Nonnegative Matrix Factorisation

Dan Jacobellis, Tyler Masthay

Motivations for Nonnegative Matrix Factorisation : Audio Source Separation

Suppose we have a matrix $\mathbf{V}$ containing nonnegative data; for example, the time-frequency image of an audio recording.

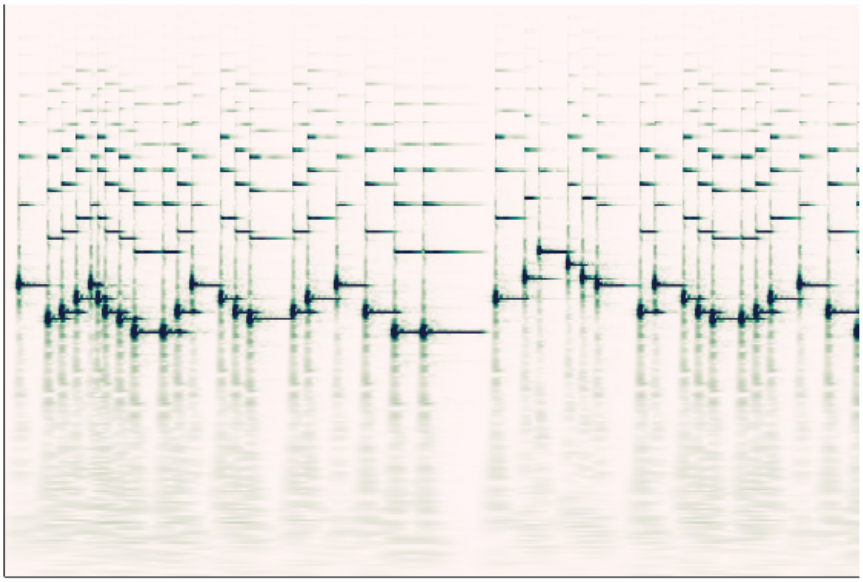

Figure 1: *Time-frequency representation of 'Korobeiniki' played on piano. The frequency is on a logarithmic scale.*

The problem of nonnegative matrix factorisation (NMF) amounts to factorising $\mathbf{V}$ into two factors $\mathbf{W}$ and $\mathbf{H}$ which are also nonnegative. That is,

$$\mathbf{V} \approx \hat{\mathbf{V}} = \mathbf{WH}$$

$$\mathbf{W} \geq \mathbf{0}$$

$$\mathbf{H} \geq \mathbf{0}$$

If the number of rows in $\mathbf{W}$ is restricted to be less than the number of rows in $\mathbf{V}$, then it may not be possible for $\hat{\mathbf{V}}$ to exactly match $\mathbf{V}$. By searching for a factorization that

approximately matches $\mathbf{V}$, we perform unsupervised learning, where $\mathbf{W}$ contains a learned dictionary and $\mathbf{H}$ contains a representation of $\mathbf{V}$ in terms of this dictionary.

This unsupervised learning method has proven to be effective for audio analysis, especially for source separation and automatic music transcription [1]. To understand why, consider the structure of the recording in ***Figure 1***. Although the recording consists of a single instrument (piano) playing a monophonic melody, the harmonics of each note create replicas of the actual melody at integer multiples of the fundamental frequency. The complexity that arises from mixing multiple intruments or adding any amount of polyphony makes direct analysis of the time-frequency image intractable for most tasks. Nonnegative matrix factorisation allows us to learn the harmonic structure of a mixture of sources, dramatically simplifying analysis.

Algorithms

The most widely used algorithm for NMF is the so-called multiplicative update rule based on the pioneering work of Lee and Sung [2]. The algorithm consists of the following steps:

- Initialize $\mathbf{W}$ and $\mathbf{H}$ with non-negative values
- Iteratively update $\mathbf{W}$ and $\mathbf{H}$ using the following rules: (n is the iteration)

$$\mathbf{H}^{n+1}_{[i,j]} \leftarrow \mathbf{H}^{n}_{[i,j]} \odot \frac{\left((\mathbf{W}^n)^\top \mathbf{V}\right)_{[i,j]}}{\left((\mathbf{W}^n)^\top \mathbf{W}^n \mathbf{H}^n\right)_{[i,j]}}$$

$$\mathbf{W}^{n+1}_{[i,j]} \leftarrow \mathbf{W}^{n}_{[i,j]} \odot \frac{\left(\mathbf{V}(\mathbf{H}^{n+1})^\top\right)_{[i,j]}}{\left(\mathbf{W}^n \mathbf{H}^{n+1}(\mathbf{H}^{n+1})^\top\right)_{[i,j]}}$$

where $\odot$ is elementwise multiplication.

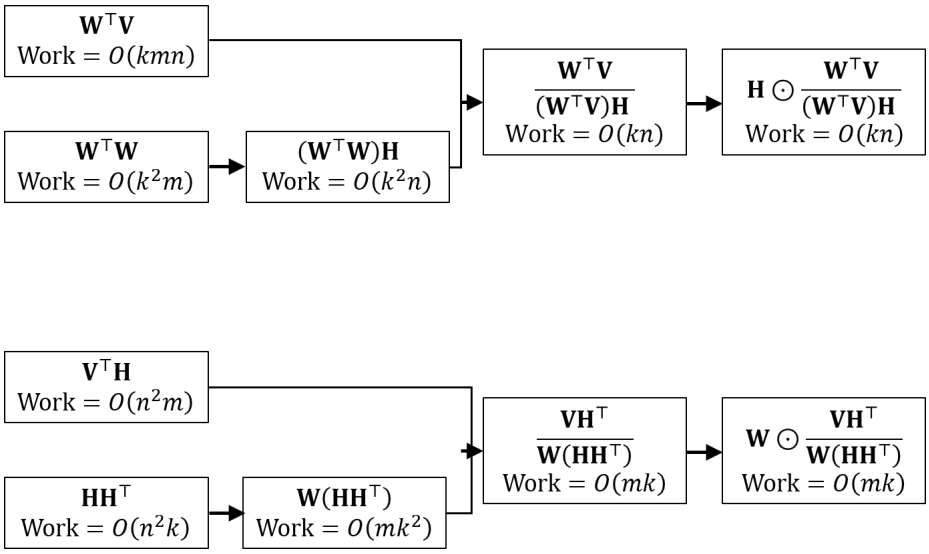

Figure 2: *DAG for the update of the W and H matrix showing the order of operations. We assume that* $\mathbf{W}$ *has size* $m \times k$ *and* $\mathbf{H}$ *has size* $k \times n$

We see that each iteration consists of multiple matrix-matrix multiplications, elementwise multiplication, and elementwise division. As a result, the algorithm is embarrassingly parallel. Furthermore, since each iteration uses the same matricies, the algorithm demands much more computation than memory operations.

Careful attention should be paid to the order in which the matricies are multiplied. Suppose that $\mathbf{V}$ has dimension $m \times n$, and that we are interested in approximating using a dictionary of k components, where $k < m$. Then, $\mathbf{W}$ has size $m \times k$ and $\mathbf{H}$ has size $k \times n$. Typically for audio, $m << n$, since the maximum number of frequency bins is limited by the Gabor limit for a desired time resolution. Choosing an ordering of the matrix-multiplications other than the order in ***Figure 2:*** will result in more work, since $n \gg k, m$.

Current State of the Art and Available Libraries

The most notable implementation of NMF available is part of the scikit-learn library, an open source collection of tools for statistical learning. This implementation offers two algothims: a coordinate descent algorithm and the multiplicative update algorithm. Each offers parallelization via OpenBLAS or MKL on some portion of the algorithm, but performance is not the primary goal of the scikit-learn implementation.

Other performance-oriented implementations exist, including GPU implementations; however, these are mostly target towards specific applications such as bioinformatics where scaling with the matrix size is important. For audio applications, we typically have a fixed, moderate size of the matrix $\mathbf{V}$, and the primary concern is runtime.

Parallelization on GPU : Motivation

Since the matricies $\hat{\mathbf{V}}$, $\mathbf{W}$, and $\mathbf{H}$ can remain stationary in memory across iterations, it is an ideal candidate for parallelization using a GPU. Since the audio applications of NMF become much more useful when they can be run on consumer hardware (or even mobile devices), GPU parallelization is our primary focus. (For example, a modern mobile phone has ~20 GFLOPS peak for its CPU but ~500 GFLOPS peak for its GPU). A typical drawback for GPU implementation is poor double precision performance compared to single precision. However, it is rare for audio to have more than 16 bits (fixed point), so it sufficient to implement the algorithm entirely in single precison floating point.

Parallelization on GPU: Implementation

Since the multiplicative update algorithm is naturally structured in terms of compute-bound operations (matrix-matrix multiplication) on data that remains stationary in memory, our goal is to find the most efficient way to structure calls (see ***Figure 2***) to the relevant linear algebra library (CUBLAS) and to minimize the time spent transfering between the host and device.

Our imlementation uses the metaprogramming abilities of Juliam which allowed us to generate the CUDA code listed in ***Figure 3*** for the update of $\mathbf{W}$ (the update of $\mathbf{H}$ is similar). Additionaly, we used CURAND to initialize $\mathbf{W}$ and $\mathbf{H}$ with random values directly on the GPU, which dramatically reduces the overhead compared to initializing the matricies on the host and sending to the GPU. After the number of iterations requested by the user is completed, the matricies are collected back to the device.

Figure 3: *Generated CUDA code for upate of* $\mathbf{H}$

Experiments

The scenario that we have chosen to benchmark our implementation is automatic music transcription (AMT). We begin 8 minutes (2^{25} samples) of uncompressed audio and apply a constant-Q transform [3] to produce an overcomplete time-frequency decomposition that splits each note on piano (from $A_1 = 55.0$ Hz to $A_8 = 7040.0$ Hz) into three frequency bins while retaining a lattice in the time-frequency plane that can be represented as a complex matrix of size 256×2^{18}. The magnitude of this matrix is what we will use for $\mathbf{V}$.

We then ran our NMF routine using a consumer GPU (GTX 970, 4GB memory), with varying number of components k and different sized chunks n of the audio. ***Figure 4*** summarizes the results. The main finding is that when the size of V exceeds what can be efficiently located on GPU memory, the performance begins to go down, since data must be transferred off the GPU *every* iteration. We also notice that, since the dimensions of the matrix

V are much different from each other, we are able to increase the number of components from 16 to 128 with almost no increase in runtime.

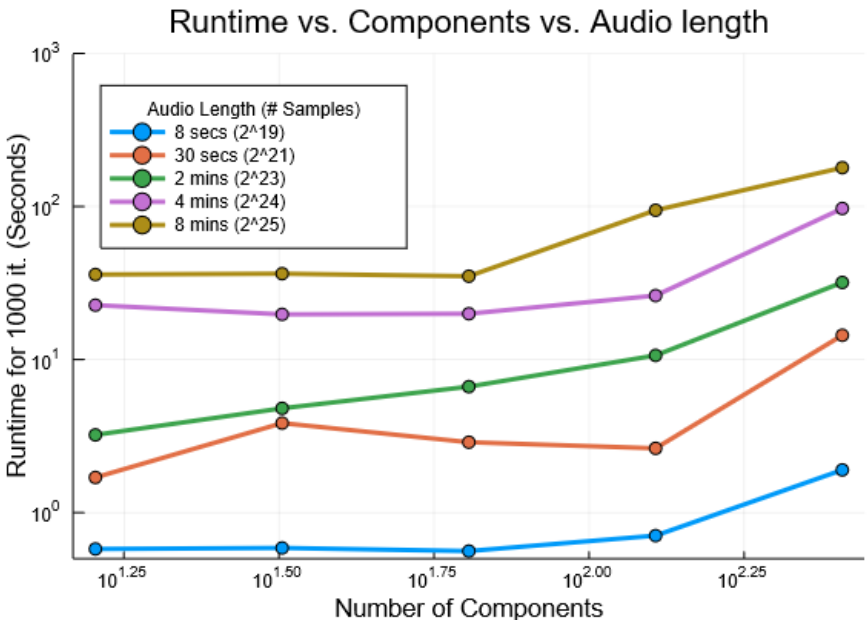

Figure 4: *Runtime for 1000 iterations of our GPU NMF implementation$*

Given our GPU performace, we then sought to find out how many CPU processes using the sci-kit learn NMF function that would be needed to match the GPU performance for a given problem size. These results are summarized in ***Figure 5***.

Figure 5: *CPU performance using scikit-learn$*

References

[1] S. Makino, Ed., Audio source separation. New York, NY: Springer Berlin Heidelberg, 2018.

[2] Lee, D.D., Seung, H.S., 2001. Algorithms for Non-negative Matrix Factorization, in: Advances in Neural Information Processing Systems 13. MIT Press, pp. 556–562.

[3] E. Vincent, T. Virtanen, and S. Gannot, Audio source separation and speech enhancement. 2018.