# Coordinate Descent Algorithm for Nonlinear Matrix Decomposition with the ReLU function

**Conference Paper** · June 2024

CITATIONS
0

**5 authors**, including:

Atharva Awari
Université de Mons
**1** PUBLICATION **0** CITATIONS
SEE PROFILE

Nicolas Gillis
Université de Mons
**197** PUBLICATIONS **4,112** CITATIONS
SEE PROFILE

# Coordinate Descent Algorithm for Nonlinear Matrix Decomposition with the ReLU function

Atharva Awari Harrison Nguyen Samuel Wertz Arnaud Vandaele Nicolas Gillis
University of Mons
Rue de Houdain 9, 7000 Belgium
{atharvaabhijit.awari, nicolas.gillis, arnaud.vandaele}@umons.ac.be, {harrison.nguyen,samuel.wertz}@student.umons.ac.be

***Abstract*—Nonlinear Matrix Decompositions (NMD) solve the following problem: Given a matrix $X$, find low-rank factors $W$ and $H$ such that $X \approx f(WH)$, where $f$ is an element-wise nonlinear function. In this paper, we focus on the case when $f$ is the rectified linear unit (ReLU) activation, that is, when $f(\cdot) = \max(0, \cdot)$, which is referred to as ReLU-NMD. All state-of-the-art algorithms for ReLU-NMD have been designed to solve a reformulation of ReLU-NMD. It turns out that this reformulation leads to a non-equivalent problem, and hence to suboptimal solutions. In this paper, we propose a coordinate-descent (CD) algorithm designed to solve ReLU-NMD directly. This allows us to compute more accurate solutions, with smaller error. This is illustrated on synthetic and real-world datasets.**

***Index Terms*—Nonlinear Matrix Decomposition (NMD), Rectified Linear Unit (ReLU), Coordinate Descent (CD).**

## I. Introduction

Low-rank matrix approximations (LRMAs) are widely used in the fields of machine learning and data analysis. When dealing with a large data set stored in a matrix $X$, it is customary to approximate it by a product of low-rank factors and hence to solve the following problem: Given $X \in \mathbb{R}^{m \times n}$ and $r \ll \min(m, n)$, find two factors, $W \in \mathbb{R}^{m \times r}$ and $H \in \mathbb{R}^{r \times n}$, such that $X \approx WH$. In data analysis, it is equivalent to linear dimensionality reduction, meaning that each column of the matrix $X$ is approximated by a linear combination of the columns of $W$, with the weights being provided by the corresponding column of $H$: for all $j$,

$$X(:, j) \quad \approx \quad \sum_{k=1}^{r} W(:, k)\, H(k, j). \tag{1}$$

Low-rank decompositions allow one to obtain a compressed representation of the original data and to automatically extract important features. The truncated singular value decomposition (TSVD) [1] and nonnegative matrix factorization (NMF) [2] are notable examples of low-rank matrix approximations.

Recently, there has been growing interest in Nonlinear Matrix Decompositions (NMDs) [3]–[5]. These nonlinear decompositions are gaining more and more popularity as they lead to more expressive models than their linear counterparts. Moreover, NMDs are more appropriate in modeling some nonlinear real-world phenomena.

Authors acknowledge the support by the European Union (ERC consolidator, eLinoR, no 101085607), by the the F.R.S.-FNRS under the PDR project T.0097.22, and by the Francqui Foundation.

Let us define NMD: Given $X \in \mathbb{R}^{m \times n}$ and $r \ll \min(m, n)$, find two factors, $W \in \mathbb{R}^{m \times r}$ and $H \in \mathbb{R}^{r \times n}$, such that

$$X \approx f(WH),$$

where $f(\cdot)$ is an element-wise nonlinear function. In general, the task is to solve the following optimization problem:

$$\min_{W,H} \|X - f(WH)\|_F^2. \tag{2}$$

In this paper, we focus on the case $f(\cdot) = \max(0, \cdot)$, that is, $f$ is the ReLU function, which is frequently used as an activation function in the hidden layers of neural networks. The main problem addressed in this work is therefore: Given $X \in \mathbb{R}^{m \times n}$ and $r \ll \min(m, n)$, solve

$$\min_{W,H} \|X - \max(0, WH)\|_F^2. \tag{3}$$

We refer to problem (3) as ReLU-NMD. ReLU-NMD is able to approximate sparse nonnegative data, such as sparse images and documents data sets, very effectively, that is, with significantly smaller error than linear models, such TSVD or NMF, using the same factorization rank. Moreover, ReLU-NMD was shown to be able to recover a faithful low-dimensional representation of high-dimensional data [4], and to be a meaningful model for matrix completion [6].

The objective function in (3) is nonconvex and nondifferentiable, which makes it very difficult to solve directly. In [4], Saul proposed a latent-variable model for ReLU-NMD:

$$\min_{Z,W,H} \|Z - WH\|_F^2 \quad \text{such that} \quad \max(0, Z) = X. \tag{4}$$

This altered formulation introduces an additional latent variable $Z$ and has the advantage of moving the nonlinearity from the objective function to the constraint, which opens the possibility of exploring new solution strategies; in particular simple alternating scheme optimizing $Z$, $W$ and $H$ alternatively [4], [7], [8]. However, the latent-variable formulation is not equivalent to the original ReLU-NMD problem, that is, an optimal solution to (4) is not necessarily optimal for (3), and vice versa. In particular, when $X$ contains negative entries, (4) is infeasible while (3) is still meaningful. The proof in the case $X \geq 0$ will be provided in the longer version of the paper, but we will illustrate the differences between (3) and (4) through numerical experiments.

**Contribution and outline:** All the previously known algorithms to tackle ReLU-NMD solve the latent variable

model (4). Our main contribution in this paper is to propose a new algorithm based on coordinate descent (CD) that addresses (3) directly, for the first time, allowing our method to reach better solutions than the state-of-the-art algorithms.

The paper is organized as follows. Section II provides a brief overview of the previous works and algorithms. In Section III, we introduce our proposed CD algorithm for ReLU-NMD (3). In Section IV, we empirically show the effectiveness of our method through various experiments on real data sets and show that CD provides better solutions than existing algorithms.

## II. Previous works

In this section, we provide a brief overview of the existing algorithms designed to solve the latent-variable formulation of ReLU-NMD (4). For the first two algorithms (see subsections II-A and II-B), Saul denotes the low-rank approximation $\Theta = WH \in \mathbb{R}^{m\times n}$, and describes iterative algorithms that optimize over $\Theta$ directly. So, (4) is formulated as:

$$\min_{Z,\Theta} \|Z-\Theta\|_F^2 \quad \text{such that} \quad \begin{cases} \operatorname{rank}(\Theta) = r, \\ \max(0,Z) = X. \end{cases} \tag{5}$$

### A. Naive scheme and extrapolation

This is an alternating optimization scheme over the matrix variables $Z$ and $\Theta$ in (5). When $\Theta$ is fixed, the optimal $Z$ is given by: for all $i, j$,

$$Z_{ij} = \begin{cases} X_{ij} & \text{if} \quad X_{ij} > 0, \\ \min(0,\Theta_{ij}) & \text{if} \quad X_{ij} = 0. \end{cases} \tag{6}$$

When $Z$ is fixed, the optimal $\Theta$ is given by the rank-$r$ TSVD of $Z$. In his follow-up paper [7], Saul also proposed a way to accelerate the naive scheme by introducing an additional momentum term on the update of $Z$ with a fixed momentum parameter. Let $Z^k$ denote the value of $Z$ after the $k$th iteration, then momentum is applied as follows:

$$Z^{k+1} \leftarrow Z^{k+1} + \alpha(Z^k - Z^{k-1}), \text{ where } \alpha \in (0,1). \tag{7}$$

### B. Expectation-Maximization (EM)

The second algorithm by Saul is a sophisticated EM method. For each $\Theta_{ij}$, a Gaussian latent variable $\tilde{Z}_{ij}$ is defined as $\tilde{Z}_{ij} \sim \mathcal{N}(\Theta_{ij}, \sigma^2)$. The observation $Z$ is a sample of $\tilde{Z}$, and the matrix $X$ is obtained from the elementwise nonlinear mapping of $Z$, that is $X = \max(0, Z)$. The model is then estimated by maximizing the likelihood of the observation $X$ in terms of the parameters, namely the matrix $\Theta$ and the variance $\sigma^2$. The overall log-likelihood under this model is given by

$$\log P(X|\Theta,\sigma^2) = \Sigma_{ij} \log P(X_{ij}|\Theta_{ij},\sigma^2).$$

This sum is maximized to estimate the parameters $\Theta$ and $\sigma^2$. To do this, EM is used. The steps are rather complicated, and we refer the interested readers to [4] for more details.

*Note:* The M-step of the algorithm requires the computation of a rank-$r$ TSVD and hence the computational cost is the same as the Naive scheme. Also, an additional momentum term can be added on the update of $Z$, as in (7).

### C. Aggressive-momentum algorithm (A-NMD)

The Polyak-type extrapolation step (7) uses a fixed momentum parameter $\alpha$ and only extrapolates the variable $Z$. The parameter $\alpha$ is kept fixed throughout the execution of the algorithm, which makes the performance more sensitive to the initial choice of $\alpha$. In [8], Seraghiti et al. propose a more aggressive strategy in order to accelerate the Naive scheme:

$$Z^{k+1} \leftarrow Z^{k+1} + \beta_k(Z^{k+1} - Z^k),$$

where the momentum parameter $\beta_k$ is adaptively adjusted at each iteration $k$ based on the objective function. Moreover, extrapolation is used for both variables $\Theta$ and $Z$. Since the parameter $\beta_k$ is chosen adaptively at each step, the algorithm is less sensitive to the initial choice of the parameter. A-NMD was shown to be significantly faster than the Naive scheme.

### D. Three-blocks algorithm (3B-NMD)

All algorithms previously described for the ReLU-NMD problem require the computation of a rank-$r$ TSVD at each iteration. This is mainly because they work with a rank-$r$ factor $\Theta$ directly and not with low-rank factors $W$ and $H$. Computation of TSVD at each iteration is an expensive step, especially when it comes to large-size data.

In order to bypass this relatively expensive step, Seraghiti et al. propose to replace $\Theta$ by the product $WH$ and solve (4) directly. The subproblems for the variables $W$ and $H$ have closed-form solutions; in fact, it amounts to solving matrix least squares problems. This reduces the number of operations from $O(mnr^2)$ operations for the TSVD to $O(mnr)$ operations for the matrix least squares problems. To speed-up 3B-NMD, an acceleration step equivalent to (7) was also used after updating both $Z$ and $\Theta$. The 3B-NMD algorithm was shown to be significantly faster than all other algorithms when handling large-size data; see [8] for further details.

## III. Proposed algorithms

We would like to emphasize that all algorithms described in Section II solve the latent variable formulation (4) of the ReLU-NMD. In this Section, we propose a new algorithm based on the coordinate-descent (CD) scheme dealing directly with the original ReLU-NMD problem (3).

CD will update the factors $W$ and $H$ alternately. Since the problem is symmetric, that is, $X \approx \max(0, WH)$ is equivalent to $X^T \approx \max(0, H^T W^T)$, we describe the update of $H$ when $W$ is fixed; the update of $W$ can be obtained by symmetry. Moreover, as in standard low-rank approximations, the problem in $H$ is separable by columns, that is, we can solve separately for each $j$:

$$\min_{H(:,j)} \|X(:,j) - \max(0, WH(:,j))\|_F^2. \tag{8}$$

We propose to solve (8) by updating one entry at a time, which is CD. Suppose all entries of $H(:,j)$ are fixed except the $i$th one, that is, $H(i,j)$. The one-variable subproblem can be written as follows:

$$\min_{x\in\mathbb{R}} f(x) = \|c - \max(0, b + ax)\|_2^2, \tag{9}$$

where $x = H(i,j) \in \mathbb{R}$, $c = X(:,j) \in \mathbb{R}^m$, $b = \sum_{k=1,k\neq i}^{r} W(:,k)H(k,j) \in \mathbb{R}^m$, and $a = W(:,i) \in \mathbb{R}^m$. By expanding the Euclidean norm, we observe that the objective in (9) is the sum of $m$ one-variable functions:

$$f(x) = \sum_{t=1}^{m} f_t(x) \text{ with } f_t(x) = (c_t - \max(0, b_t + a_t x))^2 . \tag{10}$$

We can assume w.l.o.g. that $a_t \neq 0$ since $f_t$ is constant in $x = H(i,j)$ otherwise, and hence can be removed. Each function $f_t$ has a break point at $x_t = \frac{-b_t}{a_t}$. When $a_t > 0$, $f_t$ is a constant to the left of the break point and quadratic to its right, and vice versa when $a_t < 0$. More precisely, we have

$$a_t > 0: \quad f_t(x) = \begin{cases} c_t^2 & \text{when } x \leq \frac{-b_t}{a_t}, \\ (c_t - b_t - a_t x)^2 & \text{when } x \geq \frac{-b_t}{a_t}, \end{cases}$$

$$a_t < 0: \quad f_t(x) = \begin{cases} c_t^2 & \text{when } x \geq \frac{-b_t}{a_t}, \\ (c_t - b_t - a_t x)^2 & \text{when } x \leq \frac{-b_t}{a_t}. \end{cases}$$

Between two consecutive break points, the function (10) is the sum of constant and quadratic functions $f_t$. From this observation, a simple strategy arises to find the global optimum of (9): consider the value of the objective at all breakpoints and at the minimum, if it exists, of the sum of the active quadratics between two consecutive break points.

In the following, and w.l.o.g., we assume that the functions $f_t$ are sorted such that

$$\frac{-b_t}{a_t} \leq \frac{-b_{t+1}}{a_{t+1}}.$$

Next, we explain how we find the minimizer of $f$ within the $p$th interval $I_p = (\frac{-b_p}{a_p}, \frac{-b_{p+1}}{a_{p+1}})$, with $p = 1, ..., m-1$.

Let us define two sets of indices: the set $T_+(p)$ with the indices of the functions $f_t$, $t \leq p$, with an *active* quadratic part on $I_p$,

$$T_+(p) = \{t \in \{1, ..., m\} \mid t \leq p, a_t > 0\},$$

and the set $T_-(p)$ with the indices of the functions $f_t$, $t \geq p+1$, with an *active* quadratic part on $I_p$,

$$T_-(p) = \{t \in \{1, ..., m\} \mid t \geq p+1, a_t < 0\}.$$

Let us also define $T_p = T_+(p) \cup T_-(p)$. Similarly we also define the 0th interval as $(-\infty, \frac{-b_1}{a_1}]$, and the $m$th as $[\frac{-b_m}{a_m}, \infty)$.

With this notation, the derivative of $f$ in the interior of the $p$th interval is :

$$f'(x) = 2 \sum_{p \in T_p} a_p(a_p x + b_p - c_p) \quad \text{for } x \in \left] \frac{-b_p}{a_p}, \frac{-b_{p+1}}{a_{p+1}} \right[ . \tag{11}$$

Considering the interval $I_p$, the optimum lies at one of the break points $\frac{-b_p}{a_p}$, $\frac{-b_{p+1}}{a_{p+1}}$ or in the interior when $f'(x)$ given in (11) is equal to 0 on the interval. When the global minimizer of $f(x)$ is on the left (resp. right) of the interval, the solution is $\frac{-b_p}{a_p}$ (resp. $\frac{-b_{p+1}}{a_{p+1}}$). Finally, the minimizer of $f(x)$ on the interval is given by the expression:

$$\min\left(\frac{-b_{p+1}}{a_{p+1}}, \max\left(\frac{-b_p}{a_p}, \frac{a(T_p)^T(c(T_p) - b(T_p))}{\|a(T_p)\|_2^2}\right)\right).$$

The optimal solution can be computed for every interval $p = 1, 2, \ldots, m-1$, and the overall optimum value is the corresponding update for $x = H(i,j)$. The CD algorithm optimizes alternatively over the entries of the factors $W$ and $H$, see Algorithm 1.

**Algorithm 1:** CD algorithm for ReLU-NMD

**Require:** $X \in \mathbb{R}^{m\times n}$, $W^{(0)} \in \mathbb{R}^{m\times r}$, $H^{(0)} \in \mathbb{R}^{r\times n}$, maxit.
**Ensure:** Two matrices $W \in \mathbb{R}^{m\times r}$ and $H \in \mathbb{R}^{r\times n}$ s.t. $X \approx \max(0, WH)$.

1: **for** $k = 1, \ldots,$ maxit **do**
2: $\quad H^{(k)} = \text{Algorithm 2}(X, W^{(k-1)}, H^{(k-1)})$
3: $\quad {W^{(k)}}^T = \text{Algorithm 2}(X^T, {H^{(k)}}^T, {W^{(k-1)}}^T)$
4: **end for**

**Algorithm 2:** CD algorithm for $H$

**Require:** $X \in \mathbb{R}^{m\times n}$, $W \in \mathbb{R}^{m\times r}$, $H^{(0)} \in \mathbb{R}^{r\times n}$
**Ensure:** $H^{(1)}$ is obtained as one cyclic CD update for $\min_{H\geq 0} \|X - \max(0, WH)\|_F^2$ starting at $H^{(0)}$.

1: $[r, n] = \text{size}(H^{(0)})$
2: **for** $j = 1, \ldots, n$ **do**
3: $\quad c = X(:, j)$
4: $\quad b = WH^{(0)}(:, j)$
5: $\quad$ **for** $i = 1, \ldots, r$ **do**
6: $\quad\quad a = W(:, i)$
7: $\quad\quad b = b - W(:, i)H^{(0)}(i, j)$
8: $\quad\quad H^{(1)}(i, j) = \text{argmin}_x \|c - \max(0, b + ax)\|_2^2$
9: $\quad\quad b = b + W(:, i)H^{(1)}(i, j)$
10: $\quad$ **end for**
11: **end for**

### A. Complexity and convergence

In Algorithm 2, the most computationally expensive step is step 8 where a single entry $H(i,j)$ is being updated. This is mainly because this step requires the sorting of an $m-$length vector requiring $\mathcal{O}(m \log m)$ operations. Considering the inner and the outer loops through steps 2-11 in Algorithm 2, the cost to update all entries of $H$ once is $\mathcal{O}(mnr \log m)$. Similarly, the cost to update all entries of $W$ is $\mathcal{O}(mnr \log n)$. Hence, in total, the CD algorithm requires $\mathcal{O}(mnr \max(\log m, \log n))$ operations to update once every variable.

Coordinate descent guarantees the monotonicity of the the objective function which is bounded below, and hence the objective function values converge. However, since ReLU-NMD (3) is non-convex and non-smooth, there are no convergence guarantees to a critical point for the sequence of iterates $(W^{(k)}, H^{(k)})$. Designing algorithms with such guarantees is a topic for further research.

## IV. Numerical Experiments

We have implemented Algorithm 1 in MATLAB, Julia and C++, to compare implementations in different languages, and also allow people to use ReLU-NMD in different languages. The code is available from https://gitlab.com/Atharva05/coordinate-descent-for-relu-nmd.git.

### A. Comparison of MATLAB, Julia and C++

Let us use simple synthetic data. The matrix $X \in \mathbb{R}^{m\times n}$ is generated as $X = \max(0, WH)$, where the entries of $W \in \mathbb{R}^{m\times r}$ and $H \in \mathbb{R}^{r\times n}$ are generated from the normal distribution, that is, `W = randn(m,r)` and `H = randn(r,n)` in MATLAB. The resulting matrix $X$ is around $50\%$ sparse. The different algorithms are stopped when

$$\text{relative error} = \frac{\|X - \max(0, WH)\|_F}{\|X\|_F} \leq 10^{-4}. \quad (12)$$

Table I compares the three implementations, with averaged values of time and number of iterations[1] over 5 different synthetic matrices, with the rank-$r$ TSVD as the initialization.

| | | | MATLAB | JULIA | C++ |
|---|---|---|---|---|---|
| Size | rank | iter | time(sec) | time(sec) | time(sec) |
| 50 × 50 | 5 | 87 | 0.42 | **0.14** | 0.22 |
| 200 × 200 | 10 | 37 | 2.75 | 1.77 | **1.62** |
| 500 × 500 | 25 | 25 | 23.82 | 24.07 | **19.53** |
| 1000 × 1000 | 32 | 21 | 97.37 | 98.25 | **90.49** |

TABLE I: Average computational time and iterations needed to satisfy condition (12) on synthetic data.

We observe that there is not a significant difference in timings between the three implementations.

**MATLAB vs. Julia with Multi-threading:** Since the rows (resp. columns) of the factor $W$ (resp. $H$) are updated independently, we can use multi-threading to update several columns in parallel to speed-up the process. The number of threads was set to 8 in both MATLAB and Julia. We did not include C++ as it is more tricky to activate multi-threading. Table II reports the results.

| | | | MATLAB | JULIA |
|---|---|---|---|---|
| Size | rank | iter | time(sec) | time(sec) |
| 50 × 50 | 5 | 87 | 8.51 | **0.14** |
| 200 × 200 | 10 | 37 | 5.55 | **0.97** |
| 500 × 500 | 25 | 25 | 9.44 | **8.99** |
| 1000 × 1000 | 32 | 21 | **23.58** | 32.09 |

TABLE II: Average time and iterations needed to satisfy condition in (12) on synthetic data with multi-threading.

For small matrices, multi-threading is not very effective, even slowing down computations. For $m = n = 1000$, there is a factor 4 (resp. 3) acceleration for MATLAB (resp. Julia).

Since the previous implementations of the ReLU-NMD algorithms are all available in MATLAB [4], [7], [8] and that MATLAB performs similarly than C++ and Julia, we will now stick to the multi-threaded version of MATLAB for comparing our CD algorithm with the state-of-the art methods.

[1]The number of iterations are the same for all implementations as we used the same initializations in MATLAB, Julia and C++ (namely, the SVD).

### B. Comparison with the previous methods

We now compare our CD algorithm with A-NMD and 3B-NMD from [8] which performed significantly better than the naive scheme and EM of Saul [4], [7].

*1) Synthetic low-rank data:* We first use the same synthetic data as above. Table III reports the averaged values of time and number of iterations over 5 different synthetic matrices, all algorithms use the rank-$r$ TSVD as initialization.

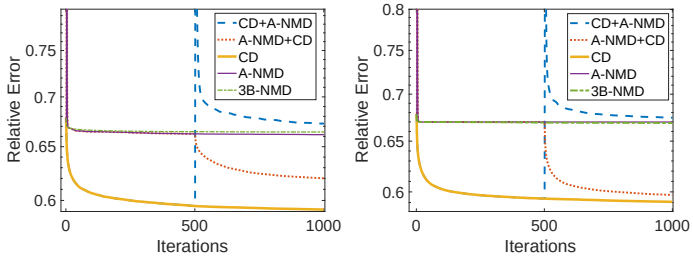
| | CD (Multi-threading) | | A-NMD | | 3B-NMD | |
|---|---|---|---|---|---|---|
| Size | time | iter | time | iter | time | iter |
| 500 × 500 | 9.15 | 25 | 0.74 | 32 | 0.07 | **23** |
| 1000 × 1000 | 22.39 | **21** | 3.45 | 27 | 0.21 | 25 |
| 1500 × 1500 | 44.08 | **20** | 2.56 | 23 | 0.38 | 25 |
| 2000 × 2000 | 71.97 | **18** | 5.20 | 26 | 0.70 | 26 |

TABLE III: Average computational time and iterations needed to satisfy condition in (12) on synthetic data with $r = 32$.

The CD algorithm converges faster than the other two algorithms in terms of the number of iterations to satisfy condition (12). However, it is significantly slower in terms of computational time as illustrated in Table III. The reason is the use of loops as well as the sorting procedure of Algorithm 1.

*2) Synthetic high-rank data:* To motivate the usefulness of the CD algorithm, let us perform another experiment on random data generated as follows: `Z = randn(m,n)` and `X = max(0,Z)`. In this way, the matrix $X$ is around $50\%$ sparse and has high rank. This problem is harder to solve since there is no exact ground-truth solution as in the previous example. In fact, it is not difficult to see that the original ReLU-NMD formulation (3) and the latent variable model (4) are equivalent when there exists an exact decomposition of rank $r$, that is, when there exists $(W, H)$ such that $X = \max(WH, 0)$.

In addition to running CD, A-NMD and 3B-NMD individually, we also performed the experiments with a hybrid combination of CD and A-NMD: we call CD+A-NMD (resp. A-NMD+CD) the algorithm that runs first CD (resp. A-NMD) with half the total iterations and then A-NMD (resp. CD) for the second half. The results are displayed on Figure 1.




(a) $m = n = 200, \text{rank} = 20$. (b) $m = n = 500, \text{rank} = 50$.

Fig. 1: Relative errors of CD, A-NMD, 3B-NMD, CD+A-NMD and A-NMD+CD on full-rank synthetic data.

Algorithms were run for 1000 iterations and initialized with the rank-$r$ TSVD. Figure 1 shows that CD reaches significantly

better solutions than A-NMD and 3B-NMD. The reason is that A-NMD and 3B-NMD rely on the reformulation (4) and do not directly solve ReLU-NMD (3). To confirm this claim, observe that when A-NMD is initialized with a solution of ReLU-NMD (3), that is, CD+A-NMD, we observe a sharp spike in the relative error and A-NMD ends up converging to a much worse solution. It is because a good solution for ReLU-NMD (3) is not a good solution for the latent formulation (4), and that $\Theta = WH$ and $Z$ with (6) leads to a sharp increase in the error. On the other hand, when CD is initialized with the solution obtained from A-NMD (A-NMD+CD), it does a good job by significantly reducing the relative error, performing better than A-NMD and 3B-NMD.

*3) Sparse Images:* Let us now compare the same algorithms on sparse image data sets as in [4], [8].

**Sparse NMF factors of the CBCL data set:** Let us illustrate another application of the CD algorithm on sparse dictionary images (around $85\%$ sparse) obtained via the rank-100 NMF decomposition of the CBCL data set as in [2]. The input matrix $X$ is a dataset of sparse images of size 361-by-100. We perform a rank-20 compression of the input matrix using A-NMD and CD. We only use A-NMD as it is performing best for this data set [8]. Note that the relative error obtained by the TSVD is large, namely $78\%$, showing that ReLU-NMD is significantly more expressive than the TSVD for sparse data sets as it will be able to achieve relative error smaller than 2.5%; see below. Figure 2 displays the relative errors. A-NMD

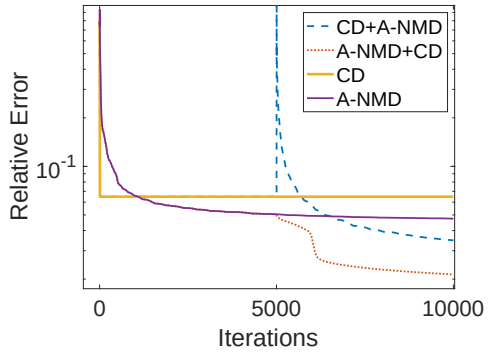
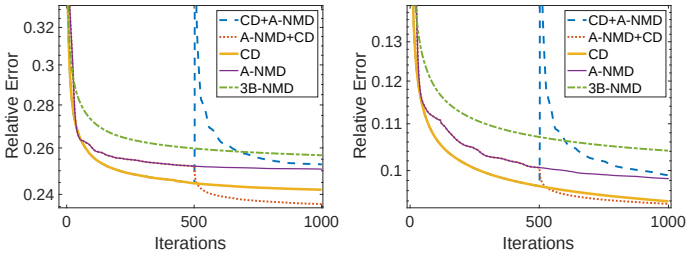



Fig. 2: Compression of a 361-by-100 sparse dictionary by A-NMD and CD initialized with the solution of A-NMD.

obtains a relative error of 4.74%. After 5000 iterations, we initialize CD with the solution obtained from A-NMD which reduces the error almost by half, to 2.41%, as before because it solves the original ReLU-NMD problem (3). Note however that CD alone is quickly stuck at a worse solution. We also initialized CD with 10 random initializations: In all 10 cases, it gets stuck at the same relative error. This is because ReLU-NMD is highly non-convex, and hence algorithms can get stuck in bad local solutions. Recall however that initializing CD with A-NMD allows CD to escape such solutions as CD+A-NMD produces the best solution. As for the other experiments, CD+A-NMD allows A-NMD to converge to a solution with lower error than A-NMD.

**MNIST dataset:** MNIST is a dataset of $28 \times 28$ greyscale images of hand-written digits [9]. The data matrix is generated by concatenating vectorized images into a matrix of size $784 \times n$ where $n = 200$ or $n = 500$. Figure 3 displays the evolution




(a) $n = 200, \text{rank} = 20$. (b) $n = 500, \text{rank} = 50$.

Fig. 3: Relative error of CD, A-NMD, and 3B-NMD on MNIST, with (a) $n = 200$, and (b) $n = 500$ images.

of the relative error when the experiment is conducted with 200 and 500 images respectively. It is evident that within the same number of iterations, CD is able to find better solutions than A-NMD and 3B-NMD. Moreover, with the hybrid combinations, that is, CD+A-NMD and A-NMD+CD, we observe a similar behaviour as in the high-rank random data case; see Figure 1 and the discussion that follows.

## V. Conclusion

In this paper, we have proposed a new algorithm based on coordinate descent (CD) to solve ReLU-NMD (3). All the previous state-of-the-art algorithms solved the latent-variable model (4) which is not equivalent to the original problem (3). Our CD algorithm provides, for the first time, a way to tackle ReLU-NMD (3) directly and is able to reach significantly better solutions than previous methods.

Further work include the acceleration of the CD algorithm, as its computational cost per iteration is at the moment larger than previous methods.

## References


[1] C. Eckart and G. Young, "The approximation of one matrix by another of lower rank," *Psychometrika*, vol. 1, no. 3, pp. 211–218, 1936.

[2] D. D. Lee and H. S. Seung, "Learning the parts of objects by non-negative matrix factorization," *Nature*, vol. 401, no. 6755, pp. 788–791, 1999.

[3] N. Whiteley, A. Gray, and P. Rubin-Delanchy, "Matrix factorisation and the interpretation of geodesic distance," *Advances in Neural Information Processing Systems*, vol. 34, pp. 24–38, 2021.

[4] L. K. Saul, "A nonlinear matrix decomposition for mining the zeros of sparse data," *SIAM J. Math. Data Sci.*, vol. 4, no. 2, pp. 431–463, 2022.

[5] L. Loconte, A. M. Sladek, S. Mengel, M. Trapp, A. Solin, N. Gillis, and A. Vergari, "Subtractive mixture models via squaring: Representation and learning," in *Int. Conf. on Learning Representations (ICLR)*, 2024.

[6] H. Liu, P. Wang, L. Huang, Q. Qu, and L. Balzano, "Symmetric matrix completion with relu sampling," in *ICML*, 2024, arXiv:2406.05822.

[7] L. K. Saul, "A geometrical connection between sparse and low-rank matrices and its application to manifold learning," *Trans. Mach. Learn. Res.*, 2023.

[8] G. Seraghiti, A. Awari, A. Vandaele, M. Porcelli, and N. Gillis, "Accelerated algorithms for nonlinear matrix decomposition with the ReLU function," in *IEEE Workshop on Mach. Learn. for Signal Process.*, 2023.

[9] L. Deng, "The MNIST database of handwritten digit images for machine learning research," *IEEE Signal Process. Mag.*, vol. 29, no. 6, pp. 141–142, 2012.