Non-Negative Matrix Factorization for Audio Signal Source Separation

Kiran Kumar
New York University - Steinhardt | M.M. Candidate - Music Technology | Dec 2016
kirankumar@nyu.edu

ABSTRACT

This paper focuses on the use of non-negative matrix factorization (NMF) to segment an audio signal into multiple components. NMF is commonly used in sound source separation (SSS), and this project aims to implement a preliminary NMF-based separation method. Accompanying MATLAB code reads an audio file, performs a Short-Time Fourier Transform (STFT) on the audio signal to convert it into the frequency domain, and performs NMF on the magnitude spectrogram to extract basis and activation matrices W and H. Multiplying these two matrices back together and converting it back into the time domain produces a signal whose content is very close to that of the original signal.

The following sections introduce relevant terms and concepts related to digital audio, discuss NMF in more detail, and outline a process used in the included MATLAB code to perform NMF on an audio signal.

1. INTRODUCTION

A single audio signal will often contain multiple sources playing simultaneously. Take a recording of a rock band for instance. Such a band will likely contain a guitarist, a bassist, a drummer, a vocalist, and a keyboardist, if not additional instruments. At any point in time in one of their songs, multiple instruments will be sounding at the same time, and while humans are very good at recognizing the sound of a guitar against the sound of a bass or a drum set, computers are not yet as proficient at this task.

Algorithmic sound source separation has many useful applications, including but not limited to: karaoke track creation; lyric transcription; voice recognition; detection of particular sounds in urban soundscapes (e.g. ambulances, construction, or gunshots); and budget-friendly recording and mixing of music in recording studios.

Non-negative matrix factorization (NMF) is one of the more popular methods of computationally identifying a source signal's individual parts. As the name implies, NMF factorizes a matrix into two other matrices, each of which has entirely non-negative components.

2. ABOUT NMF

The most simple equation/expression of what NMF does is

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

In other words, given an input matrix A, NMF produces two matrices W and H which, when multiplied back together, give an approximation of A. Again, A, W and H all contain entirely non-negative entries, and the process often works best when A is at least somewhat sparse. To produce an optimal W and H, an NMF implementation will initialize both of these matrices to random, positive numbers. The quality of the result of $W \times H$ is then evaluated using a **cost function,** and the matrices are iteratively updated until the cost function returns a result that falls within a defined threshold (i.e., is the approximation "close enough" to the original matrix A?). Lee and Seung mention two different types of cost functions [1], one of which calculates the Euclidean distance:

$$||A - B||^2 = \sum_{ij}(A_{ij} - B_{ij})^2 \tag{2}$$

and the other which looks at a divergence:

$$D(A||B) = \sum_{ij}(A_{ij} \log \frac{A_{ij}}{B_{ij}} - A_{ij} + B_{ij}) \tag{3}$$

For each method, $W \times H$ would be used as B, and both methods are lower bounded by zero (zero $\rightarrow$ $A = WH$). A paper by Kim and Park [2] discusses a common method that takes the Euclidean distance idea from (2) a little further by using an ***alternating* least squares** method. With this, two different calculations are performing in alternation, one of which keeps a fixed W and finds an optimal H, and the other which keep a fixed H and finds an optimal W, for example:

$$\min_{W \geq 0} ||H^T W^T - A^T||_F^2 \text{ (with a fixed } H) \tag{4}$$

$$\min_{H \geq 0} ||WH - A||_F^2 \text{ (with a fixed } W) \tag{5}$$

This paper discusses an application of NMF using MATLAB's built-in `nnmf` function, which uses the aforementioned alternating least squares method by default. More details are discussed in later sections.

3. DIGITAL AUDIO & NMF VARIABLES

This section discusses what the matrices A, W, and H actually represent in the context of audio source separation.

3.1 A: Source Spectrogram

A represents the frequency magnitude spectrogram of our input audio signal. When we first have our original audio signal (let's called this x), we have it in the *time domain*, where our x-axis represents time, and our y-axis represents amplitude. This is the most commonly used visual representation of audio, and it's a direct translation to how we hear sound. However, because this is a one-dimensional vector, and because audio in the time domain has both positive and negative amplitude values (representing compression and rarefaction of air molecules), we need to turn this into 2D, non-negative representation.

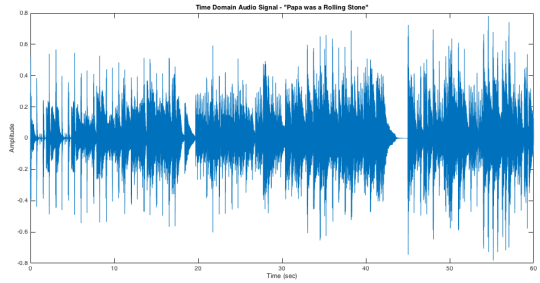

Figure 1. A time-domain representation of an audio signal.

To get the magnitude spectrum out of x, we need to do several things:

1. Buffer the one-dimensional, time-varying audio vector (usually a column vector) into a matrix of overlapping segments of time. Any two adjacent columns will then represent a very small increase in time, usually between 1-3 milliseconds. At this point, each row still represents a series of amplitude values (both positive and negative).

2. Perform a **Short-Time Fourier Transform** on the entire matrix. Another way of saying this is to take the **Discrete Fourier Transform (DFT)** (usually implemented as a **Fast Fourier Transform (FFT)** for computational efficiency) of each segment of time. This has the effect of transforming each column of time into a column of complex numbers that represent how much of various frequencies live within that chunk of time of the signal. At this point, each row now represents a different data feature. In this process, each feature is a different frequency value.

$$X(k) = \sum_{n=0}^{N-1} x(n) \times e^{(\frac{i2\pi kn}{N})} \tag{6}$$

Figure 2. General equation for the Discrete Fourier Transform

3. Take the magnitude of the values of the matrix. This is done by taking the Euclidean distance between the real and imaginary components of each complex number. For example:

$$\text{If } X_{ij} = 5 + 3i$$

$$\text{then } A_{mn} = mag(X_{mn}) = \sqrt{5^2 + 3^2}$$

Producing magnitude values of all of the matrix's values and plotting in MATLAB produces a "heat map" of sorts that shows how the amount of different frequencies in a signal change over time.

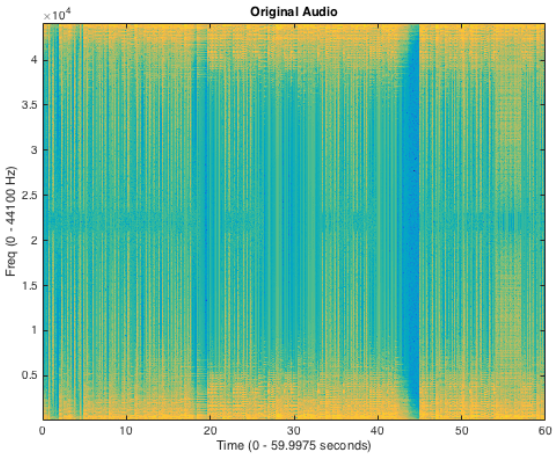

Figure 3. Magnitude Spectrogram of the same audio signal (scaled from raw magnitude to decibels to more accurately depict human hearing). The displayed color map range from magnitudes of dark blue (silence) to red (loudest).

3.2 W: Basis Functions

If A is your spectrogram matrix where the x-axis is time and the y-axis is frequency content, then W is a matrix that represents a collection of column vectors, each one of which represents a component of the full signal. For example, $W_{:,1}$ may contain mostly low frequencies and therefore indicate a bass instrument. The next column, $W_{:,2}$ may contain some mid-range frequencies that represent the sound of a woodwind instrument. In other words, each column represents a

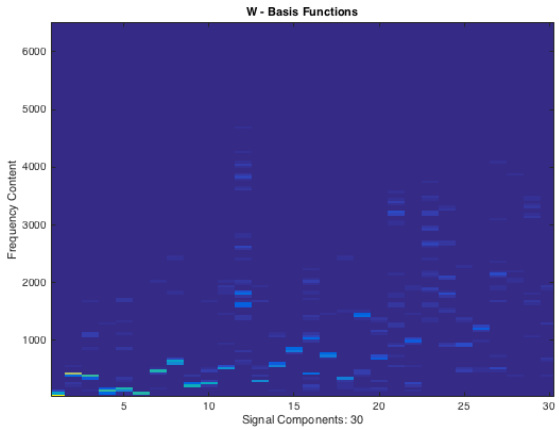

Figure 4. Matrix W: Basis functions for the matrix A.

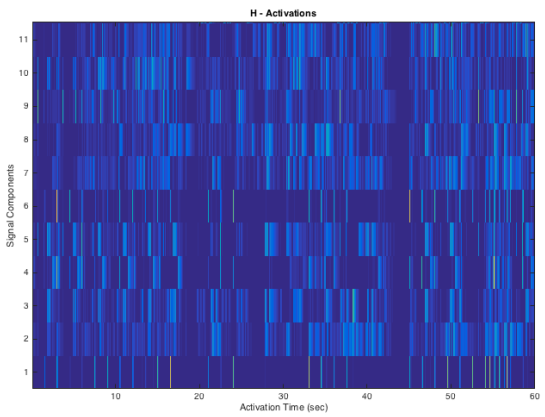

Figure 5. Matrix H: Activation timings/weighting coefficients for the components of W.

basis of A. The rows of W are therefore the same as the rows of A; each row represents a specific frequency value.

Because this divides the signal by frequency content, it is rare that a single column of W will ever correspond to the entirety of a tonal instrument's contribution to the audio. For example, if a piano played a total of 50 different keys throughout the course of a song, then a well implemented NMF may dedicate 50 different columns of W entirely to the piano, one for each note. Further refinement of a source separation process would probably cross reference those segments with a training set of other instrument recordings and classify those 50 notes as belong to the "piano" class, but that is beyond the scope of this paper.

3.3 H: Activations/Weighting Functions

While W represents the signal components, H represents how much each component contributes to the signal over time. For example, $H_{3,:}$ scales the component $W_{:,3}$, which effectively attenuates and amplifies that piece of the signal at the appropriate times. $W_{:,3} \times H_{3,:}$ therefore produces the full estimated spectrogram of the third component of the signal. If the first 50 columns of W correspond to the piano's 50 keys in the song, then one could extract the entirety of the piano from the rest of the song:

$$X_{piano} = \sum_{i=1}^{50} W_{:,i} H_{i,:}$$

Taking the Inverse Fourier Transform of X_{piano} (after a unbuffering process) would then produce an audio file that only contains the estimated piano parts of the full audio file. It's easy to then see how $W \times H$ therefore produces the estimated spectrogram of *all* components of the signal and how taking an Inverse Fourier Transform of that would let you hear an estimation of the entire song.

4. IMPLEMENTATION IN MATLAB

The attached MATLAB code tests NMF by doing the following:

1. Read an audio file into a matrix of amplitude values in the time domain. The default audio clip is a 30-second .wav file of "Papa Was a Rolling Stone" by the Temptations created with virtual instruments (drums, piano, and electric guitar).
2. Convert the signal into a matrix of frequency magnitude values over time called A.
3. Perform MATLAB's `nnmf` function on the magnitude matrix with a default `K` value of 30 unless otherwise specified. This returns a basis matrix W and an activation/weighing matrix H.
4. Calculate $W \times H$ and reverse the spectrogram process on the result in order to get a time domain signal back. This reconstructs the approximation of A back into a result that we can play and hear.
5. Calculate the residual and residual norms of both the magnitude spectrogram and the time domain reconstruction.

4.1 Read Audio

Given a file path, MATLAB's `audioread` function reads the data from a variety of audio file formats into a vector (or multi-column matrix) of amplitude sample values. Here, we use both return values of the signal `x` and the sampling rate `fs`.

```
[x, fs] = audioread(audiofile);
```

4.2 Convert to Frequency Domain

This process - converting to a magnitude spectrogram - takes several steps. Although MATLAB's built-in `spectrogram`

function is varied and powerful, it does not suit our purposes here, so custom-built STFT (short-time Fourier Transform) and inverse STFT functions were used to perform the conversion. Overall, the transform from time- to frequency-domain was as follows:

1. Force the signal to monophonic, i.e. average all columns to a single column.
2. Buffer the one-dimensional vector into multiple windows of 1024 samples that each overlap by 128 at the beginning and end.
3. Use a Hamming window to attenuate each 1024-sample column.
4. Take the FFT (Fast Fourier Transform) of each column to 1024 frequency values.
5. Take the absolute value (Euclidean distance) of each complex number of the resulting matrix to obtain the magnitude spectrogram.

4.3 Perform `NNMF`

Use MATLAB's `nnmf` function to estimate a basis matrix W and a weighing/activation matrix H from the magnitude spectrogram. Unless otherwise specified, use a default K value of 30 (components).

```
K = 30;
if nargin == 2
    K = kOverride;
end
[W, H] = nnmf(MAG, K);
```

4.4 Reverse Spectrogram

After calculating $W \times H$, the resulting matrix is fed back into a custom `iSpectrogram` function that essentially does the opposite of steps 2-5 from Section 4.2 in reverse order. More specifically, the function takes both the "calculated spectrogram" ($W \times H$) and the original complex data S from the original FFT and does the following:

- Calculate the magnitude of the original set of complex numbers S.
- Element-wise scale S by the ratio $\frac{W \times H}{|S|}$ to determine the new complex numbers to use.
- Perform an Inverse FFT on each column of the new complex matrix.
- Divide each column by the windowing function (Hamming) used in the original FFT
- Concatenate each column together into a single, one-dimensional column vector by incorporating the original window length (1024 samples) and overlap length (128 samples).
- Return only the real components of the result.

4.5 Calculate Norms & Residuals

For both the spectrogram (X) and the time-domain signals (x), we wish to know how much of a difference there is between the original (X_{org} & x_{org}) and calculated (X_{calc} & x_{calc}) data. For this, the program calculates the 2-norm of X_{org}, X_{calc}, x_{org} and x_{calc} as well as the 2-norm of the residuals $X_{org} - X_{calc}$ and $x_{org} - x_{calc}$. This was done on the normalized version of the audio signals so that only relative differences, rather than absolute differences, played a factor.

```
%Frequency Domain
normMAG = norm(MAG);
normWH = norm(W*H);
magRes = MAG - W*H;
normMR = norm(magRes);

%Time Domain
normX = norm(xSeg);
normTimeCalc = norm(resultTimeDom);
timeRes = xSeg - resultTimeDom;
normTR = norm(timeRes);
```

5. RESULTS

The program was run for 10 trials with the same song in order to see how varied the results were. For this particular song and settings, $||X_{org}||_2 = 2472.893349$, and $||x_{org}||_2 = 157.737111$. Remember the capital X represents the magnitude spectrogram data, and lowercase x represents the time-domain data.

	$\|\|X_{calc}\|\|_2$	$\|\|X_{org} - X_{calc}\|\|_2$
Trial 1	2860.825918	537.771291
Trial 2	3048.689455	732.019337
Trial 3	2819.085821	452.647369
Trial 4	2870.194229	509.913777
Trial 5	2702.795032	331.311178
Trial 6	2706.905441	327.193546
Trial 7	2869.861182	486.050132
Trial 8	2698.850051	351.151461
Trial 9	2798.592550	431.348163
Trial 10	2771.444447	389.945289
Mean	2814.724412	454.935154
St. Dev	106.8303	122.377096

Table 1. Magnitude spectrogram data for 10 trials

For the magnitude spectrogram, the norm of the reconstructed signal was definitely within the realm of that of the original signal, and norm of the residual of the two was just over 1 order of magnitude lower than either of the signals itself. As far as audibility of a signal goes, this residual is noticeable, but the signal itself is still relatively clear, especially for an unsupervised approach. The standard deviation, being less than

	$\|\|x_{calc}\|\|_2$	$\|\|x_{org} - x_{calc}\|\|_2$
Trial 1	99.017504	82.875553
Trial 2	103.857752	82.301764
Trial 3	98.783826	82.135328
Trial 4	100.763686	81.895214
Trial 5	100.379037	80.412670
Trial 6	103.277523	79.100738
Trial 7	106.974146	78.606113
Trial 8	102.859023	79.459732
Trial 9	104.613622	79.222886
Trial 10	105.508826	78.342239
Mean	102.6035	80.4352
St. Dev	2.7781	1.7124

Table 2. Time domain data for 10 trials

an order of magnitude smaller than the mean, shows that the results were fairly consistent despite the fact that NMF works by starting with random numbers.

The results for the time domain norms and differences were, at first glance, not as on par with the accuracy of the spectrogram results. The norm of the residuals were almost equal to that of the calculated results themselves. One might attribute these differences to the greater variability in time domain data as compared to frequency domain data. Sample values change between -1 and 1 so quickly that even a virtually inaudible time shift can have very large numerical results. Additionally, a small amount of accuracy is always lost when performing DFTs and IDFTs, and the custom built spectrogram and inverse spectrogram functions could have very easily produce mild inaccuracies that magnified other errors.

In listening to the reconstructed time-domain signal, the song was clearly recognizable, however much of the color of the original instruments (including transients and timbre variation in time) was lost in the synthesized version.

6. CONCLUSION & FUTURE WORK

In this paper, I have demonstrated the use of non-negative matrix factorization to decompose and subsequently reconstruct an audio signal. By asking MATLAB to find 30 bases for the spectrogram of a 30-second song clip, the program was consistently able to produce a very recognizable synthesis of the song, albeit with noticeable artifacts. By itself, NMF is a very powerful tool, and much research has been done on variants and enhancements of this process.

For my own research and work, future improvements could very easily include

- Incorporating a training set of instrument recordings for more automatic instrument identification
- More efficient and accurate cost functions
- Find ways to initialize W and H to reasonable values
- Include other known information about the recording (e.g. microphone and band setup, panning coefficients, instrumentation)
- Research potential NMF-like approaches for complex data (i.e. using FFT data directly rather than having to use a magnitude spectrogram)

Ultimately, I would like to implement an effective supervised or semi-supervised NMF process to assist with source separation for all-vocal groups, particularly contemporary a cappella groups. Because human singing voices are timbrally similar to one another, yet at the same time exhibit much greater variability in musical performances (e.g. slight pitch changes, vibrato, timbre changes, articulation of different syllables and phonemes), finding relatively few bases for a magnitude spectrogram of a vocal group would be an immense challenge. However, it's an exciting project to tackle, and given other pieces of information, I'm hoping to execute just that project as my final thesis in the Fall of 2016.

7. REFERENCES

[1] D. D. Lee and H. S. Seung, "Algorithms for Non-negative Matrix Factorization," *Advances in neural information processing systems*, pp. 556–562, 2001.

[2] H. Kim and H. Park, "Nonnegative matrix factorization based on alternating nonnegativity constrained least squares and active set method," *SIAM Journal on Matrix Analysis and Applications*, vol. 30, no. 2, pp. 712–730, 2008.