An Interior-Point Newton Algorithm for Non-negative Matrix Factorization

Eric Battenberg

December 12, 2008

Abstract

In this project, we propose a Newton step barrier method for performing non-negative matrix factorization. The algorithm is compared to a popular multiplicative gradient-based algorithm in its performance and efficiency using artificial data. The algorithm is then applied to audio source separation.

1 Introduction

Non-negative matrix factorization (NMF) is a blind source separation technique that has been successfully used in audio source separation, computer vision, and document clustering [1][2][3]. NMF can be phrased as an optimization problem in which we are given a matrix, $\mathbf{Y}$, with non-negative elements and wish to find non-negative factor matrices, $\mathbf{A}$ and $\mathbf{S}$, which minimize the error between $\mathbf{Y}$ and $\mathbf{AS}$. In basic applications, the error is computed using a Euclidean distance cost function.

$$D(\mathbf{Y}, \mathbf{AS}) = \|\mathbf{Y} - \mathbf{AS}\|_F^2 = \sum_{ij} (\mathbf{Y}_{ij} - (\mathbf{AS})_{ij})^2 \tag{1}$$

In certain applications, it becomes advantageous to compute the error using a cost function that takes into account certain aspects of the structure of the data and the desired output. For audio source separation, the performance of NMF is improved when using a matrix divergence cost function rather than the basic Euclidean distance function [1]. The matrix divergence is computed as

$$D(\mathbf{Y}\|\mathbf{AS}) = \sum_{ij} \left(\mathbf{Y}_{ij} \log \frac{\mathbf{Y}_{ij}}{(\mathbf{AS})_{ij}} - \mathbf{Y}_{ij} + (\mathbf{AS})_{ij} \right) \tag{2}$$

Using the above matrix divergence cost function as the objective to be minimized, we can phrase NMF as the following optimization problem:

$$\begin{aligned} \underset{\mathbf{A},\mathbf{S}}{\text{minimize}} \quad & D(\mathbf{Y}\|\mathbf{AS}) \\ \text{subject to} \quad & \mathbf{A}_{ij} \geq 0 \quad \forall i,j \quad \mathbf{A} \in \mathbb{R}_+^{N\times K} \\ & \mathbf{S}_{ij} \geq 0 \quad \forall i,j \quad \mathbf{S} \in \mathbb{R}_+^{K\times F} \\ & \mathbf{Y} \in \mathbb{R}_+^{N\times F} \end{aligned} \tag{3}$$

Where the dimensions N and F are determined by the size of the matrix to be factored, and K is chosen to be the number of sources we desire for the decomposition.

The above problem is not jointly convex in both of the optimization matrix variables; however, it is convex in each matrix individually. Lee and Seung have introduced an algorithm that alternates gradient-based multiplicative updates with respect to each of the two matrices to be optimized [4]. The updates are as follows:

$$\mathbf{S}_{\alpha\mu} \leftarrow \mathbf{S}_{\alpha\mu}\frac{\sum_i \mathbf{A}_{i\alpha}\mathbf{Y}_{i\mu}/(\mathbf{AS})_{i\mu}}{\sum_k \mathbf{A}_{k\alpha}} \qquad \mathbf{A}_{i\alpha} \leftarrow \mathbf{A}_{i\alpha}\frac{\sum_\mu \mathbf{S}_{\alpha\mu}\mathbf{Y}_{i\mu}/(\mathbf{AS})_{i\mu}}{\sum_\nu \mathbf{S}_{\alpha\nu}} \tag{4}$$

Lee and Seung show that the objective, $D(\mathbf{Y}\|\mathbf{AS})$, is non-increasing under these update rules. Because the above algorithm is much easier to implement than additive gradient-based methods and is less computationally complex than conjugate gradient methods, these updates have been widely used in audio-applications that employ NMF. The main drawback of this algorithm and any gradient-based algorithm used to minimize a non-convex function is that it will typically converge to local minima rather than an optimal global minimum.

To get around this local minima problem, we can run the algorithm multiple times with $\mathbf{A}$ and $\mathbf{S}$ initialized to different randomly generated values each time. Then we can select the result that achieves the smallest value of the cost function. These is also research being done to intelligently choose initial values for $\mathbf{A}$ and $\mathbf{S}$ in order to limit the number of times the algorithm must be run (ideally it would be one). In this paper we focus on how well NMF performs with random initializations.

In the following section, we propose an iterative algorithm that uses a logarithmic barrier function to maintain non-negativity of the iterates and performs Newton steps to minimize the cost function plus the barrier at each step. Along the lines of the Lee and Seung algorithm, our proposed algorithm alternates between Newton steps with respect to each of the two factor matrices. The barrier weight is then updated according to the accuracy of the previous Newton step. The specifics of the algorithm are covered in the following section.

2 Algorithm

2.1 Cost functions with barriers

In order to enforce non-negativity of the matrices $\mathbf{A}$ and $\mathbf{S}$, we employ a logarithmic barrier function.

$$\phi(\mathbf{x}) = -\sum_i \log(x_i) \tag{5}$$

Our new cost functions with respect to $\mathbf{A}$ and $\mathbf{S}$ then become:

$$f(\mathbf{A}) = tD(\mathbf{Y}\|\mathbf{AS}) + \phi(\mathbf{A}) \tag{6}$$

$$f(\mathbf{S}) = tD(\mathbf{Y}\|\mathbf{AS}) + \phi(\mathbf{S}) \tag{7}$$

2.2 Newton step

The Newton step with respect to a function $f(\mathbf{x})$ at the point $\mathbf{x}$ is:

$$\triangle \mathbf{x}_{\mathrm{nt}} = -\bigtriangledown^2 f(\mathbf{x})^{-1} \bigtriangledown f(\mathbf{x}) \tag{8}$$

So we must compute the Hessian and gradient of each of the cost functions with respect to $\mathbf{A}$ and $\mathbf{S}$ (eq. 6). We notice that the cost functions are separable with respect to the rows of $\mathbf{A}$ and with respect to the columns of $\mathbf{S}$. To see this we write:

$$f(\mathbf{A}) = \sum_i \left(t_{\mathbf{A}} \sum_j \left((\mathbf{a}^{(i)}\mathbf{S})_j - \mathbf{Y}_{ij} \log(\mathbf{a}^{(i)}\mathbf{S})_j \right) + \phi(\mathbf{a}^{(i)}) \right) + c \tag{9}$$

Where $\mathbf{a}^{(i)}$ is the ith row of $\mathbf{A}$, $t_{\mathbf{A}}$ controls the strength of the barrier, and c equals the terms that are constant with respect to $\mathbf{A}$.

For $\mathbf{S}$ we have:

$$f(\mathbf{S}) = \sum_i \left(t_{\mathbf{S}} \sum_j \left((\mathbf{A}\mathbf{s}_{(i)})_j - \mathbf{Y}_{ij} \log(\mathbf{A}\mathbf{s}_{(i)})_j \right) + \phi(\mathbf{s}_{(i)}) \right) + c \tag{10}$$

Where $\mathbf{s}_{(i)}$ is the ith column of $\mathbf{S}$, $t_{\mathbf{S}}$ controls the strength of the barrier, and c equals the terms that are constant with respect to $\mathbf{S}$.

Because the two cost function are separable, we can treat the rows of $\mathbf{A}$ and the columns of $\mathbf{S}$ as independent vectors to be optimized separately. The gradient and Hessian with respect to $\mathbf{a}^{(i)}$ are calculated as follows:

$$\mathbf{g}_{\mathbf{a}^{(i)}} = \bigtriangledown f(\mathbf{a}^{(i)}) = t_{\mathbf{A}} \left(\mathbf{1} - \frac{\mathbf{y}^{(i)}}{\mathbf{a}^{(i)}\mathbf{S}} \right) \mathbf{S}^{\mathrm{T}} - \mathbf{1}/\mathbf{a}^{(i)} \tag{11}$$

$$\mathbf{H}_{\mathbf{a}^{(i)}} = \bigtriangledown^2 f(\mathbf{a}^{(i)}) = t_{\mathbf{A}} \mathbf{S}\mathbf{F}_i\mathbf{S}^{\mathrm{T}} + \mathrm{diag}\left(\mathbf{1}/\mathbf{a}^{(i)}/\mathbf{a}^{(i)} \right) \tag{12}$$

$$\text{where} \quad \mathbf{F}_i = \mathrm{diag}\left(\mathbf{y}^{(i)}/(\mathbf{a}^{(i)}\mathbf{S})/(\mathbf{a}^{(i)}\mathbf{S}) \right) \tag{13}$$

Where division is carried out element-wise. The gradient and Hessian with respect to $\mathbf{s}_{(i)}$ are:

$$\mathbf{g}_{\mathbf{s}_{(i)}} = \bigtriangledown f(\mathbf{s}_{(i)}) = t_{\mathbf{S}}\mathbf{A}^{\mathrm{T}}\left(\mathbf{1} - \frac{\mathbf{y}_{(i)}}{\mathbf{A}\mathbf{s}_{(i)}}\right) - \mathbf{1}/\mathbf{s}_{(i)} \tag{14}$$

$$\mathbf{H}_{\mathbf{s}_{(i)}} = \bigtriangledown^2 f(\mathbf{s}_{(i)}) = t_{\mathbf{S}}\mathbf{A}^{\mathrm{T}}\mathbf{G}_i\mathbf{A} + \mathrm{diag}\left(\mathbf{1}/\mathbf{s}_{(i)}/\mathbf{s}_{(i)}\right) \tag{15}$$

$$\text{where} \quad \mathbf{G}_i = \mathrm{diag}\left(\mathbf{y}_{(i)}/(\mathbf{A}\mathbf{s}_{(i)})/(\mathbf{A}\mathbf{s}_{(i)})\right) \tag{16}$$

The computation of the above differentials is fairly straightforward; however, we can greatly improve the efficiency with which we calculate the Hessian by computing $\mathbf{A}^{\mathrm{T}}\mathbf{G}_i\mathbf{A}$ and $\mathbf{S}\mathbf{F}_i\mathbf{S}^{\mathrm{T}}$ a dyadic sum, i.e.

$$\mathbf{A}^{\mathrm{T}}\mathbf{G}\mathbf{A} = \sum_i G_{ii}{\mathbf{a}^{(i)}}^{\mathrm{T}}\mathbf{a}^{(i)} \tag{17}$$

Where $\mathbf{G}$ is diagonal and we're taking a linear combination of the outer product of each row of $\mathbf{A}$. Since all of the outer products can be stored and reused for Hessian calculation with respect to subsequent rows of $\mathbf{A}$, this method gives a 20x speedup in the Matlab implementation compared to direct matrix multiplication. We can use a similar dyadic sum to compute the Hessian with respect to columns of $\mathbf{S}$.

After we compute the gradient, $\mathbf{g}$, and Hessian, $\mathbf{H}$, the Newton step direction can be computed by solving the system $\mathbf{H}\mathbf{x} = \mathbf{g}$. The amount of computation required in this step is insignificant compared to the computation of the Hessian.

2.3 Backtracking line search

After we compute the Newton step direction, we must determine a suitable Newton step magnitude. To do this, we use a simple backtracking line search, by starting with an initial magnitude of $t_{bt} = 1$ and shrinking the magnitude until we get:

$$f(x + t_{bt} \bigtriangleup x) \leq f(x) + \alpha t_{bt} \bigtriangledown f(x)^{\mathrm{T}} \bigtriangleup x \tag{18}$$

For $\alpha \in (0, 0.5)$. If the above condition does not hold, we update t_{bt} by multiplying it by $\beta \in (0, 1)$.

After we find an appropriate t_{bt}, we update x:

$$x \leftarrow x + t_{bt} \bigtriangleup x \tag{19}$$

2.4 Softening the barrier

We use the above procedures to compute updates to $\mathbf{A}$ and $\mathbf{S}$. For a given $t_{\mathbf{A}}$, we update $\mathbf{A}$ using one Newton step. Then we update $\mathbf{S}$ once. In order to soften the barrier and allow values to approach zero, we must increase $t_{\mathbf{A}}$ and $t_{\mathbf{S}}$.

Each Newton step carries with it the value λ^2, which can be used to decide when the Newton iterations for a certain t have converged sufficiently.

$$\lambda^2 = \bigtriangledown f(x)^{\mathrm{T}} \bigtriangledown^2 f(x)^{-1} \bigtriangledown f(x) \tag{20}$$

When the λ^2 value for a Newton step is less than a specified threshold then we increase the corresponding barrier parameter $t_{\mathbf{A}}$ or $t_{\mathbf{S}}$ by multiplying it by a value $\mu > 1$.

2.5 Putting it all together

The entire algorithm can be summarized as follows:

1. Initialize $\mathbf{A}$ and $\mathbf{S}$ to random positive values and select an initial value for the barrier parameters $t_{\mathbf{A}}$ and $t_{\mathbf{S}}$
2. Complete a Newton step to update $\mathbf{S}$, then a Newton step to update $\mathbf{A}$.
3. If λ^2 for an iteration is less than a specified value then we multiply the corresponding barrier value by β and continue to step 4, otherwise we return to step 2.
4. If both $t_{\mathbf{A}}$ and $t_{\mathbf{S}}$ are greater than specified values we terminate the iterations and output the final values of $\mathbf{A}$ and $\mathbf{S}$.

3 Results

We evaluate our algorithms in two ways. First we use artificially generated ground truth matrices for $\mathbf{A}$ and $\mathbf{S}$ and use their product as the input matrix to be factored. We compare the quality of factorizations and time to convergence for the Lee and Seung multiplicative algorithm and for our own. We generate mixing matrices $\mathbf{A}$ of different condition numbers to test the robustness of the NMF procedures.

Second we use an actual spectrogram from a musical piece as the input matrix and test the ability of the algorithms to separate out the drum and percussion components from the spectrogram.

3.1 Artificial Data

The rows of an example ground-truth $\mathbf{S}$ matrix are plotted in Figure 1. This $\mathbf{S}$ matrix is multiplied by various 32x4 $\mathbf{A}$ matrices of different condition numbers to test the ability of the two algorithms to arrive at the ground-truth factors for many different random initializations of the matrices. An example of the resulting observation matrix $\mathbf{Y} = \mathbf{AS}$ is shown in Figure 2

In the tables below we have the results for 100 trials using different randomly generated initial matrices. Table 1 shows the results using an $\mathbf{A}$ matrix with a condition number of 3.75. This low condition number

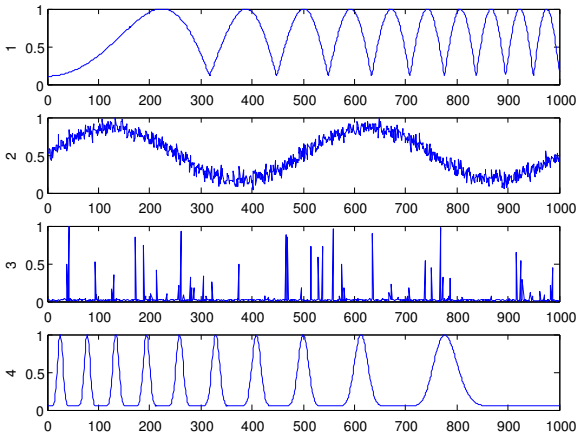

Figure 1: The rows of an example **S** matrix

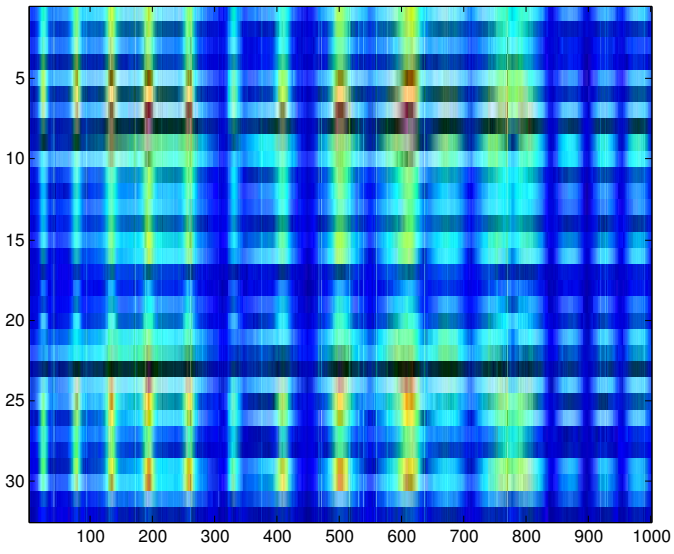

Figure 2: The example **S** matrix from Figure 1 mixed with a 32x4 **A** matrix with condition number 3.75

	Lee/Seung	Barrier Newton
Best(Avg) Cost	$7.5\text{x}10^{-5}$ (0.026)	$2.9\text{x}10^{-4}$ (0.004)
Best(Avg) Source Div	966 (3290)	658 (1370)
Avg Time [sec]	51.7	34.5

Table 1: Results using **A** matrix with condition number 3.75

means that the columns of the matrix **A** are not too similar, and therefore it should be easier to distinguish when individual sources (rows of **S**) are present.

For this particular matrix **A**, our Barrier Newton method achieves a lower value of the cost function on average than does the Lee and Seung algorithm; however, it doesn't achieve the best minimum of the trials. The smallest value of the cost function isn't our actual objective in NMF though, so we also compare the divergence between the resulting **S** matrices with that of the ground-truth matrix. In this case, the Barrier Newton method achieves a better overall and average source divergence. In addition, it converged around 50% faster on average. The best version from the 100 trials of the recovered source matrix is shown in Figure 3.

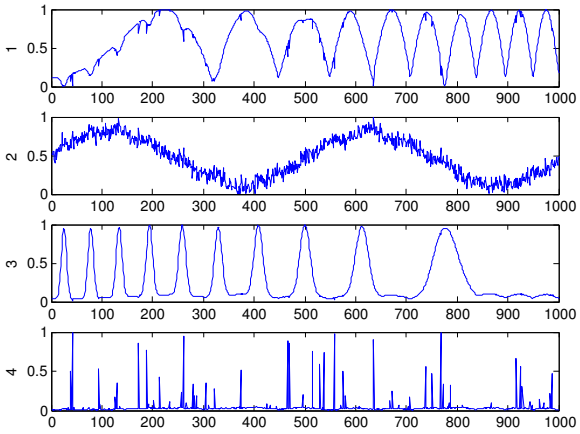

Figure 3: For the **A** matrix with condition number 3.75, this shows the rows of the resulting **S** matrix (using the barrier-Newton method) with best divergence with respect to the ground truth **S** matrix. The plot definitely resembles the original **S** matrix shown in Figure 1

Table 2 shows the results when the **A** matrix has a condition number of 50. This time the Barrier Newton method does worse with respect to source divergence; however, we can see that it achieves much lower values of the cost function in 1/10 the time. In this case, the high condition number doesn't allow for easy distinction between sources in the mixed matrix. Because the columns of **A** are so similar in this case, the Barrier Newton method does a much better job of reconstructing the original mixed matrix even though it does not recover the actual sources well. An example of the recovered sources is shown in Figure 4.

	Lee/Seung	Barrier Newton
Best(Avg) Cost	$1.5\text{x}10^{-5}$ (0.110)	$7.0\text{x}10^{-6}$ ($7.1\text{x}10^{-5}$)
Best(Avg) Source Div	$1.8\text{x}10^{4}$ ($2.74\text{x}10^{4}$)	$2.4\text{x}10^{4}$ ($3.68\text{x}10^{4}$)
Avg Time [sec]	75.8	7.7

Table 2: Results using **A** matrix with condition number 50

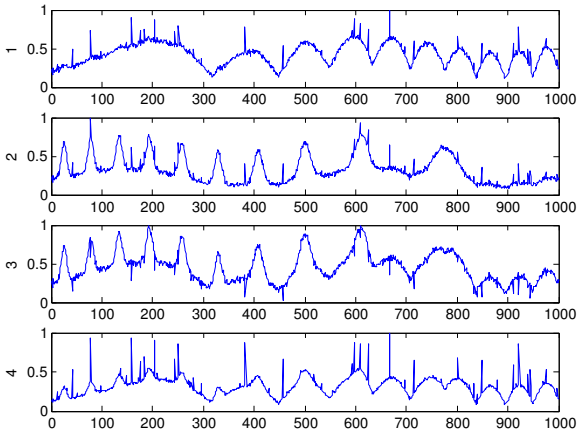

Figure 4: For the **A** matrix with condition number 50, this shows the rows of the resulting **S** matrix (using the barrier-Newton method) with best divergence with respect to the ground truth **S** matrix.

3.2 Music Data

Testing the ability of NMF to separate an audio signal into separate musical sources is a difficult task because it requires a bit of hand labeling and there is always the question of what is actually ground truth. In this example, we use a 10 sec clip of audio that includes a snare drum, a bass drum, and two guitar mixed together on a single channel of audio. We are able to apply NMF in this case despite the single channel observation by computing a 512 frequency bin spectrogram of the signal. These 512 frequency bins serve as the mixed observations, and the spectrogram is used as the mixed matrix $\mathbf{Y}$.

We will attempt to factorize the spectrogram into 10 sources. The idea is that hopefully the contribution from the snare drum and bass drum will use as few sources as possible (ideally one each), and the various guitar notes played will use the remaining sources. After NMF is complete, we use our previously developed feature extraction and SVM classification system to distinguish between sources that could be drums and those that are not [5]. By doing this we can reconstruct a spectrogram containing only the drum instruments by zeroing the contribution from the sources classified as non-drum. The resulting drum spectrogram can be used to resynthesize an audio waveform containing an approximation of the drum track from the song.

To evaluate the results, we must have some sort of ground-truth drum track data. Luckily there are archives of audio freely available online that contains unmixed audio tracks to be used for just this purpose. In this experiment, we use 10 seconds of the track *TV On* by Kismet from the MTG Mass Resources archive [6].

The results are shown in Table 3. The Barrier Newton method is able to achieve a slightly lower value of the cost function on average. The second row of the table shows the divergence between the resulting NMF drum track spectrograms and the ground-truth spectrogram computed when only the drum instruments are mixed into the audio signal. We see that the Lee/Seung algorithm achieves a better minimum, though Barrier Newton performs significantly better on average. The signal-to-noise ratio also shows this trend, where Barrier Newton performs better on average. The startling result here though is the much longer average time to convergence of the Barrier Newton method. Even after tweaking the convergence and update parameters, it still takes about four times as long to sufficiently converge on average.

	Lee/Seung	Barrier Newton
Best(Avg) Cost	$9.16\text{x}10^6$ ($9.30\text{x}10^6$)	$9.16\text{x}10^6$ ($9.25\text{x}10^6$)
Best(Avg) Drum Track Div	$2.56\text{x}10^7$ (4.9810^7)	$2.81\text{x}10^7$ ($3.63\text{x}10^7$)
Best(Avg) Drum SNR	12.16 (11.53)	12.08 (11.67)
Avg Time [sec]	41.4	162.5

Table 3: Results for drum track separation on 10 sec of sample audio

3.3 Conclusion

Alternative techniques for Non-negative matrix factorization are definitely desirable. The Lee/Seung multiplicative algorithm is very easy to implement and works fairly quickly, which is why it is so widely used. As seen in this project, using a higher order and more complex iterative algorithm for NMF can work very well for certain problems. In the target application of music source separation, the initial results of the proposed Barrier Newton method are not that promising. If the algorithm isn't able to converge in a small number of steps, the cost of computing the Hessian in each step makes this approach painfully slow.

Future work along these lines may include developing an intelligent update method for the Barrier parameters which takes into account the complementary slackness information contained in the dual of the optimization problems. This could allow for much larger increases in the Barrier parameters and therefore much quicker convergence.

References

[1] T. Virtanen, "Monaural sound source separation by nonnegative matrix factorization with temporal continuity and sparseness criteria," *IEEE Transactions on Audio, Speech, and Language Processing*, vol. 15, no. 3, pp. 1066–1074, 2007.

[2] D. Guillamet and J. Vitria, "Classifying faces with nonnegative matrix factorization," in *Proc. 5th Catalan Conference for Artificial Intelligence*.

[3] W. Xu, X. Liu, and Y. Gong, "Document clustering based on non-negative matrix factorization," in *Proceedings of the 26th annual international ACM SIGIR conference on Research and development in informaion retrieval*. ACM Press New York, NY, USA, 2003, pp. 267–273.

[4] D. Lee and H. Seung, "Algorithms for Non-negative Matrix Factorization," *Advances In Neural Information Processing Systems*, pp. 556–562, 2001.

[5] E. Battenberg, "Improvements to Percussive Component Extraction Using Non-Negative Matrix Factorization and Support Vector Machines," http://www.eecs.berkeley.edu/ ericb/school/masters.pdf, 2008.

[6] M. Vinyes, "MTG MASS Resources," http://www.mtg.upf.edu/static/mass/resources, 2008.