Response to Review 2

Benjamin Sapper[1], Daven Henze[1], and Jose Jimenez[2]

[1]University of Colorado Boulder, 11 Engineering Dr, Boulder, CO 80309, United States
[2]Cristol Chemistry and Biochemistry, Boulder, CO 80309, United States

Correspondence: Benjamin Sapper (bsapper77@gmail.com)

We thank Anonymous Reviewer 2 for their response. We have considered both reviewers' comments and believe that we have greatly improved the manuscript from them. We also have made additional changes to the manuscript separate from the reviewers' suggestions, mainly making our figures (1, 3, 5-9, 12, 14, 15, A2-A10) clearer, clarifying parts of the text, and correcting spelling and grammar errors. Notably, the external vs. internal weighting comparison in Figures 7, 8, A2, A3, A4, and A5 were redone with a different random seed. However, the interpretation of these results did not change.

In our response, red text is the original comment from the reviewer, black text is our response, and blue text is our updates to the manuscript.

1 Main Criticism 1

The key point of the algorithm is that the combination of RHALS with the external weighting scheme is faster than competing schemes. However, to solidly back this claim, it is not sufficient to only provide runtimes, especially from unoptimized MATLAB implementations. For example, the computational costs of several formulas, e.g the updates on p9, are very sensitive to the order of the operations, the parenthesization, etc. MATLAB is very convenient, but does not attempt to optimize matrix expressions. Were these issues taken into account? The authors also need to show that the "competing" codes have been implemented with runtime minimization in mind. Unfortunately, it is not sufficient to compare performance-oblivious implementations when the objective is to reduce runtimes.

Both the MU and ALS algorithms listed in the paper have been formulated so that computing the relative matrix expressions is optimized. We chose not to explore implementing outside software for running these algorithms so we could specify certain diagnostics and features within the algorithm, such as regularization and the stopping condition. The basic HALS algorithm is translated from Python code referenced in Erichson et al. (2018), which references the function *_update_cdnmf_fast* in the *scikit-learn* Package in Python (Pedregosa et al., 2011). We argue that this is a "competitive" implementation of HALS.

We note that certain initial diagnostic calculations within these algorithms may slightly increase run times, such as processing the weighting matrix and allocating memory towards tracking the stopping condition, but we keep them within the algorithms for testing, as these are uniform across all algorithms and their contribution to run time is negligible.

MATLAB is well known to be computationally efficient in performing of matrix algebra, and we believe that the implementation of these algorithms in MATLAB yields a fair comparison of computational costs.

Nevertheless, we agree that the manuscript would be improved by adding the number of operations needed for each algorithm. Thus, we have inserted Table 1 into the manuscript.

Table 1. Comparison of the number of operations per step for each algorithm used. We also track the number of elementwise multiplications and divisions (implemented in Matlab as ".*" and "./") in the third column.

Comparison of Cost/Step for Different Algorithms		
Algorithm	Number of Operations Using Matrix Multiplication	Number of Elementwise Multiplications/Divisions
ALS (IW)	$\mathcal{O}(mnk^2)$	$2mnk$
ALS (EW)	$\mathcal{O}(mnk)$	0
MU (IW)	$\mathcal{O}(mnk)$	$4mn + 4(m+n)k$
MU (EW)	$\mathcal{O}(mnk)$	$4(m+n)k$
MU (Randomized)	$\mathcal{O}((k+l)nk)$	$4((k+l)+n)k$
HALS (IW)	$\mathcal{O}(mnk^2)$	$2mnk$
HALS (EW)	$\mathcal{O}(mnk)$	0
RHALS	$\mathcal{O}((k+l)nk)$	0
Post Processing	$\mathcal{O}(mnk)$	0

We have also added the following text to the manuscript in Section 4.1 (lines 404-413)

Table 1 shows the number of operations required at each step (update of $\mathbf{W}$ and $\mathbf{H}$). Preprocessing steps such as finding $\mathbf{A} \oslash \mathbf{\Sigma}$ and $\mathbf{\Sigma} \odot \mathbf{\Sigma}$ are excluded from the table. External Weighting eliminates almost all elementwise operations needed, which is a slow memory bound operation (Jia et al., 2020). However, systems with an abundance of free memory and/or GPUs may find that internal weighting methods are sufficiently quick for small or medium size problems. Furthermore, the performance of matrix operations within these algorithms may vary between programming languages and libraries, such as between the numpy, numexpr, and Theano libraries in Python (Bergstra et al., 2010).

Analyzing Table 1 allows us to see why, in practice, it takes longer for internally weighted ALS and HALS to converge than internally weighted MU, as the former will have more matrix multiplication operations and elementwise operations for our target rank, $k = 6$. However, when the algorithms are externally weighted, ALS and HALS become much more computationally efficient, allowing runtime to diminish as well.

2 Main Criticism 2

The authors should also specify the costs of each method per iteration (preferably in terms of the matrix primitives used).

See above.

3 Main Criticism 3

Comparisons are with weighted versions of ALS and MU. These are prone to slow convergence and or even failure to reach a reasonable local optimal solution. ALS, for example (see N. Gillis, p283 of https://doi.org/10.1137/1.9781611976410) "is easy to implement ... and sometimes provides reasonable solutions, typically for sparse matrices; ... However, it comes with no theoretical guarantee and in fact often diverges in practice, especially for dense input matrices; ... Therefore, ALS can be recommended only as a warm-start stage for theoretically better grounded approaches...." See chapter 8 of the aforementioned book and Table 8.2 for a comparison of methods: ALS and MU stand at the "low end" in several respects.

We recognize that ALS has been deemed a suboptimal choice for PMF problems. However, with the data used in the paper, we found that ALS almost always converged to reasonable solutions, albeit at slower run times. The additional algorithms Table 8.2 in Gillis (2020) lists include Projected Gradient Descent (PGM), Alternating Nonnegative Least Squares (ANLS), and Alternating Direction Method of Multipliers (ADMM), as well as techniques to speed up convergence of any NMF algorithm, such as updating $\mathbf{H}$ several times before updating $\mathbf{W}$. Since the main purpose of testing several algorithms is to show the differences between using internal and external weighting, as well as randomization, we choose not to add the implementation of these alternative algorithms. However, we add the following sentences to the Background in the manuscript:

In Section 1.2.2 (lines 67-69)

It is possible to perform NMF using other forms of gradient descent - for example, the projected gradient method (PGM) sets the step size to the inverse of the maximum eigenvalue of Hessian of the cost function, and may lead to faster convergence than MU (Gillis, 2020). However, we choose to only test MU, due to its widespread use and flexibility (Gillis, 2020).

In Section 1.2.3 (lines 75-82)

Nonnegative ALS has no theoretical convergence guarentee and, in some problems, may fail to converge to a feasible solution (Gillis, 2020). For this reason, Alternating Nonnegative Least Squares (ANLS) and Alternating Direction Method of Multipliers (ADMM) are interesting alternatives. In ANLS, indices of an "active set" are set to zero, and the rest are updated via an unconstrained optimization (Kim and Park, 2011). The active set is then updated to contain the indices with the new negative factor elements. In ADMM, an auxiliary factor matrix $\mathbf{Y}$ is formed, and an additional term is added to the cost function which penalizes the distance between the target factor matrix $\mathbf{W}$ or $\mathbf{H}$ and $\mathbf{Y}$ (Gillis, 2020). Both of these methods may lead to faster and better convergence than nonnegative ALS, and are preferred by Gillis (2020). However, we find that the simple

nonnegative ALS almost always converges to feasible solutions for our dataset, and we do not explore these alternative methods.

We have also considered these comments for the convergence of the external weighting step – see our response to Main Criticism 4.

4 Main Criticism 4

In Section 4.1, I am missing an analysis of the contribution of the number of factors to the cost. Indeed, in some cases the discussion is incomplete: For example on p11 it is stated that "... the algorithm may take longer for a larger number of factors and in certain cases. More work is needed regarding the different convergence issues that may arise." In light of the fact that the external weighting scheme is a key contribution of this paper, the authors need to address these issues in greater detail.

See Point 1 for the discussion of the number of factors and computational costs.

We omit numerical results of the number of factors vs runtime, as we have already listed the computational complexity in our response to point 1.

More recent research has found that adding L2 regularization to the post processing step of the External Weighted algorithms improves convergence. Thus, we have deleted the following in Section 2.3 (pg 11):

However, the algorithm may take longer for a larger number of factors and in certain cases. More work is needed regarding the different convergence issues that may arise.

and have added (lines 320-326):

In practice, one can use L2 regularization in the external weighting steps, equal to 0.01 for our data, as the least squares method may become increasingly ill-posed as the number of factors increases. This value may need to be altered based on the magnitude of the values in the data as well as the number of factors. However, we found that adding L2 regularization lowered the similarity of the factors to the given factors from the solution using PMF2.

We used this method for all externally weighted algorithms tested in Section 4. However, theoretically, any nonnegative matrix factorization algorithm could be used for the post processing step. This may become relevant, since as noted in Section 2.2.3, the nonnegative ALS method described above may have convergence issues for certain factorization problems (Gillis, 2020).

5 Main Criticism 5

It would also be useful to provide information on the cost of each major step and iteration rather than only comparing runtimes to convergence.

See Point 1 for the discussion about computational complexity per step. As we already have a discussion of the percentage of each step of the RHALS algorithm (randomization, main algorithm, and post processing step) in Section 4.1 of the manuscript, we do not add specific numerical results on these values.

6 Main Criticism 6

The paper makes a brief reference to a WNMF method for missing entries (Yahaya et al.) However the method is dismissed because of the cost of EM without further ado. The authors should address a) how they would handle missing entries, b) how does their method compare in view of related papers by the same team, some included in the publically available thesis of F. Yamaha. It is worth noting that the paper "Random Projection Streams for (Weighted) Nonnegative Matrix Factorization," doi: 10.1109/ICASSP39728.2021.9413496." discusses computational complexity, something I am missing in this manuscript as explained earlier.

a) External Weighting can only apply to continuous weights, as the pre and post processing steps can only take nonzero real entries. We have clarified this by adding the following in Section 2.3 (lines 293-296) in the manuscript:

We note that the uncertainty matrix $\mathbf{\Sigma}$ must only contain nonzero real entries. Thus, it can not handle problems with binary weights, such as PMF problems with missing entries. One approach to PMF with binary weights is the Expectation-Maximization (EM) approach detailed in Section 1.5.1 (Yahaya et al., 2021; Zhang et al., 2006).

b) See Response to Review 1 for our discussion on Dr. Yamaha's results on the Expectation-Maximization approach to weighted PMF , which includes the addition of Section 1.5.1 as mentioned above, as well as Section 4.3.2. We have also deleted the following from Section 2.3:

Furthermore, how would the uncertainties be included after the dimension of the data is reduced by a randomization step? One way to address the weighting problem introduced by dimension reduction is through an expectation maximization step layed out in (Yahaya et al., 2019). At each iteration, the data matrix $\mathbf{A}$ *is scaled based on the uncertainties and current factors, and then compressed again into a lower dimension. However, this method is still computationally expensive – in fact, the computationally expensive step of expectation maximization eliminates any of the time benefits gained by randomization.*

7 Main Criticism 7

I spotted some (trivial to correct but somewhat sloppy-type) errors. Some examples: Formula (7) and the formula for the quality of fit parameter (line 161) are incorrect. In the former, one must use the absolute values of the terms of the sum and in the latter it is the individual terms that must be squared, not the sum. Line 324 on p13 also has such an error. On p. 19, line 527, it is stated that "H is a k × n matrix that is assumed to be of full column rank". However, assuming that k < n, it cannot be full column rank, but at best, full row rank. There might be other errors as well.

These have been corrected: Formula (7) is now

$$\mathcal{Q} = \sum_{i=1}^{m}\sum_{j=1}^{n}\left(\frac{\mathbf{A}_{ij} - \sum_{d=1}^{k}\mathbf{W}_{id}\mathbf{H}_{dj}}{\sigma_{ij}}\right)^2$$

and the equation for the quality of fit parameter is now

$$\mathcal{Q} = \sum_{i}\sum_{j}\left(\frac{\mathbf{A}_{ij} - \sum_{k}\mathbf{W}_{ik}\mathbf{H}_{kj}}{\sigma_{ij}}\right)^2$$

On p. 13, the weighted squared error formula has been corrected to

$$\sqrt{\sum_{i}\sum_{j}\left(\frac{\mathbf{A}_{ij} - \sum_{k}\mathbf{W}_{ik}\mathbf{H}_{kj}}{\sigma_{ij}}\right)^2}$$

Also, in the Appendix, the phrase

H *is a k ×× n matrix that is assumed to be of full column rank*

has been replaced by

H is a k × n matrix that is assumed to be of full row rank.

8 Other Comments

[page 1]: Please state whether the matrix is assumed to be strictly positive or nonnegative. If the latter, then why use PMF instead of NMF which is more general and the term used in some of the paper's major citations.

We see the terms "Positive Matrix Factorization" (PMF) and "Nonnegative Matrix Factorization" (NMF) to be interchangable. For example, Paatero et. al uses the terms "positive" and "nonnegative" in the title of Paatero and Tapper (1994). Eq. 1 on line 35 of the manuscript already specifies that the factor matrices **W** and **H** can take the value of 0, and thus we see no need to clarify this further.

[page 2 and beyond]: The authors state: ”In Eq. (1), the division by Σ represents elementwise division”. I strongly recommend against using fractional notation when dealing with matrices. They should use Hadamard division or Hadamard multipication with a variable that is defined to contain the inverses of the elements of Σ. In fact, on p265, the authors actually use the (MATLAB infix operator) for elementwise division. This is certainly better than the / symbol, but even better to use the ”Hadamard notation”.

We have replaced the notation of $/$ for elementwise division with $\oslash$.

[page 3]: Better include the section number when you refer to Sections, e.g. Methods (Section 2)

We have rewrote the phrase *the Methods Section* on line 89 as Section 2.

[page 4]: The use of ”*” to denote matrix multiplication is inconsistent with the rest of the paper.

We have replaced “$*$" on line 112 with “$\cdot$".

[page 4]: A proper and complete discussion on computational optimizations and parallelization would merit much more than the few statements in sections 1.3. I propose that the authors take this into account and eliminate section 1.3.2, incorporating the few comments in some other section.

We have gotten rid of section 1.3.2. Section 1.3 is now just titled "Random Projections." We have also clarified the use of GPUs by changing *can be parallelized* to can be parallelized using Graphics Processing Units (GPUs) on line 425.

[page 6]: Please clarify the statement of line 140. How does the incorporation of the weighting via the uncertainty matrix affect the costs?

As mentioned earlier in the response to Main Criticism 1, elementwise operations are slow memory bound processes for large matrices. We have updated the manuscript on lines 145-148 as follows:

However, doing this is very computationally expensive due elementwise operations with the uncertainty matrix $\mathbf{\Sigma}$. Elementwise operations of large arrays are inefficient processes compared to other operations of the same computational complexity, such as matrix-vector multiplication, due to the large allocation of memory towards intermediary results (Jia et al., 2020).

[page 8]: Formula (10) is missing the opeining (left) parenthesis. Also in line 210 better say ”denotes the trace of the matrix” rather than ”takes the trace of the matrix”. After formula (14) better say ”Differentiating with respect...” instead of ”Taking the

derivative ...".

We have fixed these minor inaccuracies.

[page 9]: Careful in the parenthesization of Formula (16). Some parentheses are missing

The only issue with Eq. 16 is that we mistakenly returned to elementwise notation while the uncertainty matrices $\mathbf{\Sigma}_i$ and $\mathbf{\Sigma}_p$ were still diagonal. We replace $\mathbf{\Sigma}_i$ and $\mathbf{\Sigma}_p$ with $\mathbf{\Sigma}(i,:)$ and $\mathbf{\Sigma}(:,p)$, the i^{th} row and p^{th} column of $\mathbf{\Sigma}$, respectively, in Eqs. 19-24 (previously 16-21).

[page 9]: In formula (20) and elsewhere, aren't the $\mathbf{\Sigma}_j$ matrices diagonal? If so, then the transpose symbol is redundant. Also careful in the parenthesization of Formula (20). Some parethesizations can be more efficient than others.

See above for the issue with $\mathbf{\Sigma}_i$ and $\mathbf{\Sigma}_p$. In Eq. 20, the algorithm is made more computationally efficient by finding $\mathbf{H}(\mathbf{H}^T(j,:) \oslash (\mathbf{\Sigma}^T(i,:) \odot \mathbf{\Sigma}^T(i,:)))$ before multiplying on the left by $\mathbf{W}$ (and similarly in Eq. 21). We also note that the product $\mathbf{WH}$ ($\mathbf{H}^T\mathbf{W}^T$ in Eq. 21) is not preallocated before the update, the same implementation as Erichson et al. (2018). We have updated the manuscript on lines 258-260 as follows.

In Eqs. 20 and 21, the Hessians $\mathbf{H}(\mathbf{H}^T(j,:) \oslash (\mathbf{\Sigma}^T(i,:) \odot \mathbf{\Sigma}^T(i,:)))$ and $\mathbf{W}^T(\mathbf{W}(:,j) \oslash (\mathbf{\Sigma}(:,p) \odot \mathbf{\Sigma}(:,p)))$ should be found prior to multiplication by $\mathbf{W}(i,:)$ and $\mathbf{H}^T(:,p)$, respectively, to minimize computational costs. We do not preallocate the products $\mathbf{WH}$ and $\mathbf{H}^T\mathbf{W}^T$, which is the same implementation as in Erichson et al. (2018).

[page 9]: In formula (20) and elsewhere: have you defined the maximum operator $[\cdot]_+$?

We have updated the manuscript on line 248 as:

where $[\boldsymbol{v}]_+$ projects all negative values of $\boldsymbol{v}$ to 0.

[page 10]: Some readers might question the characterization "rotation" for a non-orthogonal matrix. In the linear algebra literature, rotation matrices preserve the 2-norm and the Frobenius norm and are orthogonal. It might be worth clarifying this.

We agree with the reviewer that it is best to clarify this. We add to Section 1.4 (lines 118-119):

We note that $\mathbf{T}$ does not necessarily represent a true rotation in a mathematical form, which would require $\mathbf{T}$ to be orthogonal.

and to Section 2.2 (lines 262-263)

As mentioned in Section 1.4, we do not attempt to constrain these “rotations" to be norm preserving. However, it is possible to find approximate rotations.

We also put quotation marks around the first word “rotational" in Section 1.4 (line 117). Additionally, we cite Paatero et al. (2002) on line 123.

[page 11]: In line 273, better say ”Here $W^\dagger$ and $H^\dagger$ denote the pseudoinverses ...” It is then stated that ”Running this code in Matlab on a single CPU, we found the update rules using the pseudoinverses were less computationally expensive.” Less expensive than what? How are the pseudoinverses computed? What is the computational cost of this part of the algorithm relative to the rest? How many iterations are necessary? Do they lead to convergence? In particular, looking at the code it seems that the method calls MATLAB’s pinv. Is this necessary? What if the matrix is m×n with n $<<$ m? Also reading function randomized nnmf.m one sees the comment: ”NOTE: This code is not adapted from any research papers or pre existing code. The only motivation is that this code seems to produce feasible results and we haven’t thought of any better alternatives. More testing needs to be done on the convergence of this method to feasible solutions.” Has this been done?

We have replaced

Here, $\mathbf{W}^\dagger$ denotes the pseudoinverses of $\mathbf{W}$

with (line 304)

Here, $\mathbf{W}^\dagger$ and $\mathbf{H}^\dagger$ denotes the pseudoinverses of $\mathbf{W}$ and $\mathbf{H}$

We have addressed the Computational Cost in 1 in response to Main Criticism 1. Both proposed post processing steps are $\mathcal{O}(mnk)$, and we found experimentally that using MATLAB’s pinv, a Krylov subspace method (Feng et al., 2018), generated slightly faster results. As mentioned on page 11 on line 287, the algorithm takes 20-40 iterations to converge. As mentioned in response to Main Criticism 4, regularization can be added to improve the convergence, especially if a large number of factors are chosen. As noted in the Appendix of the manuscript, using the pseudoinverse is equivalent to using least squares, as long as $\mathbf{W}$ and $\mathbf{H}$ are full rank. If n $\ll$ m, the update of $\mathbf{W}$ using $\mathbf{H}^\dagger$ is a less overdetermined problem compared to the update of $\mathbf{H}$. However, since $\mathbf{W}$ and $\mathbf{H}$ are alternatingly updated, it is not clear as to whether the algorithm would be affected by changing the ratio of m to n. That being said, further improvements in the external weighting algorithm are a subject for future research.

We have updated the manuscript on lines 309-310:

In Matlab, a Krylov subspace algorithm is used for calculating the pseudoinverses, and both update rules are $\mathcal{O}(mnk)$ (Feng et al., 2018).

Additionally, we replaced

Running this code in Matlab on a single CPU, we found the update rules using the pseudoinverses were less computationally expensive.

with (lines 310-311)

Running this code in Matlab on a single CPU, we found the update rules using the pseudoinverses were faster.

We also removed the above comment in the code for randomized_nnmf.m. The new doi for the algorithm code is https://zenodo.org/badge/latestdoi/537111580.

[page 12]: In Figure 3, better say that MATLAB notation is used, in which the digits following the e symbol are the exponent to which

Since many readers may not be familiar with MATLAB notation, we keep the description of “e" as is. However, we add to line 344

and the number to the right is the exponent (i.e. e03=10^3).

[page 13]: there is no nonnegative SVD, the authors mean something else (possibly the so called ”nonnegative double SVD” initialization from ref. [1]).

We have corrected lines 372-373 to read the nonnegative double SVD (NNDSVD).

[page 19:] In line 525, $\mathbf{W}^\dagger$ must be bold according to the paper’s notation. In any case, I suggest not to include Appendix B; its findings are well known linear algebra facts.

This has been corrected. We keep Appendix B, as some readers may have a background primarily in atmospheric chemistry, and may be interested as to why the external weighting algorithms are identical mathematically.

[page 21, Bibliography]: Line 543: (Burred) Incomplete reference: no date, publication information, etc.

We have added the following peer reviewed reference to line 61 (where Burred is orginally cited): (Gillis, 2020). We have also added a more complete reference for the original source – see Burred (2017).

[Bibliography] The authors should also take into account the WNMF work described in the well cited 2008 PhD KU Louvain thesis by N.D. Ho . Another important reference (as noted earlier) is the book by N. Gillis.

We have added in the citation of N.D. Ho's thesis as an alternative derivation of weighted HALS into Section 2.1 on line 216.

Another derivation of weighted HALS is given in Ho (2014).

[Bibliography] The entry "Tan, W., C. S. F. L. L. C. W. Z. and Cao, L...." is probably wrong. See 10.1145/3225058.3225096.

See Tan et al. (2018).

[page 27]: Fig4: Do you mean log n or $log(n^2)$. Which base log? What are the exact sizes of the smallest and largest data sets?

The logarithm used is the base 10 log, and it is taken of n. The caption to Figure 4 has been updated as

Size of data matrix vs run time (in seconds) in RHALS algorithm, performed with three different random seeds (containing the absolute value of random normal variables). The x-axis shows the base 10 log of n, and the y-axis plots the base 10 log of run time. The median run time is plotted, with the error bars plotting the maximum and minimum run times. Note that run time initially decreases due to a coincidental reduction in the number of steps to convergence, but problems of environmental interest are generally located on the right of the graph or beyond its right edge.

References

Bergstra, J., Breuleux, O., Bastien, F., Lamblin, P., Pascanu, R., Desjardins, G., Turian, J., Warde-Farley, D., and Bengio, Y.: Theano: A CPU and GPU Math Compiler in Python, pp. 18–24, Austin, Texas, https://doi.org/10.25080/Majora-92bf1922-003, 2010.

Burred, J.: Detailed derivation of multiplicative update rules for NMF, https://www.semanticscholar.org/paper/Detailed-derivation-of-multiplicative-update-rules-Burred/3376b4df752f2428c451e530f9c6e0ce3a3f05e4, 2017.

Erichson, N. B., Mendible, A., Wihlborn, S., and Kutz, J. N.: Randomized nonnegative matrix factorization, Pattern Recognition Letters, 104, 1–7, https://doi.org/10.1016/j.patrec.2018.01.007, 2018.

Feng, X., Yu, W., and Li, Y.: Faster Matrix Completion Using Randomized SVD, in: 2018 IEEE 30th International Conference on Tools with Artificial Intelligence (ICTAI), pp. 608–615, https://doi.org/10.1109/ICTAI.2018.00098, iSSN: 2375-0197, 2018.

Gillis, N.: Chapter 8: Iterative algorithms for NMF, in: Nonnegative Matrix Factorization, Data Science, pp. 261–305, Society for Industrial and Applied Mathematics, https://doi.org/10.1137/1.9781611976410.ch8, 2020.

Ho, N.-D.: Non-negative Matrix Factorization Algorithms and Applications, Ph.D. thesis, UNIVERSITÉ CATHOLIQUE DE LOUVAIN, Belgium, https://doi.org/10.1002/9781118679852.ch15, 2014.

Jia, L., Liang, Y., Li, X., Lu, L., and Yan, S.: Enabling Efficient Fast Convolution Algorithms on GPUs via MegaKernels, IEEE Transactions on Computers, 69, 986–997, https://doi.org/10.1109/TC.2020.2973144, conference Name: IEEE Transactions on Computers, 2020.

Kim, J. and Park, H.: Fast Nonnegative Matrix Factorization: An Active-Set-Like Method and Comparisons, SIAM Journal on Scientific Computing, 33, 3261–3281, https://doi.org/10.1137/110821172, publisher: Society for Industrial and Applied Mathematics, 2011.

Paatero, P. and Tapper, U.: Positive matrix factorization: A non-negative factor model with optimal utilization of error estimates of data values, Environmetrics, 5, 111–126, https://doi.org/10.1002/env.3170050203, _eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/env.3170050203, 1994.

Paatero, P., Hopke, P. K., Song, X.-H., and Ramadan, Z.: Understanding and controlling rotations in factor analytic models, Chemometrics and Intelligent Laboratory Systems, 60, 253–264, https://doi.org/10.1016/S0169-7439(01)00200-3, 2002.

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., and Cournapeau, D.: Scikit-learn: Machine Learning in Python, MACHINE LEARNING IN PYTHON, p. 6, 2011.

Tan, W., Chang, S., Fong, L., Li, C., Wang, Z., and Cao, L.: Matrix Factorization on GPUs with Memory Optimization and Approximate Computing, in: Proceedings of the 47th International Conference on Parallel Processing, ICPP '18, pp. 1–10, Association for Computing Machinery, New York, NY, USA, https://doi.org/10.1145/3225058.3225096, 2018.

Yahaya, F., Puigt, M., Delmaire, G., and Roussel, G.: How to Apply Random Projections to Nonnegative Matrix Factorization with Missing Entries?, in: 2019 27th European Signal Processing Conference (EUSIPCO), pp. 1–5, https://doi.org/10.23919/EUSIPCO.2019.8903036, iSSN: 2076-1465, 2019.

Yahaya, F., Puigt, M., Delmaire, G., and Roussel, G.: Random Projection Streams for (Weighted) Nonnegative Matrix Factorization, in: ICASSP 2021 - 2021 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), pp. 3280–3284, https://doi.org/10.1109/ICASSP39728.2021.9413496, iSSN: 2379-190X, 2021.

Zhang, S., Wang, W., Ford, J., and Makedon, F.: Learning from Incomplete Ratings Using Non-negative Matrix Factorization, in: Proceedings of the 2006 SIAM International Conference on Data Mining (SDM), Proceedings, pp. 549–553, Society for Industrial and Applied Mathematics, https://doi.org/10.1137/1.9781611972764.58, 2006.