Matrix Analysis Spring 2022 - Course project
Reading report of
Accelerated Multiplicative Updates and Hierarchical ALS Algorithms for Nonnegative Matrix Factorization

An Vuong

Abstract

This paper mainly focuses on the discussion provided by (Gillis & Glineur, 2012), wherein the authors gave analysis on computational cost of factor updates in two-block coordinate descent scheme for Nonnegative Matrix Factorization (NMF) and proposed a novel method for setting stopping criterion. This work summarizes my understanding of the aforementioned procedure, it also provides application to a new facial recognition dataset containing more than ten thousands images, an analysis of the dataset and a study of key facial characteristics preservation of NMF are provided. The codes can be found at https://github.com/anvuongb/matrix-analysis-final. Codes with weights, high resolution figures and samples data can be found at https://1drv.ms/u/s!ArynER9YzmcRgfFCkRWSRTbTaZB19g?e=Wabh7n

1. Introduction

An important component of language understanding, pattern recognition, and signal processing is how to find a fitting representation of high-dimensional data with multiple components mixing with each other. In order words, the decomposition of a signal, be it an image or a text document, into its principal components is of vital importance in multiple disciplines. This decomposition, often called factorization, can help us better understand the structure of the data by looking at it from the latent space perspective. Furthermore, in many cases, this latent representation is of much lower dimensions compared to that of the original data. Hence, by tackling the decomposition problem, a compressed view on the data can also be achieved, which in turn can lead to a tremendous gain in computation efficiency.

Through out the years, many methods have been proposed to solve this problem, among them, some of the more famous algorithms are Principle Component Analysis (PCA) (probably the oldest multivariate analysis technique (Abdi & Williams, 2010)), Latent Discriminant Analysis (LDA), and Independent Component Analysis (ICA). While these algorithms are all essentially different in their approaches to the factorization problem, the resulting decomposed signals all shared one same characteristics: they all have negative values in their components. This means, in order to get the whole, some of the parts are subtracted while others are added. While this is not a problem mathematically, it does not make sense in some real-world applications, such as Hyperspectral Unmixing (Bioucas-Dias et al., 2012) where the material distribution must be nonnegative or energy disaggretion (Kolter & Johnson, 2011) where energy consumption should be positive (a negative sign can appear but it means direction of energy transmission). To tackle this issue, a new method called Nonnegative Matrix (NMF) Factorization was proposed, in which the latent representations are strictly nonnegative.

While NMF had already been utilized for years in geoscience and chemistry, it was not until the publication of (Lee & Seung, 1999) that the method came to attention of computer science community. The Multiplicative update (MU) method provided by Lee & Seung was simple yet very effective. Since the publication of this paper, NMF has been successfully applied in multiple research areas such as: hyperspectral unmixing (Ma et al., 2013), electricity disaggregation (Kolter et al., 2010), topic modelling (Arora et al., 2016) and many more. Despite these successful attempts at applying NMF, the update rule provided by (Lee & Seung, 1999) was shown to be rather slow in convergence rate (Gillis & Glineur, 2008). For this reason, many other NMF algorithms have been proposed to improve the efficiency.

In (Li & Zhang, 2009), the authors proposed a monotonic, fixed-point algorithm called FastNMF that was shown to achieve a significantly better accuracy compared to (Lee & Seung, 1999) in the same duration, or achieve the same accuracy during a much shorter amount of time. Similarly, Chapter 4 in (Ho, 2008) proposed a new method, termed Rank-one residue iteration (RRI), it was shown in the thesis that RRI was up to five times faster than MU and, interestingly, MU even failed to run in many tests where RRI did not. Independently, a procedure similar to FastNMF and RRI was also mentioned in (Cichocki et al., 2007), hence these algorithms will be referred to as Hierarchical Alternating Least Squares (HALS) from this point.

In (Gillis & Glineur, 2012), the authors provided an analysis of computational cost of MU and HALS, they then presented novel stopping criterion in an attempt to speed up the convergence of MU and HALS. Using the criterion, an application to (Lin, 2007b) and a discussion on Stationary point convergence were also provided by the authors. This paper attempts to summarize my understanding of the previously mentioned topics as well as provides an extension to the numerical experiments part by applying the method on a real-world facial image dataset.

The following parts of this paper are structured as follow: Section 2 introduces NMF and provides a brief summary of MU and HALS. Section 3 discusses the computational cost of MU and HALS. Section 4 studies the main contribution of the paper, i.e. the Accelerated MU/HALS algorithm, a suggested modification is also provided. Section 5 provides the details of the new dataset, some running time results of the new algorithms as well as a study of key facial characteristics preservation of NMF. Lastly, Section 6 concludes the paper.

2. Background

2.1. Notations

All notations are the same as what we used during the lectures with some exceptions in order to avoid confusion, mostly for myself, when I attempt to discuss concepts from multiple papers at once. In this paper, NMF is formulated as $\mathbf{X} = \mathbf{W}\mathbf{H}$ instead of $\mathbf{X} = \mathbf{W}\mathbf{H}^T$. $\mathbf{X}_{i,}$ and $\mathbf{X}_{,l}$ represent the i-th row and j-column of matrix $\mathbf{X}$, respectively. $\mathbf{W}^{(k)}$ denotes the value of $\mathbf{W}$ after k iterations while $\mathbf{W}_t$ is the value of $\mathbf{W}$ at time t. $\mathbf{W}^k$. $\otimes$ is the element-wise (Hadamard) multiplication operator while $\odot$ represents the element-wise (Hadamard) division operator, sometimes $\frac{[.]}{[.]}$ is used instead of $\odot$, it has the same meaning, that is, $\frac{[\mathbf{A}]}{[\mathbf{B}]}$ is the same as $\mathbf{A} \odot \mathbf{B}$. $\mathbf{W} \geq 0$ means all elements of $\mathbf{W}$ is greater than or equal to 0. $\mathbf{W}_{ij}$ represents the value at i-th row and j-th column. δ_{ij} is the Kronecker delta, $\delta_{ij} = 1$ if $i = j$ and 0 everywhere else. $[\mathbf{W}]_+$ is the projection of $\mathbf{W}$ onto the nonnegative orthant. $\max\{\mathbf{x}, a\}$ means, for $a \in \mathbb{R}$ and a vector $\mathbf{x} \in \mathbb{R}^n$, $\max\{\mathbf{x}, a\}$ set all elements in $\mathbf{x}$ to a if they are less than a, otherwise they remain the same.

2.2. Nonnegative Matrix Factorization

Let $\mathbf{X} \in \mathbb{R}^{M\times N}$. NMF problem can be formulated as finding $\mathbf{W} \in \mathbb{R}^{M\times L}$, $\mathbf{H} \in \mathbb{R}^{L\times N}$ such that:

$$\mathbf{X} = \mathbf{W}\mathbf{H} \tag{1}$$

$$\text{s.t. } \mathbf{W} \geq 0, \mathbf{H} \geq 0 \tag{2}$$

Preferably, L is much smaller than $\min(M, N)$, then NMF leads to a low-rank approximation of $\mathbf{X}$. Since the perfect decomposition, that is, the equality in (1), does not exist in general, some characterizations of goodness-of-fit, i.e., how well $\mathbf{W}\mathbf{H}$ reconstructs $\mathbf{X}$, need to be utilized to evaluate the solutions of $\mathbf{W}$ and $\mathbf{H}$, it can be thought of as a loss function:

$$\min_{\mathbf{W},\mathbf{H}} \mathcal{L}(\mathbf{W}, \mathbf{H}) \tag{3}$$
$$\text{s.t. } \mathbf{W} \geq 0, \mathbf{H} \geq 0$$

While $\mathcal{L}$ can be any function, in general a suitable function of $\mathcal{L}$ is not convex in $(\mathbf{W}, \mathbf{H})$, and $\mathcal{L}$ is usually selected to be the sum of squares of the errors, which leads to the following minization problem

$$\min_{\mathbf{W},\mathbf{H}} \|\mathbf{X} - \mathbf{W}\mathbf{H}\|_F^2 \tag{4}$$
$$\text{s.t. } \mathbf{W} \geq 0, \mathbf{H} \geq 0$$

Since (4) is non-convex and in general an NP-hard problem, solving for global optimum is infeasible for high-dimensional problems. Thus, even though algorithms achieving global optimum does exist (Floudas & Visweswaran, 1993), in practice it is not scalable (Heiler et al., 2006). Hence, a local optimum, while being suboptimal, is more preferred due to its scalability. A class of such solutions is block coordinate descent, wherein at each step, one of the factors $\mathbf{W}$ or $\mathbf{H}$ is fixed while the other is optimized, this can be summarized as: randomly set starting point $\mathbf{W}^{(0)}$ and $\mathbf{H}^{(0)}$ then iteratively for $k = 1, 2, ...$ do:

1. $\mathbf{W}^{(k)} = \arg\min_{\mathbf{W}} \|\mathbf{X} - \mathbf{W}\mathbf{H}\|_F^2$ s.t. $\mathbf{W} \geq 0$
2. $\mathbf{H}^{(k)} = \arg\min_{\mathbf{H}} \|\mathbf{X} - \mathbf{W}\mathbf{H}\|_F^2$ s.t. $\mathbf{H} \geq 0$

By this process, an NMF problem can be broken down into multiple sub-problems, each is a nonnegative least squares problems that can be solved optimally, e.g. by using SVD or other methods mentioned in (Berry et al., 2007). An important observation here is that, at iteration $k+1$, optimal values of $\mathbf{W}^{(k)}$ and $\mathbf{H}^{(k)}$ are not required to perform optimization, this means we can approximately solve for local optimum in each sub-problem and achieve faster speed. In essence, we are making a tradeoff between accuracy and speed. Next sections introduce two such algorithms, namely MU from (Lee & Seung, 1999) and HALS (FastNMF/RRI) from (Li & Zhang, 2009) and (Ho, 2008).

2.3. Multiplicative Update

In the *Nature* paper (Lee & Seung, 1999) and later in (Seung & Lee, 2001), the authors introduced the Multiplicative update rule as follow:

$$\mathbf{W}^{(k+1)} = \mathbf{W}^{(k)} \otimes \frac{[\mathbf{X}{\mathbf{H}^{(k)}}^T]}{[\mathbf{W}^{(k)}\mathbf{H}^{(k)}{\mathbf{H}^{(k)}}^T]} \tag{5}$$

$$\mathbf{H}^{(k+1)} = \mathbf{H}^{(k)} \otimes \frac{[{\mathbf{W}^{(k)}}^T\mathbf{X}]}{[{\mathbf{W}^{(k)}}^T\mathbf{W}^{(k)}\mathbf{H}^{(k)}]} \tag{6}$$

This update rule was shown to be non-increasing under the loss function defined in (4), its convergence was difficult to prove but a proof to a slightly modified version was provided in (Lin, 2007a). While (5) and (6) were groundbreaking and have been the workhorse of NMF ever since, there are numerous issues associated with this method, namely:

- It requires $\mathbf{W}$ and $\mathbf{H}$ to be strictly positive, otherwise it will result in a dividing by 0 problem. In practice, even if $\mathbf{W}$ and $\mathbf{H}$ are strictly positive, they can contain small values that can cause numerical instability if not handled correctly.
- Since $\mathbf{W}$ and $\mathbf{H}$ need to be strictly positive, this means they are both dense matrix, thus making the computation very slow compared to other sparse algorithms.
- While this is advantageous compared to gradient descent based method in the sense that there is no parameters that need to be tuned, such as learning rate, the stopping criterion is still an open question.

These issues has motivated the development of different approaches to tackle NMF through the years. The next subsection introduces Hierarchical Alternating Least Squares method.

2.4. Hierarchical Alternating Least Squares

Essentially, the idea of HALS comes down to:

1. Randomly set starting point $\mathbf{W}^{(0)}$ and $\mathbf{H}^{(0)}$ to some $\epsilon \geq 0$.
2. Fix $\mathbf{H}$, for $j = 1, 2, .., L$, update a single column $\mathbf{W}_{,j}$ while keeping other columns fixed.
3. Fix $\mathbf{W}$, for $i = 1, 2, .., L$, update a single row $\mathbf{H}_{i,}$ while keeping other rows fixed.
4. Repeat step 2 and 3 until some stopping criteria is met.

By fixing everything except for a single column $\mathbf{W}_{,j}$ of $\mathbf{W}$, the minimization problem in (4) reduces to:

$$\min_{\mathbf{W}_{,j}} \ \|\mathbf{X} - \mathbf{W}\mathbf{H}\|_F^2 \ \text{ s.t. } \mathbf{W}_{,j} \geq 0 \tag{7}$$

Moving $\mathbf{W}_{,j}$ out of the multiplication in (7) gives:

$$\min_{\mathbf{W}_{,j}} \ \|\mathbf{X} - \sum_{k \neq j} \mathbf{W}_{,k}\mathbf{H}_{k,} - \mathbf{W}_{,j}\mathbf{H}_{j,}\|_F^2 \ \text{ s.t. } \mathbf{W}_{,j} \geq 0$$
$$\iff \min_{\mathbf{W}_{,j} \geq 0} \ \sum_{i=1}^{M} \|(\mathbf{X}_{i,} - \sum_{k \neq j} \mathbf{W}_{i,k}\mathbf{H}_{k,}) - \mathbf{W}_{i,j}\mathbf{H}_{j,}\|_F^2 \tag{8}$$

From Theorem 1 in (Li & Zhang, 2009), (8) has the following closed-form solution:

$$\mathbf{W}_{,j} = \max\{\frac{(\mathbf{X}\mathbf{H}^T)_{,j} - \mathbf{W}\mathbf{E}_{,j}}{(\mathbf{H}\mathbf{H}^T)_{jj}}, 0\} \tag{9}$$

where $\mathbf{E}$ is defined as $\mathbf{E}_{ab} = [\mathbf{H}\mathbf{H}^T]_{ab}(1 - \delta_{aj}\delta_{jb}),\ \forall a, b, j \in \{1, 2, ..., L\}$. From (9), $\mathbf{W}_{ij}$ can be obtained:

$$\mathbf{W}_{ij} = \max(0, \frac{(\mathbf{X}\mathbf{H}^T)_{ij} - \mathbf{W}_{i,}\mathbf{E}_{,j}}{(\mathbf{H}\mathbf{H}^T)_{jj}}) \tag{10}$$

Since each optimization problem $\mathbf{W}_{ij}$ is unrelated to each other (each row of $\mathbf{W}$ only contributes the corresponding row in the multiplication $\mathbf{W}\mathbf{H}$), the optimization problem (7) has the solution:

$$\mathbf{W}_{,j} = \begin{bmatrix} \max(0, [(\mathbf{X}\mathbf{H}^T)_{1j} - \mathbf{W}_{1,}\mathbf{E}_{,j}]/(\mathbf{H}\mathbf{H}^T)_{jj}) \\ \max(0, [(\mathbf{X}\mathbf{H}^T)_{2j} - \mathbf{W}_{2,}\mathbf{E}_{,j}]/(\mathbf{H}\mathbf{H}^T)_{jj}) \\ \dots \\ \max(0, [(\mathbf{X}\mathbf{H}^T)_{Mj} - \mathbf{W}_{M,}\mathbf{E}_{,j}]/(\mathbf{H}\mathbf{H}^T)_{jj}) \end{bmatrix} \tag{11}$$

(Ho, 2008) also provided a similar approach (chapter 4). In (Gillis & Glineur, 2012), in order to facilitate the analysis of computational cost, (11) was used to represent the update $\mathbf{W}^{(k+1)}$ as:

$$\mathbf{W}_{,j}^{(k+1)} = \max\{\frac{\mathbf{A}_{,j} - \sum_{i=1}^{j-1} \mathbf{W}_{,i}^{(k+1)}\mathbf{B}_{ij} - \sum_{i=j+1}^{L} \mathbf{W}^{(k)}\mathbf{B}_{ij}}{\mathbf{B}_{jj}}, 0\} \tag{12}$$

where $\mathbf{A} = \mathbf{X}{\mathbf{H}^{(k)}}^T$ and $\mathbf{B} = \mathbf{H}^{(k)}{\mathbf{H}^{(k)}}^T$ for $j = 1, 2, ..., L$. Since $\mathbf{H}$ and $\mathbf{W}$ are symmetric in this formulation, similar update rules can also be obtained for $\mathbf{H}$. From now on, we mostly focus on the analysis of $\mathbf{W}$.

Compared to MU in (5), the formulation in (11) has the following desirable advantage: there is no stability issues, at least theoretically according to Theorem 3 from (Li & Zhang, 2009). Furthermore, the resulting $\mathbf{W}$ and $\mathbf{H}$ from HALS can have 0 in their values, that means the spatial complexity of HALS should be lower than that of MU. That being said, since computers have their limitations in floating point representation, care still must be taken, especially when selecting ϵ. HALS's computational cost and whether it is actually faster than MU will be analyzed in the next section.

Algorithm 1 MU update for $\mathbf{W}^{(k)}$ (Gillis & Glineur, 2012)

```
.
1: $\mathbf{A} = \mathbf{X}{\mathbf{H}^{(k)}}^T$;
2: $\mathbf{B} = \mathbf{H}^{(k)}{\mathbf{H}^{(k)}}^T$;
3: $\mathbf{C} = \mathbf{W}^{(k)}\mathbf{B}$;
4: $\mathbf{W}^{(k+1)} = \mathbf{W}^{(k)} \otimes (\mathbf{A} \odot \mathbf{B})$;
```

Algorithm 2 MU update for $\mathbf{H}^{(k)}$

```
.
1: $\mathbf{A} = {\mathbf{W}^{(k)}}^T\mathbf{X}$;
2: $\mathbf{B} = {\mathbf{W}^{(k)}}^T\mathbf{W}^{(k)}$;
3: $\mathbf{C} = \mathbf{B}\mathbf{H}^{(k)}$;
4: $\mathbf{H}^{(k+1)} = \mathbf{H}^{(k)} \otimes (\mathbf{A} \odot \mathbf{B})$;
```

3. Computational cost analysis

3.1. Computational cost analysis of MU

Let $P = \text{nnz}(\mathbf{X})$, from Algorithm 1 we have:

- Step 1 is of $\mathcal{O}(MNL)$ or $\mathcal{O}(PL)$ if $\mathbf{X}$ is sparse
- Step 2 is of $\mathcal{O}(NL^2)$
- Step 3 is of $\mathcal{O}(ML^2)$
- Step 4 is of $\mathcal{O}(ML)$

So computational complexity of Algorithm 1 is $\mathcal{O}(NML + NL^2 + ML^2 + ML)$ if $\mathbf{X}$ is dense and $\mathcal{O}(PL + NL^2 + ML^2 + ML)$ if $\mathbf{X}$ is sparse. In (Gillis & Glineur, 2012) the authors counted the exact number of float operations so there was a factor of 2 at each step, I chose Big-O notation since it looks simpler. In general, $P = M \times N$ if $\mathbf{X}$ is dense, so the computational complexity of Algorithm 1 is:

$$\mathcal{O}(PL + NL^2 + ML^2 + ML) \tag{13}$$

For $\mathbf{H}$, the analysis is exactly the same, Algorithm 2 is included for reference (mostly for helping myself with coding). The computational complexity of Algorithm 2 is:

$$\mathcal{O}(PL + ML^2 + NL^2 + NL) \tag{14}$$

Algorithm 3 HALS update for $\mathbf{W}^{(k)}$ (Gillis & Glineur, 2012)

```
.
1: $\mathbf{A} = \mathbf{X}{\mathbf{H}^{(k)}}^T$;
2: $\mathbf{B} = \mathbf{H}^{(k)}{\mathbf{H}^{(k)}}^T$;
3: for $j = 1, 2, ..., L$ do
4:     $\mathbf{C}_{,j} = \sum_{l=1}^{j-1} \mathbf{W}_{,l}^{(k+1)}\mathbf{B}_{lj} + \sum_{l=j+1}^{L} \mathbf{W}_{,l}^{(k)}\mathbf{B}_{lj}$;
5:     $\mathbf{W}_{,j}^{(k+1)} = \max\{\frac{\mathbf{A}_{,j}-\mathbf{C}_{,j}}{\mathbf{B}_{jj}}, 0\}$;
6: end for
```

Algorithm 4 HALS update for $\mathbf{H}^{(k)}$

```
.
1: $\mathbf{A} = {\mathbf{W}^{(k)}}^T\mathbf{X}$;
2: $\mathbf{B} = {\mathbf{W}^{(k)}}^T\mathbf{W}^{(k)}$;
3: for $i = 1, 2, ..., L$ do
4:     $\mathbf{C}_{i,} = \sum_{l=1}^{i-1} \mathbf{H}_{l,}^{(k+1)}\mathbf{B}_{il} + \sum_{l=i+1}^{L} \mathbf{H}_{l,}^{(k)}\mathbf{B}_{il}$;
5:     $\mathbf{H}_{i,}^{(k+1)} = \max\{\frac{\mathbf{A}_{i,}-\mathbf{C}_{i,}}{\mathbf{B}_{ii}}, 0\}$;
6: end for
```

3.2. Computational cost analysis of HALS

For the updating of $\mathbf{W}$, I believe there are some errors with the notations in (Gillis & Glineur, 2012) (especially Algorithm 2 in there), Algorithm 3 included here contains some corrections. For the computational analysis, the first two steps are exactly the same as they are in Algorithm 1, step 4 is of $\mathcal{O}(M(L-1))$, and step 5 is simply of $\mathcal{O}(M)$, so the total computational complexity of Algorithm 3 is:

$$\mathcal{O}\Big(PL+NL^2+L\big(M(L-1)+M\big)\Big) = \mathcal{O}(PL+NL^2+ML^2) \tag{15}$$

Again, compared to (Gillis & Glineur, 2012), constants were dropped here since I am using Big-O. Similarly for $\mathbf{H}$ (Algorithm 4 is included for reference):

$$\mathcal{O}\Big(PL+ML^2+L\big(N(L-1)+N\big)\Big) = \mathcal{O}(PL+ML^2+NL^2) \tag{16}$$

3.3. Comparison between MU and HALS

From (13) and (15), it can be seen that the update of $\mathbf{W}$ by HALS is a bit faster compared to that of MU, by ML operations. The same holds for $\mathbf{H}$, HALS saves NL operations in comparison to MU.

An interesting and important observation here is that both MU and HALS share the same first two steps and the first step $\mathbf{A} = \mathbf{X}{\mathbf{H}^{(k)}}^T$ would be the most expensive one in term of computation if we assume $P \geq L(M+N)$, which is often the case since in practice we would like for $L \ll \min(M, N)$, this condition ensures that NMF achieves compression, in order words, it performs low-rank approximation. Next section discusses how (Gillis & Glineur, 2012) took advantage of this.

4. Accelerated NMF

Based on the analysis in the previous, since calculating $\mathbf{A} = \mathbf{X}{\mathbf{H}^{(k)}}^T$ seems to be the most expensive step for both MU and HALS, the authors of (Gillis & Glineur, 2012) proposed the following modifications:

- For MU, repeat steps 3 and 4 several times before updating $\mathbf{H}$

- For HALS, repeat steps 3 to 6 several times before updating $\mathbf{H}$

This proposal tries to maximize the utilization of each calculation of $\mathbf{A} = \mathbf{X}{\mathbf{H}^{(k)}}^T$, since this is supposed to be the most expensive step in our setting. To answer the question of how many steps we need to repeat before updating $\mathbf{H}$, a metric was introduced:

$$\rho = \frac{\text{total time spent for each update}}{\text{total spent in inner loop for each update}} \tag{17}$$

where inner loop means step 3-4 for MU, and steps 3-6 for HALS. Based on (17), we have:

$$\rho_{\mathbf{W}-\text{MU}} = \frac{PL + NL^2 + ML^2 + ML}{ML^2 + ML} = 1 + \frac{P + NL}{ML + M} \tag{18}$$

$$\rho_{\mathbf{H}-\text{MU}} = 1 + \frac{P + ML}{NL + N} \tag{19}$$

$$\rho_{\mathbf{W}-\text{HALS}} = 1 + \frac{P + NL}{M} \tag{20}$$

$$\rho_{\mathbf{H}-\text{HALS}} = 1 + \frac{P + ML}{N} \tag{21}$$

Since we are assuming $P \geq L(M + N)$:

$$\begin{aligned} \rho_{\mathbf{W}-\text{MU}} &\geq 1 + \frac{L(M+N) + NL}{ML + M} \\ &= \frac{ML + M + ML + NL + NL}{ML + M} \\ &= \frac{2ML + 2NL + M}{M(L+1)} \\ &> \frac{2ML}{M(L+1)} = \frac{2L}{L+1} \approx 2 \end{aligned} \tag{22}$$

For the dense $\mathbf{X}$, with the assumption that $L < \min(M, N)$, (22) becomes:

$$\begin{aligned} \rho_{\mathbf{W}-\text{MU}} &= 1 + \frac{MN + NL}{ML + M} \\ &= 1 + \frac{N(M+L)}{M(L+1)} > 1 + \frac{N}{L+1} \end{aligned} \tag{23}$$

From (22) and (23), we can see that in order to better utilize the calculated value of $\mathbf{A}$, i.e. making $\rho_{\mathbf{W}-\text{MU}}$ closer to 1, the inner loop (step 3-4) of MU can be repeatedly executed for $1 + \rho_{\mathbf{W}-\text{MU}}$ times before updating $\mathbf{H}$. Fig.1 shows that if $L \ll N$, which is often the case, much efficiency can be gained by using the mentioned procedure.

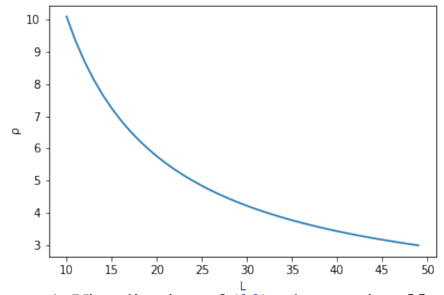

Figure 1. Visualization of (23) when setting $N = 500$

Based on this idea, (Gillis & Glineur, 2012) set the numbers of times to execute inner loop before updating $\mathbf{H}$ to:

$$\#\text{ inner loop iter} = 1 + \alpha\rho_{\mathbf{W}-\text{MU}} \tag{24}$$

where α is a new hyperparameter that acts as learning rate for this optimization technique. Since both MU and HALS are monotonically non-increasing [1], an error measurement is introduced:

$$E(t) = \frac{e(t) - e_{min}}{e(0) - e_{min}} \tag{25}$$

where $e(t) = \|\mathbf{X} - \mathbf{W}_t\mathbf{H}_t\|_F$ and e_{min} is the minimum value of $e(t)$ across all runs of all different algorithms, We have $0 \leq E(t) \leq 1$. This $E(t)$ is a normalized version of $e(k)$ across all algorithms, hence, it can be used to make comparison. Based on this, three algorithms were presented in (Gillis & Glineur, 2012), which I called: Naively accelerated MU/HALS (5), Less naively accelerated MU/HALS (6), and the main contribution of the paper, Accelerated MU/HALS (7). The conditions in line 6 & 15 of Algorithm 6 (also in Algorithm 7) are meant to stop executing the inner loop if the latest execution's improvement is less than the first execution's improvement, a new hyperparameter ϵ is also introduced to control this stopping condition.

Basically, Algorithm 5 simply uses ρ from (18)(19)(20)(21) as number of times to run the inner loop, a problem with this approach is, maybe $\mathbf{W}^{(k)}$ is already in near optimal state, so more time should be spent on updating $\mathbf{H}^{(k)}$, but since the number of iterations is fixed by ρ, we are wasting some computation resources. Algorithm 6 tries to fix this by introducing a stopping condition, but since there is no limitation of number of iterations, if we are not careful in selecting ϵ, this can run for a very long time without making any more substantial progress. Algorithm 7, the main contribution, simply is a combination of Algorithm 5 and Algorithm 6, stopping condition is used together with a fixed maximum number of iterations based on ρ.

Some comments I have with Algorithm 6 is that $\|\mathbf{W}^{(k,p+1)} - \mathbf{W}^{(k,p)}\|_F \leq \epsilon\|\mathbf{W}^{(k,1)} - \mathbf{W}^{(k,0)}\|_F$ is not

[1]The authors of (Gillis & Glineur, 2012) originally said they are monotonically decreasing but I am not quite sure if that is true for MU since there seems to be no proof of that.

exactly cheap to calculate, it is still about $6ML$ float operations. Furthermore, it is expected that the first few inner loop iterations are almost guaranteed to make good improvements. This means calculating the stopping conditions at every inner loop is kind of wasteful. An improvement I may have for this is to only occasionally check for stopping condition at the beginning, then as we get closer to ρ, the frequency of checking stopping condition increases accordingly, more like a step-decay in Machine Learning perspective.

This concludes my summary of the paper, Projected gradient part is omitted since it is just the same idea applied to (Lin, 2007b). The convergence to stationary part is also omitted here since it is also based on the idea from (Lin, 2007a) and did not exactly prove the convergence of Algorithm 7. Next section introduces the dataset and provides experimental results.

5. Experimental results

5.1. Dataset introduction

Figure 2. Small subset of private dataset

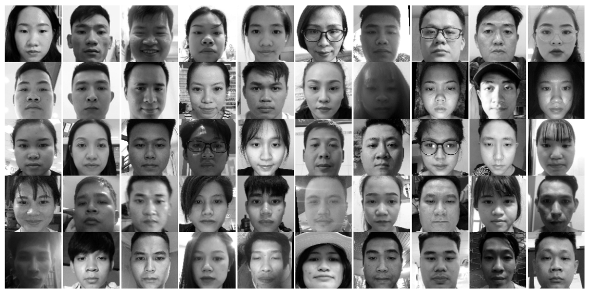

Figure 3. Pre-processed small subset of private dataset

A dataset of Vietnamese faces from my previous work was used to perform the experiments. This dataset is proprietary so it cannot be publicly shared, a small sample is given in Fig.2. This is one of the harder dataset compared to MSCeleb-1M (Guo et al., 2016) or CelebA (Liu et al., 2015) since there was almost no control (pose, lightning, rotation, etc.) when the photo was taken, the only requirement was it must be a selfie taken from a smartphone. Since this dataset is quite noisy, as can be seen from Fig.2, a pre-processing process is used:

1. Perform face alignment using a TensorRT-optimized(Vanholder, 2016) Centernet(Zhou et al., 2019)-based model that performs both face detection and landmarks estimation in one shot. Landmarks are used to perform affine transform that scales the facial image into a standardized 112×112 array. [2]
2. Images are then converted to grayscale to perform NMF.

Some pre-processed samples are shown in Fig.3. This dataset has a few hundred thousands photos, of which 10001 are randomly chosen for this study.

5.2. Experiments overview

Most of the original experiments from (Gillis & Glineur, 2012) will not be repeated here for the following reasons:

- The running time has been studied quite extensively, the authors also provided Matlab codes.
- The hardware has vastly changed since 2012, with the introduction of Advanced Vector Extension (AVX-2 and AVX-512) instructions into consumer-level CPUs, beginning with Intel's Haswell in 2013 [3], most matrix workloads have been greatly accelerated. Recently, with the advancement of GPU computing, this improvement has been pushed even further, sometimes by a thousand folds (Nishino & Loomis, 2017). Since hardware has been more and more optimized to specific workloads, making an objective comparison between algorithms, especially with matrix workload, has become a lot more tricky in recent years.

Instead, the following two experiments are carried out in this paper:

1. Apply Algorithm. 7 to the mentioned dataset, extract components and perform reconstruction.
2. See if NMF preserves key characteristics of the faces. To do this, a proprietary facial embedding model based on ArcFace (Deng et al., 2019) is used to calculate embedding vectors of original face and reconstructed face, the Euclidean between them is then calculated, if it is small, then NMF does preserve key characteristics used by the embedding model[4].

[2]This model is from part of my previous works at VNG Corp on facial recognition and camera surveillance, it is also proprietary, in-person demo available upon request. Any dataset that has standardized images can be used with the codes, I am interested in this dataset because I want to see if there is anything noteworthy about it from NMF perspective.

[3]I could not find a publication for this, technical specifications can be found at `https://www.intel.com/content/www/us/en/developer/articles/technical/intel-avx-512-instructions.html`

[4]The embedding model was trained with more than 300,000 images. While the training dataset was in RGB, by empirical

Some characteristics of the dataset are given in Table. 1. Some runtime information of $E(t)$ will also be shown, but the main focus of this section is the two experiments listed above. All experiments were run on AMD Ryzen 3900X CPU. For GPU-based tasks (face alignment and embedding vector extraction), NVIDIA RTX3080 was used. Codes are written in Python with the help of numpy, pytorch, and tensorflow.

5.3. Results

5.3.1. Accelerated MU and HALS

Fig.4 shows the evolution of $E(t)$ across different algorithms, Fig.5 shows the same information but it is plotted against #iterations instead of time. The zoomed in portion of Fig.4 demonstrates that HALS converges a lot faster than MU, confirming (Li & Zhang, 2009). It can also be noted that accelerated version of MU is a lot faster than the original one. But the same cannot be said for HALS, the accelerated version seems to be a little bit slower[5]. These figures are included to confirm the validity of my Python implementation[6].

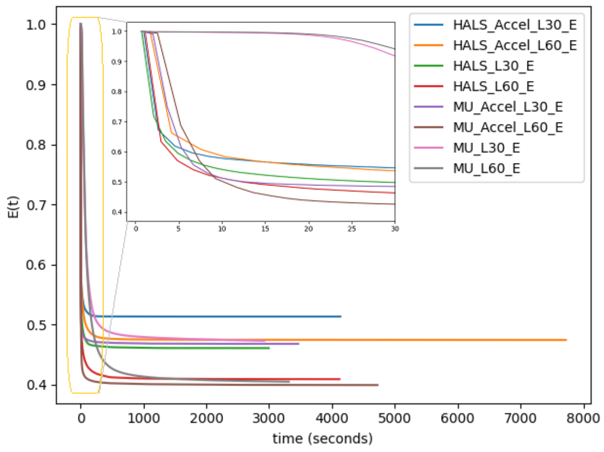

Figure 4. Evolution of $E(t)$

Fig.6 shows the reconstruction of faces shown in Fig.3, the corresponding components (columns of $\mathbf{W}$) are shown in Fig.7. These are generated by Accelerated MU with $L = 30$, $\alpha = 1$, $\epsilon = 0.1$, and run with 1000 updating iterations. Results for $L = 60$ are also shown in Fig.8 and Fig.9. It can be seen that higher value of L gives better reconstruction. It is noted that space saving by NMF is impressive, for e.g, storing $\mathbf{W}$ and $\mathbf{H}$ ($L = 60$) takes only 10.3MB in my machine, the original $\mathbf{X}$ takes 957MB, a 90 times improvement. Since the results for other algorithms are very similar, they will be omitted to save space.

An observation from the first experiment is that MU did not separate this dataset into different facial components as well as it was shown in (Lee & Seung, 1999). This may be due to the high variance of images, as shown in Fig.2. In order words, the dataset used by (Lee & Seung, 1999) is more "homogeneous", thus the task of separating them into key facial components is a bit easier.

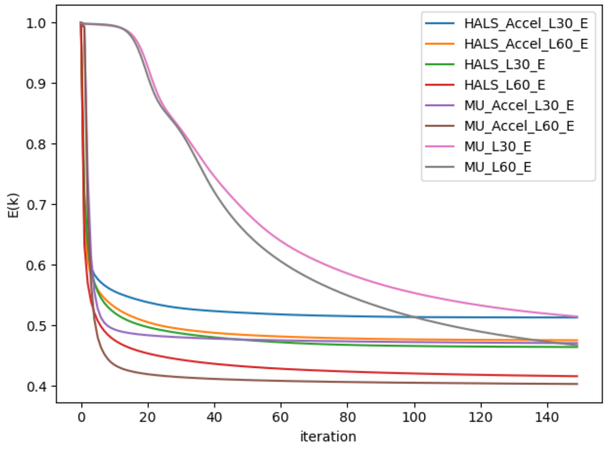

Figure 5. Evolution of $E(k)$

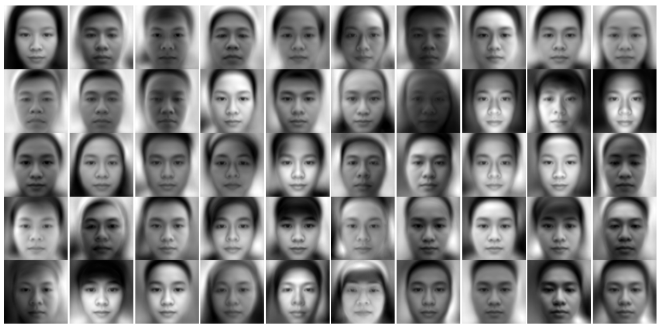

Figure 6. Reconstruction of Fig.3 by Acclerated MU, L=30

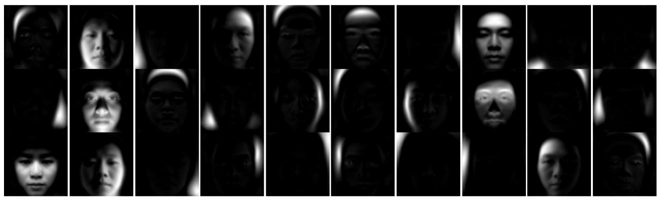

Figure 7. NMF components with MU, L=30

evidences, it works just as well with grayscale images during inference.

[5]The authors addressed this issue in (Gillis & Glineur, 2012), they changed the calculation for HALS since they claimed it was due to the fact that HALS had to loop through all columns of $\mathbf{W}$ and all rows of $\mathbf{H}$. I am not sure I agree with this, so I keep the same time calculation for both HALS and MU here.

[6]For HALS, my codes provide 2 implementations: one follows Algorithm. 3 exactly (named "*paper*" in the codes), the other one (named "*gillis*") adapts from (Gillis & Glineur, 2012) Matlab codes. The first one is much slower, this might be due numerical instability, since line 5 from Algorithm. 3 can cause some troubles with floating-point operations, I used the adapted version in this paper.

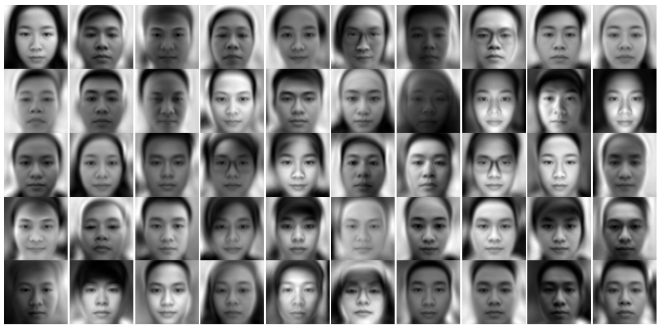

Figure 8. Reconstruction of Fig.3 by Acclerated MU, L=60

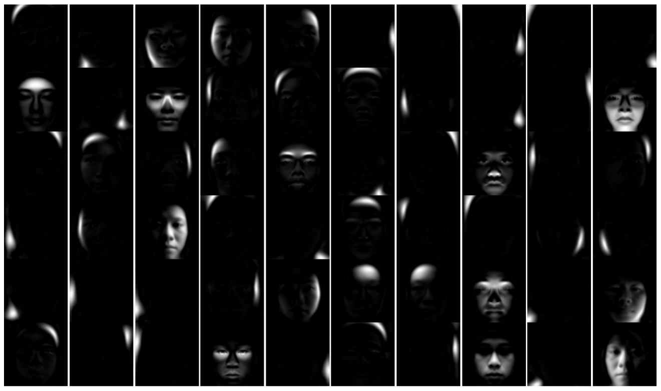

Figure 9. NMF components with MU, L=60

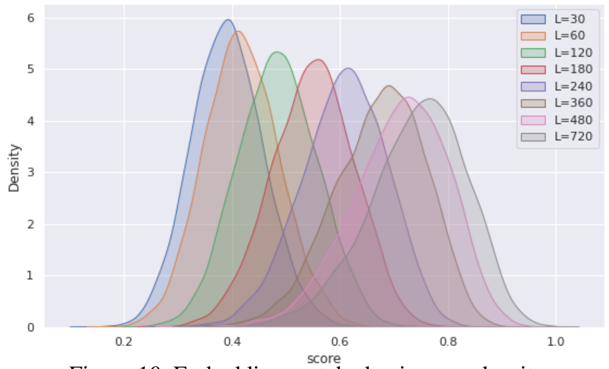

Figure 10. Embedding matched pair score density

5.3.2. Euclidean distances comparison

Since Euclidean distance is an unscaled measurement, a normalization process was used in the code (check function $get\_confidence\_score()$). This process converts distance to a value within 0-1, where 1 means they have the same embedding and 0 means they are totally different. Based on this, two comparisons are provided:

1. Embeddings from a raw image and its reconstructed version are compared, the distributions[7] for different values of L[8] are shown in Fig.10. This is the intra-group comparison, if NMF preserves key characteristics for embedding model, this score should be high.
2. Another important metric in facial recognition is how well the model separates two different faces, i.e the score should be low for negative pairs. To do this, 1000 faces are randomly selected from the set, for each of them, 10 other **different** faces (from the same set) are also randomly selected. The scores are then calculated and plotted using the same process as the first comparison. Calculations for different values of L as well as for the original dataset are provided in Fig.11. This is the inter-group comparison.

[7]Calculated using Seaborn kernel density estimation.

[8]Experiments were run for HALS with $L = \{30, 60, 120, 180, 240, 360, 480, 720\}$, each run was limit to 1.5 hours.

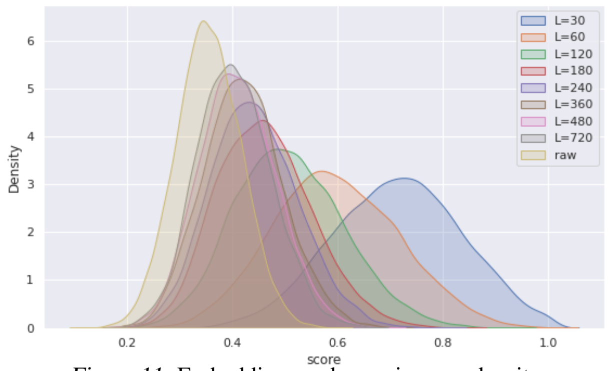

Figure 11. Embedding random pair score density

As can be seen from Fig.10, lower values of L perform badly in this test, in practice, a non-critical facial recognition demands the mean score to be around 0.7, while critical one needs at least 0.9, with the 3σ region less than 0.1. It is also noted that larger value of L, while leads to better scores, makes the distribution a lot wider. This means although we are improving the accuracy by increasing L, higher variance is also being introduced into the model. I am not sure if this is some kind of bias-variance tradeoffs, but Fig.10 looks quite like this issue.

Fig.11 tells the same story, albeit in a reverse direction. Higher values of L give better separation between negative pairs, it also reduces the variance. In order words, by increasing L, the model gets better at differentiating between 2 different faces, which coincides perfectly with my intuition. A plot for original dataset (label *raw*) is also provided in this figure, a near-perfect reconstruction should approach this distribution very closely.

Another interesting measurement is the tradeoff between compression ratio (storage of $\mathbf{W}$ and $\mathbf{H}$ divided by storage of $\mathbf{X}$) and accuracy (the mean scores). This is shown in Fig.12. It seems at around 10% compression ratio, the gains in accuracy start to become less substantial. An important thing to note here is that this embedding model never sees reconstructed images during the training phase, I speculate that if it is made aware of this fact during training, perhaps by mixing in a small amount of reconstructed images, the performance would be a lot better.

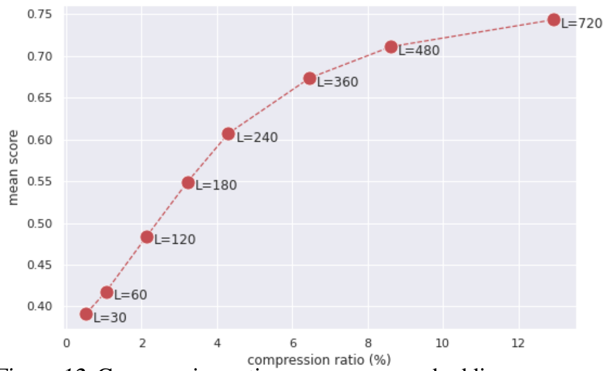

Figure 12. Compression ratio versus mean embedding score

6. Conclusions

This paper provided a summary of (Gillis & Glineur, 2012), an implementation in Python and an application to the new dataset with a focus on facial recognition characteristics. Although the results matched my expectation and intuition about NMF, it was still very fun to see its realizations in codes and in figures. I made some visualizations at https://arxiv.anvuong.dev/nmf/components.gif and https://arxiv.anvuong.dev/nmf/reconstructed.gif, they look pretty neat.

From the last experiment (the Euclidean distance), I do not know if it is possible to constrain NMF such that its factors preserve some desired properties of the original dataset, I have seen some works on Constrained NMF, but I am not sure if it can be used for the embedding vector problem, i.e the objective function is the minimization of distance between positive pair AND maximization of distance between negative pair. This seems like a fun problem to me.

This concludes this paper and also my works for the course Matrix Analysis. Thank you, I have learnt a lot.

Algorithm 5 Naively accelerated MU/HALS

```
1:  Initialize $\mathbf{W}^{(0)}$ and $\mathbf{H}^{(0)}$
2:
3:  $\mathbf{A} = \mathbf{X}{\mathbf{H}^{(k)}}^T,\ \mathbf{B} = \mathbf{H}^{(k)}{\mathbf{H}^{(k)}}^T$;
4:  for $p = 1, 2, ..., 1 + \alpha\rho_{\mathbf{W}}$ do
5:      Calculate $\mathbf{W}^{(k)}$ based on Algorithm 1 (step 3-4) or Algorithm 3 (step 3-6)
6:  end for
7:
8:  $\mathbf{A} = {\mathbf{W}^{(k)}}^T\mathbf{X},\ \mathbf{B} = {\mathbf{W}^{(k)}}^T\mathbf{W}^{(k)}$;
9:  for $p = 1, 2, ..., 1 + \alpha\rho_{\mathbf{H}}$ do
10:     Calculate $\mathbf{H}^{(k)}$ based on Algorithm 2 (step 3-4) or Algorithm 4 (step 3-6)
11: end for
```

Algorithm 6 Less naively accelerated MU/HALS

```
1:  Initialize $\mathbf{W}^{(0)}$ and $\mathbf{H}^{(0)}$
2:
3:  $\mathbf{A} = \mathbf{X}{\mathbf{H}^{(k)}}^T,\ \mathbf{B} = \mathbf{H}^{(k)}{\mathbf{H}^{(k)}}^T,\ p = 1$;
4:  while true do
5:      Calculate $\mathbf{W}^{(k)}$ based on Algorithm 1 (step 3-4) or Algorithm 3 (step 3-6)
6:      if $\|\mathbf{W}^{(k,p+1)} - \mathbf{W}^{(k,p)}\|_F \leq \epsilon\|\mathbf{W}^{(k,1)} - \mathbf{W}^{(k,0)}\|_F$ then
7:          break;
8:      end if
9:      $p = p + 1$;
10: end while
11:
12: $\mathbf{A} = {\mathbf{W}^{(k)}}^T\mathbf{X},\ \mathbf{B} = {\mathbf{W}^{(k)}}^T\mathbf{W}^{(k)},\ p = 1$;
13: while true do
14:     Calculate $\mathbf{H}^{(k)}$ based on Algorithm 2 (step 3-4) or Algorithm 4 (step 3-6)
15:     if $\|\mathbf{H}^{(k,p+1)} - \mathbf{H}^{(k,p)}\|_F \leq \epsilon\|\mathbf{H}^{(k,1)} - \mathbf{H}^{(k,0)}\|_F$ then
16:         break;
17:     end if
18:     $p = p + 1$;
19: end while
```

Table 1. Some characteristics of the dataset

METHOD	M	N	L	$\rho_{\mathbf{W}}$	$\rho_{\mathbf{H}}$
MU	12544	10001	30	324	406
HALS	12544	10001	30	10025	12582
MU	12544	10001	60	165	207
HALS	12544	10001	60	10049	12620

References

Abdi, H. and Williams, L. J. Principal component analysis. *Wiley interdisciplinary reviews: computational statistics*, 2(4):433–459, 2010.

Arora, S., Ge, R., Kannan, R., and Moitra, A. Computing a nonnegative matrix factorization—provably. *SIAM Journal on Computing*, 45(4):1582–1611, 2016.

Berry, M. W., Browne, M., Langville, A. N., Pauca, V. P., and Plemmons, R. J. Algorithms and applications for approximate nonnegative matrix factorization. *Computational statistics & data analysis*, 52(1):155–173, 2007.

Bioucas-Dias, J. M., Plaza, A., Dobigeon, N., Parente, M., Du, Q., Gader, P., and Chanussot, J. Hyperspectral unmixing overview: Geometrical, statistical, and sparse regression-based approaches. *IEEE journal of selected topics in applied earth observations and remote sensing*, 5(2):354–379, 2012.

Cichocki, A., Zdunek, R., and Amari, S.-i. Hierarchical

Algorithm 7 Accelerated MU/HALS

```
1: Initialize $\mathbf{W}^{(0)}$ and $\mathbf{H}^{(0)}$
2:
3: $\mathbf{A} = \mathbf{X}\mathbf{H}^{(k)^T},\ \mathbf{B} = \mathbf{H}^{(k)}\mathbf{H}^{(k)^T}$;
4: for $p = 1, 2, ..., 1 + \alpha\rho_{\mathbf{W}}$ do
5:     Calculate $\mathbf{W}^{(k)}$ based on Algorithm 1 (step 3-4) or Algorithm 3 (step 3-6)
6:     if $\|\mathbf{W}^{(k,p+1)} - \mathbf{W}^{(k,p)}\|_F \leq \epsilon\|\mathbf{W}^{(k,1)} - \mathbf{W}^{(k,0)}\|_F$ then
7:         break;
8:     end if
9: end for
10:
11: $\mathbf{A} = \mathbf{W}^{(k)^T}\mathbf{X},\ \mathbf{B} = \mathbf{W}^{(k)^T}\mathbf{W}^{(k)}$;
12: for $p = 1, 2, ..., 1 + \alpha\rho_{\mathbf{H}}$ do
13:     Calculate $\mathbf{H}^{(k)}$ based on Algorithm 2 (step 3-4) or Algorithm 4 (step 3-6)
14:     if $\|\mathbf{H}^{(k,p+1)} - \mathbf{H}^{(k,p)}\|_F \leq \epsilon\|\mathbf{H}^{(k,1)} - \mathbf{H}^{(k,0)}\|_F$ then
15:         break;
16:     end if
17: end for
```

als algorithms for nonnegative matrix and 3d tensor factorization. In *International Conference on Independent Component Analysis and Signal Separation*, pp. 169–176. Springer, 2007.

Deng, J., Guo, J., Xue, N., and Zafeiriou, S. Arcface: Additive angular margin loss for deep face recognition. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 4690–4699, 2019.

Floudas, C. A. and Visweswaran, V. Primal-relaxed dual global optimization approach. *Journal of Optimization Theory and Applications*, 78(2):187–225, 1993.

Gillis, N. and Glineur, F. Nonnegative factorization and the maximum edge biclique problem. *arXiv preprint arXiv:0810.4225*, 2008.

Gillis, N. and Glineur, F. Accelerated multiplicative updates and hierarchical ALS algorithms for nonnegative matrix factorization. *Neural Computation*, 24(4):1085–1105, apr 2012. doi: 10.1162/neco_a_00256. URL https://doi.org/10.1162%2Fneco_a_00256.

Guo, Y., Zhang, L., Hu, Y., He, X., and Gao, J. Ms-celeb-1m: A dataset and benchmark for large-scale face recognition. In *European conference on computer vision*, pp. 87–102. Springer, 2016.

Heiler, M., Schnörr, C., Bennett, K. P., and Parrado-Hernández, E. Learning sparse representations by nonnegative matrix factorization and sequential cone programming. *Journal of Machine Learning Research*, 7(7), 2006.

Ho, N.-D. *Nonnegative matrix factorization algorithms and applications*. PhD thesis, Citeseer, 2008.

Kolter, J., Batra, S., and Ng, A. Energy disaggregation via discriminative sparse coding. *Advances in neural information processing systems*, 23, 2010.

Kolter, J. Z. and Johnson, M. J. Redd: A public data set for energy disaggregation research. In *Workshop on data mining applications in sustainability (SIGKDD), San Diego, CA*, volume 25, pp. 59–62, 2011.

Langley, P. Crafting papers on machine learning. In Langley, P. (ed.), *Proceedings of the 17th International Conference on Machine Learning (ICML 2000)*, pp. 1207–1216, Stanford, CA, 2000. Morgan Kaufmann.

Lee, D. D. and Seung, H. S. Learning the parts of objects by non-negative matrix factorization. *Nature*, 401(6755): 788–791, 1999.

Li, L. and Zhang, Y.-J. Fastnmf: highly efficient monotonic fixed-point nonnegative matrix factorization algorithm with good applicability. *Journal of Electronic Imaging*, 18(3):033004, 2009.

Lin, C.-J. On the convergence of multiplicative update algorithms for nonnegative matrix factorization. *IEEE Transactions on Neural Networks*, 18(6):1589–1596, 2007a.

Lin, C.-J. Projected gradient methods for nonnegative matrix factorization. *Neural computation*, 19(10):2756–2779, 2007b.

Liu, Z., Luo, P., Wang, X., and Tang, X. Deep learning face attributes in the wild. In *Proceedings of International Conference on Computer Vision (ICCV)*, December 2015.

Ma, W.-K., Bioucas-Dias, J. M., Chan, T.-H., Gillis, N., Gader, P., Plaza, A. J., Ambikapathi, A., and Chi, C.-Y. A signal processing perspective on hyperspectral unmixing: Insights from remote sensing. *IEEE Signal Processing Magazine*, 31(1):67–81, 2013.

Nishino, R. and Loomis, S. H. C. Cupy: A numpy-compatible library for nvidia gpu calculations. *31st confernce on neural information processing systems*, 151, 2017.

Seung, D. and Lee, L. Algorithms for non-negative matrix factorization. *Advances in neural information processing systems*, 13:556–562, 2001.

Vanholder, H. Efficient inference with tensorrt. In *GPU Technology Conference*, volume 1, pp. 2, 2016.

Zhou, X., Wang, D., and Krähenbühl, P. Objects as points. *arXiv preprint arXiv:1904.07850*, 2019.