1**: Faulty Activation Matrix

## 3. METHOD

### 3.1 User Interface

The program is run from a Python script and uses a text based, command line user interface. On launch, the script first prompts the user to record the three beatboxing sounds one at a time that will be used for the template vectors. Each time the recording is finished, the user will have the opportunity to listen back to the recording and re-record if necessary. Although this seems as if it is purely for the purpose of a positive user experience, this manual quality check by the user actually allows for more accurate results from the NMF algorithm later in constructing the output. Once all template samples are recorded, the script will prompt the user to record a ten (10) second snippet of beatboxing, using the sounds that it was trained on. All audio throughout the project is recorded using the Pyaudio library and then converted from bytes objects to Numpy arrays for processing.

### 3.2 Dictionary Training

The user's recorded drum elements are individually processed using NMF to acquire their respective spectral dictionary elements. While testing the program, we focused on three individual drum sounds: a kick, snare, and hihat. The NMF training algorithm uses randomly initialized vectors for both the W and H matrices and the aforementioned KL divergence multiplicative update rules. Since we are only using three sounds in our early tests, the NMF algorithm will only use three components. However, due to the nature of the NMF algorithm, components have no ordinal priority, making it difficult to keep track of which matrix element correlates with which components in a non-analytical way. As a result, we run each recording through its own individual NMF decomposition training process and then manually append them together, maintaining order within our program.

### 3.3 Activation Matrix of Beatbox Performance

With the now trained dictionary matrix, the algorithm can begin processing the beatboxing sample provided by the user. The goal is to retrieve the activation matrix for our three components. To achieve this, the NMF algorithm takes a randomly initialized H matrix, but uses our pretrained dictionary elements to initialize the W matrix. Once the processing is complete, we will have a calculated activation matrix for each of our components, allowing us to begin determining the locations of proper onsets.

With the activation matrix attained, a component corresponding to each element, we must determine when to trigger a preloaded drum sample. Intuition tells us that the simplest way to accomplish this would be to use a local maxima function with a set threshold to find the peaks of each activation matrix. Implementing this, however, exposes this simple solution as faulty. As shown in Fig. 1[1], while the local maxima function highlights the peaks in the plots, two problems are clear. The first is that an imperfect signal can cause quick successive false peak detections one after another, creating multiple undesired or false triggers. This issue can be solved fairly quickly using the SciPy

[1]All figures in this paper were produced during testing for the purpose of presentation. They are not part of the interactive Python script's output.

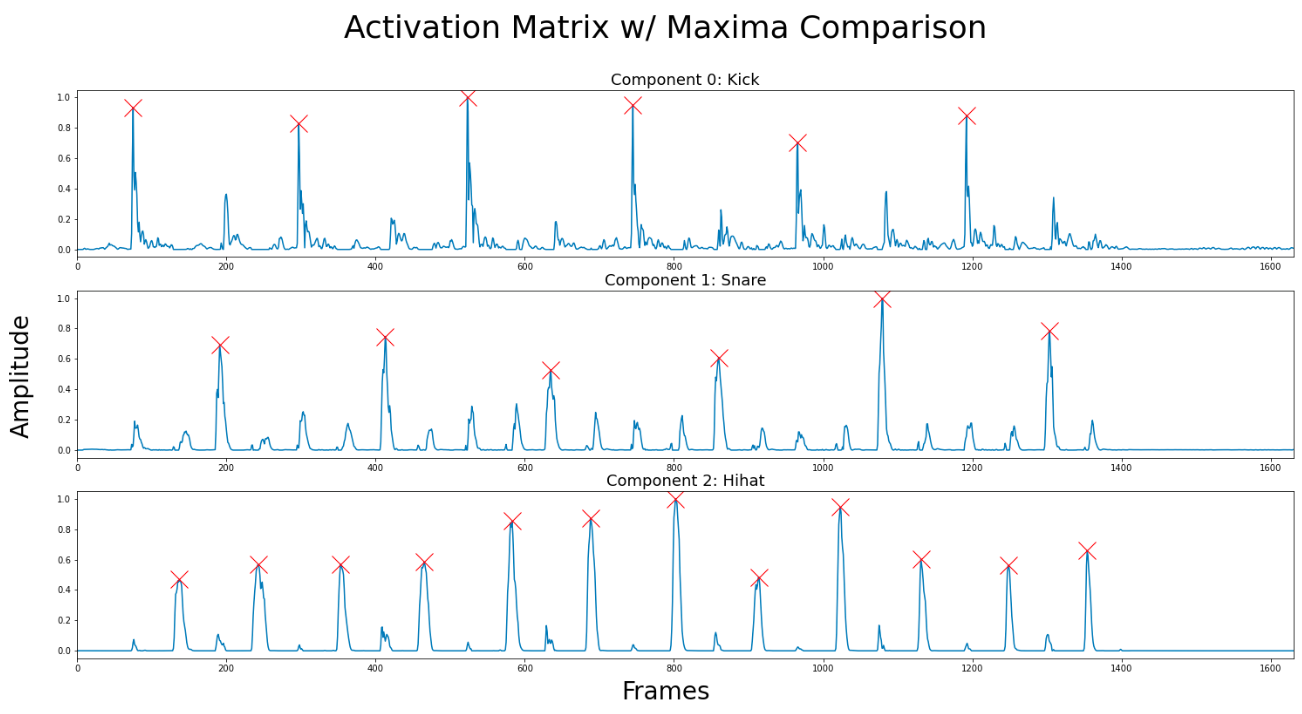

**Figure 2**: Activation Matrix with Amplitude Comparison

library's 'find_peaks' function, which includes a parameter to ensure that successive peak detections are past a minimum distance threshold [4]. The second problem, as we'll come to see, cannot be solved as easily.

### 3.4 Cross-Element Maxima Comparison

As seen in Fig. 2, there are often repeated false detections across different components—for example, a single performed 'snare' may trigger an activation in all three activation matrix elements even though it obviously can belong to only the snare. To compound this difficulty, the peaks across components often do not line up temporally in frame space, making it impossible to reliably compare these peaks within a singular given frame. While the distance parameter solved our previous issue, there is no comparable solution that considers multi-dimensional arrays such as ours, forcing us to develop a novel approach. The program first collapses all detected maxima into a single vector and sorts them numerically. For each maxima point within this combined vector, the following point is examined. If it is within a certain distance threshold, the algorithm will then attempt to differentiate the correct activation. The amplitudes of both detected maxima points are compared. The largest is marked as the correct detection, the other is removed, and the algorithm continues on. As shown in Fig. 2, the combination of these two methods produces an accurate detection of activations, giving us one cross-element activation per onset. We go through and filter out the false detections from each component's list of maxima and can move on to sample triggering.

### 3.5 Sample Triggering

Now that we have an accurate list of maxima detections for each component, we can restructure it into a form that can be used to trigger samples in temporal space. For each component, an array of zeros the length of the number of frames is created, and an impulse of 1 is placed at the index of the detected maxima. The next step is reconstructing our temporal data from this frame data. This, however, is quite straightforward, as we don't need to worry about any actual data beyond our 1 sample impulse and can simply use zero-padding to interpolate our signal. With our final triggering signal calculated, we can go through sample by sample, and play the corresponding preloaded drum audio files whenever an impulse is detected, giving us our final output signal.

## 4. EXPERIMENTAL RESULTS

Running this system with two distinctly different beatboxing patterns and vocal styles (one for each of us), we were able to analytically determine the aforementioned distance thresholds as 18 frames (using a frame length of 1,024 samples) within an activation matrix element and 10 frames across elements. In other words, a distance of 18 frames between local maxima in a component's activation matrix element solved the issue of false peaks, while staying within a distance of 10 frames in comparing detected maxima accurately solved the issue of one recorded sound showing as an activation for multiple components.

With our devised maxima comparison method, we were able to achieve very accurate component specific onset detection, as seen in Fig. 3, generated using a simple

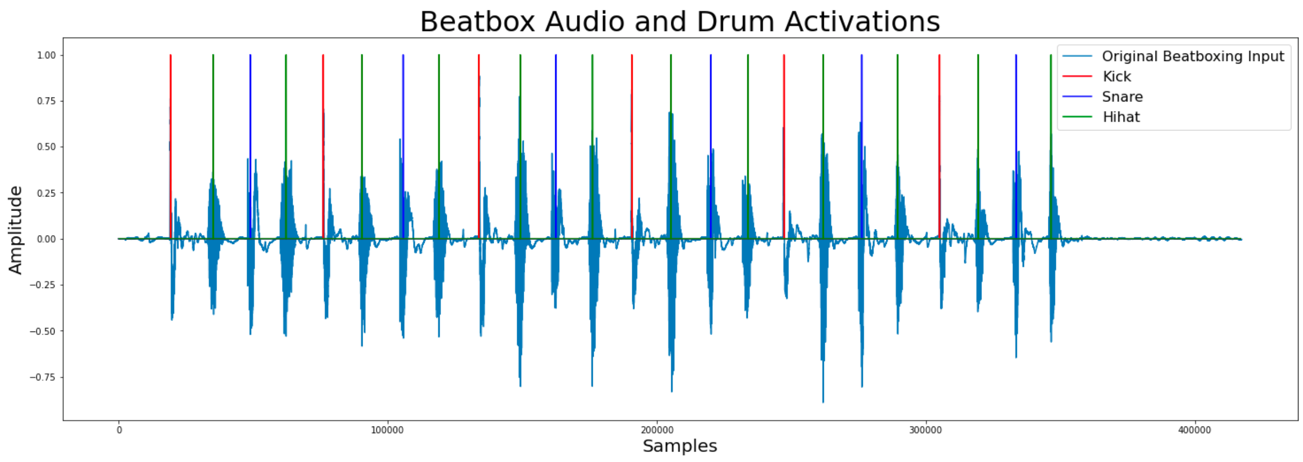

**Figure 3**: Detected Component Triggers

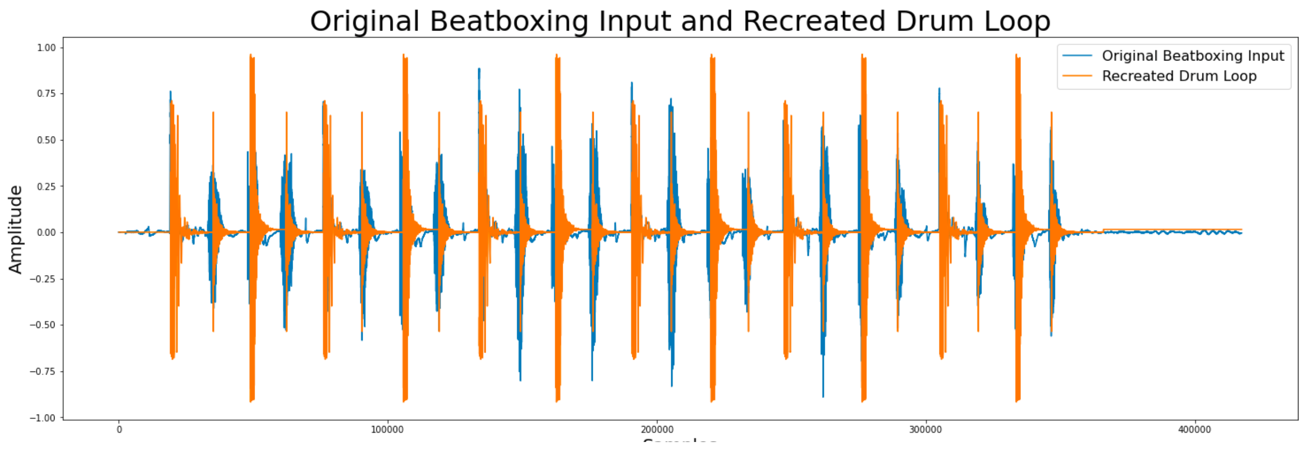

**Figure 4**: Final Output Comparison

beatboxing pattern. These activations—now clearly representative of only one sound—were then used to trigger corresponding preloaded drum samples and compile them into one output waveform, as shown in Fig. 4. In this case, we made the obvious choice to load a recorded kick drum sound for the 'kick' user element, a recorded snare for the 'snare' element, and a recorded hihat for the 'hihat' element. That being said, the chosen samples do not need to be similar to the recordings. As the training recordings are simply used to decompose the full beatboxing performance into an activation matrix and its local maxima, the activations could be used to trigger any sample, similar or not.

In testing, we regularly attempted to decompose each other's performed beatboxing loop with our own recorded individual beatboxing sounds. Notably, this nearly always resulted in improper activation detection. Although this may sound like a flaw in the design, this actually shows promise for future expanded use. Many advanced beatboxers perform with a variety of slightly varied but similar sounds; for example, they may have several low frequencies, quick plosive sounds, typically acting as kick drums in different styles. The inaccuracy shown in testing each other's recordings suggests that the NMF algorithm would likely be able to accurately detect multiple very similar drum sounds.

## 5. CONCLUSION

Although the current offline system is shown to be successful with our testing audio and shows promise for advanced use, there are a handful of factors that limit its effectiveness and real-time potential. Based on our experimental results, we have determined that an NMF based approach is simply not conducive to a real-time environment. Much of our algorithm, mainly the amplitude comparison for trigger detection, is based on some form of lookahead, which poses issues for online implementation. Any sort of lookahead would inherently have to introduce some form of latency to our system, which is the largest concern for performance. Based on the degree of lookahead we found necessary for the algorithm to effectively run, ranging between 10 and 20 frames, we would expect to see extremely noticeable latency introduced, rendering our program undesirable and unusable. Even without this issue, the NMF algorithm is an iterative process, requiring many iterations and updates to properly process the data, something that would complicate smooth real-time operation.

The offline performance exceeded our expectations considering the novelty of our implementation. NMF is primarily a source separation algorithm, not a source identification algorithm. This provided an interesting challenge as we aimed to essentially convert the NMF algorithm's purpose. For a working real-time model, we propose a sequence of source identification and onset detection, likely attainable through a properly trained neural network and simple spectral onset detection.  Considering the novelty of our project, we are very pleased with the results and are interested to see what more can be accomplished.

## 6. **REFERENCES**

[1] C. -Y. Cai, Y. -H. Su and L. Su, "Dual-channel Drum Separation for Low-cost Drum Recording Using Non-negative Matrix Factorization," 2021 Asia-Pacific Signal and Information Processing Association Annual Summit and Conference (APSIPA ASC), 2021, pp. 17-22.

[2] P. Smaragdis and J. C. Brown, "Non-negative matrix factorization for polyphonic music transcription," 2003 IEEE Workshop on Applications of Signal Processing to Audio and Acoustics (IEEE Cat. No.03TH8684), 2003, pp. 177-180, doi: 10.1109/ASPAA.2003.1285860.

[3] Lee, D., Seung, H., "Learning the parts of objects by non-negative matrix factorization," *Nature* 401, 788–791 (1999). https://doi.org/10.1038/44565.

[4] P. Virtanen et al., 'SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python', Nature Methods, vol. 17, pp. 261–272, 2020.