Department of Computer Science Technical Report TR-2006-21
October 2006 (revised October 2007), University of British Columbia

Computing nonnegative tensor factorizations

Michael P. Friedlander* Kathrin Hatz†

October 7, 2007

Abstract

Nonnegative tensor factorization (NTF) is a technique for computing a parts-based representation of high-dimensional data. NTF excels at exposing latent structures in datasets, and at finding good low-rank approximations to the data. We describe an approach for computing the NTF of a dataset that relies only on iterative linear-algebra techniques and that is comparable in cost to the nonnegative matrix factorization. (The better-known nonnegative matrix factorization is a special case of NTF and is also handled by our implementation.) Some important features of our implementation include mechanisms for encouraging sparse factors and for ensuring that they are equilibrated in norm. The complete MATLAB software package is available under the GPL license.

Keywords N-dimensional arrays, tensors, nonnegative tensor factorization, alternating least squares, block Gauss-Seidel, sparse solutions, regularization, nonnegative least-squares

1 Introduction

A fundamental problem in the analysis of large datasets is the identification of components that capture important features and filter less explanatory ones. A notable approach to this problem is principal component analysis (PCA) [HTF02]. In cases where the data are nonnegative (such as in images), nonnegative matrix factorization (NMF) has proven a successful approach for detecting the essential features of the data [LS99, PT94]. A generalization of the latter approach to include tensors (i.e., multidimensional arrays that may have order greater than two) can often represent the structure of the data more naturally than matrices; this approach results in a nonnegative tensor factorization (NTF) [WW01, SH05]. Several algorithms and implementations exist for the NMF case, but we are not aware of efficient implementations that have been extended to the more general tensor case. In this paper we describe an algorithm and its implementation for computing the NTF of a dataset. An important feature of our implementation is that it maintains factors that are equilibrated and have unit-norm columns. The resulting modified PARAFAC decomposition has a diagonal core tensor that captures the scale of the problem. (See, e.g., [FBH03] and [Kol06, §5.3] for a definition of the PARAFAC decomposition, and [dSL07] for a discussion on the theory of related decompositions.) Also, the implementation provides a mechanism, based on 1-norm regularization, that encourages sparse factors.

Consider the N-dimensional data tensor $\mathcal{V} \in \mathbb{R}^{I_1 \times \cdots \times I_N}$, where $I_1, \ldots, I_N$ are integers that describe the size of each of the tensor's dimensions. Of particular interest to us is

*Department of Computer Science, University of British Columbia, Vancouver V6T 1Z4, BC, Canada (mpf@cs.ubc.ca). Research supported in part by the National Science and Engineering Council of Canada.

†The work of this author was done as a visiting student at the Department of Computer Science, University of British Columbia, Vancouver V6T 1Z4, BC, Canada. Permanent address: Interdisciplinary Center for Scientific Computing of the Ruprecht-Karls-University of Heidelberg (khatz@ix.urz.uni-heidelberg.de).

the case in which each component of $\mathcal{V}$ is nonnegative. We henceforth use the shorthand notation $\mathcal{V} \geq 0$ to denote componentwise nonnegativity. The goal of NTF is to decompose $\mathcal{V}$ into a set of $N+1$ constitutive factors $\mathcal{G}$, $A^{(1)}, \ldots, A^{(N)}$ that can be combined to form an approximation of $\mathcal{V}$. A vital property of the NTF is that each factor is also nonnegative. (Each $A^{(n)}$, indexed over $n = 1, \ldots, N$, is a matrix; $\mathcal{G}$ is called the *core tensor*.) The factorization then satisfies

$$\mathcal{V} \approx \mathcal{G} \times_1 A^{(1)} \times_2 \cdots \times_N A^{(N)} \qquad \text{with} \qquad \mathcal{G}, A^{(1)}, \ldots, A^{(N)} \geq 0. \tag{1.1}$$

(The properties of the operators $\times_1, \times_2, \ldots$ are described in detail in §2.1.) When $N = 2$, $\mathcal{G}$ and $\mathcal{V}$ are simply 2-mode tensors—i.e., matrices—and the factorization above reduces to

$$V \approx A^{(1)} G A^{(2)T} \qquad \text{with} \qquad G, A^{(1)}, A^{(2)} \geq 0, \tag{1.2}$$

where $G := \mathcal{G}$ and $V := \mathcal{V}$.

The nonnegativity constraints in (1.1) and (1.2) can be motivated in several ways. Based on intuition, the parts of a dataset (represented by the factors $A^{(1)}, \ldots, A^{(N)}$) are thought to generally combine additively; the core tensor $\mathcal{G}$ encodes the relative weights of the various parts. It is also known that in many applications the quantities involved must be nonnegative; in such cases it can be difficult to interpret the results of a decomposition such as PCA that does not constrain the factors to also be nonnegative.

There are a variety of applications for nonnegative matrix and tensor factorizations, one of the most popular of which is image compression. In this application, one approach is to transform each image of a set into a vector; the set of vectors are then assembled into a matrix. NMF is then applied to this matrix. An alternative approach based on tensors has the advantage of treating the data in a more natural way: a set of two-dimensional images (each represented by a matrix) can be stacked one behind the other into a three-dimensional tensor. This construction preserves the two-dimensional character of an image and avoids a loss of information [HPS05]. Shashua and Levin [SL01] demonstrate that the compression of a tensor representation via NTF can be more efficient than the compression of a matrix representation.

1.1 The least-squares problem

A common approach to computing a decomposition of $\mathcal{V}$ is to minimize the least-squares error in the approximation shown in (1.1), i.e.,

$$\begin{array}{ll} \underset{\mathcal{G}, A^{(1)}, \ldots, A^{(N)}}{\text{minimize}} & \frac{1}{2} \left\| \mathcal{G} \times_1 A^{(1)} \times_2 \cdots \times_N A^{(N)} - \mathcal{V} \right\|_F^2 \\ \text{subject to} & \mathcal{G}, A^{(1)}, \ldots, A^{(N)} \geq 0; \end{array} \tag{1.3}$$

for examples of this approach, see [WW01, HPS05, Lin07]. Importantly, such a problem is ill-posed because the solution set of (NTF) is necessarily unbounded. To see this, note that if $\{\mathcal{G}, A^{(1)}, \ldots, A^{(n)}\}$ is any solution of (NTF), then

$$\left\{ \alpha_0 \mathcal{G}, \alpha_1 A^{(1)}, \ldots, \alpha_N A^{(N)} \right\} \qquad \text{with} \qquad \prod_{i=0}^{N} \alpha_i = 1,$$

for example, is also a solution of (1.3). This is true for any scaling of the factor $A^{(n)}$ or of the core tensor $\mathcal{G}$. It is therefore necessary to further constraint the factors in order to phrase (1.3) as a well-posed problem.

The Tucker decomposition [Kol06, §4.4] accomplishes this goal by requiring each of the factors $A^{(n)}$ to be orthonormal, so that $\|A^{(n)}\|_2 = 1$. (The Tucker decomposition, however,

does not enforce nonnegativity.) Coupled with a nonnegativity constraint, however, such a requirement may in practice be too costly to compute, or lead to an infeasible problem. We therefore strike a balance by imposing a constraint on the column norms of each $A^{(n)}$, and choosing the factors as the solution of the nonlinear least-squares problem

$$\text{(NTF)} \quad \begin{array}{ll} \underset{\mathcal{G},A^{(1)},\ldots,A^{(N)}}{\text{minimize}} & \frac{1}{2}\left\|\mathcal{G} \times_1 A^{(1)} \times_2 \cdots \times_N A^{(N)} - \mathcal{V}\right\|_F^2 \\ \text{subject to} & \mathcal{G}, A^{(1)}, \ldots, A^{(N)} \geq 0, \quad \|A^{(n)}\|_1 \leq 1, \ n = 1, \ldots, N. \end{array}$$

For the matrix case ($N = 2$), the NTF problem reduces to

$$\text{(NMF)} \quad \begin{array}{ll} \underset{G,A^{(1)},A^{(2)}}{\text{minimize}} & \frac{1}{2}\left\|A^{(1)} G A^{(2)T} - V\right\|_F^2 \\ \text{subject to} & G, A^{(1)}, A^{(2)} \geq 0, \quad \|A^{(n)}\|_1 \leq 1, \ n = 1, \ldots, N. \end{array}$$

We choose to express the 1-norm constraint as an inequality, as this preserves the convexity of the feasible set; the magnitude on the right-hand side of this constraint is arbitrary—what matters most is that it is bounded.

We note that in the literature, (NMF) typically appears with the objective

$$\tfrac{1}{2}\|A^{(1)} A^{(2)} - V\|_F^2;$$

the implication is that the core matrix $G \equiv I$. (See, e.g., [LS99, WW01, LS01, HS05, Hoy04, Lin07].) However, our formulation keeps G explicit. A second, apparently small, difference is the transpose on the second factor. As we discuss in §4.1, these two differences play a vital role in enforcing the 1-norm constraint in (NTF) and (NMF).

1.2 Approach

Two different types of algorithms are commonly used for computing nonnegative matrix and tensor factorizations. The first, and more popular, approach is based on the multiplicative update rule [LS99]. The method is simple to implement, but has been observed to converge slowly in practice. The second approach is based on alternating least squares (ALS), which is a special case of the block Gauss-Seidel method for more general nonlinear optimization problems. Applied to (NTF), the ALS approach holds all variables fixed except for one; this results in a nonnegative linear least-squares subproblem for each variable. The Gauss-Seidel approach solves each subproblem in turn and updates the variables before solving the next subproblem.

Throughout our algorithm development, we consider the most general case where (NTF) is optimized over a general tensor $\mathcal{G}$, and each of the N factors $A^{(n)}$. However, our implementation is restricted to the situation that generally arises in practice: $\mathcal{G}$ is a diagonal 3-mode tensor (implying $N = 3$). Moreover, we do not explicitly optimize over $\mathcal{G}$. Instead, we exploit the diagonal elements of $\mathcal{G}$ as a device for equilibrating the scalings of the tensor factors $A^{(1)}, \ldots, A^{(N)}$, thus enforcing the 1-norm constraints.

We choose ALS as the basis for our approach because it allows us to apply an efficient bound-constrained linear least-squares solver for each subproblem. It turns out that keeping the factors equilibrated is critical for the efficiency of our implementation, which uses a conjugate-gradient approach for solving the bound-constrained least-squares subproblems. We describe this solver in §4.2.

2 Tensor notation and basic operations

The symbol A (and any version of A modified by superscripts or superscripts) always denotes a matrix. The symbols $\mathcal{G}$ and $\mathcal{V}$ always denote n-mode tensors. Lower case roman letters

always denote vectors, and lower case Greek letters always denote scalars. The $I_n \times I_n$ identity matrix is denoted by $I_{(I_n)}$. A vector of all ones is denoted by e, and its size is implied by its context. The operator $\mathrm{vec}(\cdot)$ stacks the columns of its matrix argument into a single vector.

Let $\mathcal{G} \in \mathbb{R}^{J_1 \times \cdots \times J_N}$ be an N-dimensional tensor and let $A^{(n)}$ be a matrix of size $I_n \times J_n$, for each $n = 1, \ldots, N$. (Note that we use n as an index variable throughout the remainder of the paper.) Denote the $(i_1 i_2 \ldots i_N)$th element of the tensor $\mathcal{G}$ by $g_{i_1 i_2 \ldots i_N}$. In the case when $\mathcal{G}$ is a matrix, for example, g_{ij} denotes the (i,j)th element of $\mathcal{G}$, as usual.

A tensor $\mathcal{G}$ is diagonal if

$$(\mathcal{G})_{i_1 i_2 \ldots i_N} = \begin{cases} g_{i_1 i_2 \ldots i_N} & \text{if } i = i_1 = i_2 = \cdots = i_N \text{ with } i = 1, \ldots, \min\{J_k\}, \\ 0 & \text{otherwise;} \end{cases}$$

and is the identity tensor if each $g_{i_1 i_2 \ldots i_N}$ above is equal to one.

The Khatri-Rao product (denoted by $\odot$; see [Kol06, §3.1]) of an I-by-K matrix A and a J-by-K matrix B results in an (IJ)-by-K matrix, and is given by

$$A \odot B = [a_{:1} \otimes b_{:1} \quad a_{:2} \otimes b_{:2} \quad \cdots \quad a_{:K} \otimes b_{:K}],$$

where $\otimes$ is the Kronecker product of the columns $a_{:.}$ and $b_{:.}$ of A and B, respectively. The Kronecker product and the Khatri-Rao product of $N-1$ matrices (skipping the nth matrix $A^{(n)}$), are defined as

$$A^n_{\otimes} = A^{(N)} \otimes \cdots \otimes A^{(n+1)} \otimes A^{(n-1)} \otimes \cdots \otimes A^{(1)},$$
$$A^n_{\odot} = A^{(N)} \odot \cdots \odot A^{(n+1)} \odot A^{(n-1)} \odot \cdots \odot A^{(1)};$$

in both cases the result is a matrix. The Kronecker product of the N matrices $A^{(N)}, \ldots, A^{(1)}$ is given by

$$A_{\otimes} = A^{(N)} \otimes \cdots \otimes A^{(1)}.$$

The core of our implementation relies on two main tensor operations: the n-mode product and the n-mode matricization. We follow Kolda's description [Kol06] of these operations and briefly summarize below the needed notation. See also [dLdMV01] for background on tensors and their operations.

2.1 The n-mode product

The n-mode product defines the multiplication of a tensor with a matrix. The n-mode product of the N-mode tensor $\mathcal{G}$ with the matrix $A^{(n)}$ is denoted by $\mathcal{G} \times_n A^{(n)}$. The result is of size $J_1 \times \cdots \times J_{n-1} \times I_n \times J_{n+1} \times \cdots \times J_N$. Elementwise, the n-mode product can be expressed by

$$(\mathcal{G} \times_n A^{(n)})_{j_1 \ldots j_{n-1} i j_{n+1} \ldots j_N} = \sum_{j_n=1}^{J_n} g_{j_1 j_2 \ldots j_N} a_{i j_n}.$$

The number of columns in $A^{(n)}$ must be equal to the size of nth mode of $\mathcal{G}$.

2.2 The n-mode matricization

We often make use of transformations that change tensors into matrices (n-mode matricization), and matrices into vectors (vectorization). The n-mode matricization transforms the tensor $\mathcal{G}$ into a matrix $G_{(n)}$ defined by

$$G_{(n)} \in \mathbb{R}^{J_n \times K} \qquad \text{with} \qquad K = \prod_{\substack{i=1 \\ i \neq n}}^{N} J_i,$$

for each $n = 1, \ldots, N$. Note that the number of rows of $G_{(n)}$ is equal to the size of the nth dimension of the tensor; the number of columns is expanded to accommodate all the other dimensions of the tensor. The matrix $G_{(n)}$ can be expressed elementwise as

$$(G_{(n)})_{i_n k} = g_{i_1 i_2 \ldots i_N} \qquad \text{with} \qquad k = 1 + \sum_{\substack{m=1 \\ m \neq n}}^{N} \left[(i_m - 1) \prod_{\substack{m'=1 \\ m' \neq n}}^{m} I_{m'} \right].$$

2.3 Derivatives

In order to verify optimality of a solution estimate of (NTF), we need an expression for the derivatives of the objective

$$\phi(\mathcal{G}, A^{(1)}, \ldots, A^{(N)}) := \tfrac{1}{2} \left\| \mathcal{G} \times_1 A^{(1)} \times_2 \cdots \times_N A^{(N)} - \mathcal{V} \right\|_F^2$$

with respect to $\mathcal{G}$ and each $A^{(n)}$. Let

$$\mathcal{R} := \mathcal{G} \times_1 A^{(1)} \times_2 \cdots \times_N A^{(N)} - \mathcal{V} \tag{2.1}$$

be the residual. The derivatives of $\mathcal{R}$ with respect to $\mathcal{G}$ and $A^{(n)}$ are derived in [Kol06, §4.5] and are given by

$$\left(\frac{\partial \mathcal{R}}{\partial \mathcal{G}} \right)_{i_1 \cdots i_N j_1 \cdots j_N} = a^{(1)}_{i_1 j_1} a^{(2)}_{i_2 j_2} \cdots a^{(N)}_{i_N j_N} \qquad \text{and} \qquad \left(\frac{\partial \mathcal{R}}{\partial A^{(n)}} \right) = A^n_{\otimes} G^T_{(n)} \otimes I_{(I_N)},$$

where $a^{(k)}_{ij}$ is the (ij)th element of $A^{(k)}$. The first derivative has dimensions $\prod_{i=1}^N I_i \times \prod_{i=1}^N J_i$ and the second derivative has dimensions $\prod_{i=1}^N I_i \times J_n I_n$. (With the appropriate matricization, the derivative $\partial \mathcal{R} / \partial \mathcal{G}$ can be expressed simply as $A_{\otimes}$.) The Frobenious norm is quadratic, and so $\partial \phi(\cdot) / \partial \mathcal{R} = \mathcal{R}$. We then apply the chain rule, and use [Kol06, Proposition 4.3(c)] to obtain

$$\frac{\partial \phi(\cdot)}{\partial \mathcal{G}} = \mathcal{R} \times_1 A^{(1)T} \times_2 \cdots \times_N A^{(N)T}, \qquad \text{and} \qquad \frac{\partial \phi(\cdot)}{\partial A^{(n)}} = R_{(n)} A^n_{\otimes} G^T_{(n)}.$$

The reduced gradients (components corresponding to variables that are positive) must be small at an approximation solution.

3 The NTF algorithm

Given a nonnegative tensor $\mathcal{V} \in \mathbb{R}^{I_1 \times \cdots \times I_N}$, NTF computes an approximate factorization

$$\mathcal{V} \approx \mathcal{G} \times_1 A^{(1)} \times_2 \cdots \times_N A^{(N)}$$

into the N nonnegative matrix factors $A^{(n)} \in \mathbb{R}^{I_n \times J_n}$, $n = 1, \ldots, N$, and the nonnegative tensor $\mathcal{G} \in \mathbb{R}^{J_1 \times \cdots \times J_N}$. These factors are chosen to solve the constrained nonlinear least-squares problem (NTF). The ALS approach transforms (NTF) into a sequence of $N + 1$ subproblems. In order to formulate each linear least-squares subproblem, we need to develop two transformations that allow us to isolate each factor $A^{(n)}$ and the core tensor $\mathcal{G}$.

The matrix $A^n_{\otimes} G^T_{(n)}$ arises at several key points in our implementation. This overall product

$$A^n_{\otimes} G^T_{(n)} \quad \text{has dimensions} \quad \prod_{\substack{i=1 \\ i \neq n}}^{N} I_i \times J_n \tag{3.1}$$

(where typically $J_n \ll I_i$), whereas the intermediate matrix

$$A^n_{\otimes} \quad \text{has dimensions} \quad \prod_{\substack{i=1\\ i\neq n}}^{N} I_i \times \prod_{\substack{i=1\\ i\neq n}}^{N} J_i,$$

which can be much larger than (3.1) and prohibitively large to store. If we make the assumption that $\mathcal{G}$ is diagonal, then we can use the Khatri-Rao product [Kol06, §3.1] to derive an equivalent expression that does not involve the large matrix $A^n_{\otimes}$. In particular, if $\mathcal{G}$ is diagonal, then

$$A^n_{\otimes} G^T_{(n)} = A^n_{\odot} D^T, \tag{3.2}$$

where the diagonal (and possibly rectangular) matrix D has elements $(D)_{ii} = (\mathcal{G})_{i\ldots i}$, $i = 1, \ldots, \min_k\{J_k\}$. Importantly, the intermediate matrix

$$A^n_{\odot} \quad \text{has dimensions} \quad \prod_{\substack{i=1\\ i\neq n}}^{N} I_i \times J_n,$$

which is the same as the overall matrix shown in (3.1). Our general algorithmic development does not assume that $\mathcal{G}$ is diagonal, and so we continue to use $A^n_{\otimes} G^T_{(n)}$; we make the switch to $A^n_{\odot} D^T$ in §4, where we assume that $\mathcal{G}$ is diagonal.

3.1 The linear least-squares subproblems

At each iteration of the ALS method, we need to isolate one of the factors in order to derive a linear least-squares subproblem. We use two properties of n-mode matricization for this purpose. It follows from [Kol06, Proposition 3.7] that if

$$\mathcal{X} = \mathcal{G} \times_1 A^{(1)} \times_2 \cdots \times_N A^{(N)},$$

then for each $n = 1, \ldots, N$,

$$X_{(n)} = A^{(n)} G_{(n)} (A^n_{\otimes})^T \tag{3.3}$$

and

$$\operatorname{vec} X_{(n)} = A_{\otimes} \operatorname{vec} G_{(n)}. \tag{3.4}$$

In order to isolate any given factor $A^{(n)}$, $n = 1, \ldots, N$, we use (3.3) to rewrite the objective of (NTF) as

$$\begin{aligned} \phi(\mathcal{G}, A^{(1)}, \ldots, A^{(N)}) &\equiv \tfrac{1}{2} \left\| \mathcal{G} \times_1 A^{(1)} \times_2 \cdots \times_N A^{(N)} - \mathcal{V} \right\|_F^2 \\ &= \tfrac{1}{2} \left\| A^{(n)} G_{(n)} (A^n_{\otimes})^T - V_{(n)} \right\|_F^2 \\ &= \tfrac{1}{2} \left\| A^n_{\otimes} G^T_{(n)} A^{(n)T} - V^T_{(n)} \right\|_F^2. \end{aligned} \tag{3.5}$$

From (3.5) we see that the optimization of ϕ over $A^{(n)}$ reduces to I_n independent linear least-squares problems: each involves the same matrix $A^n_{\otimes} G^T_{(n)}$, but I_n different right-hand-side vectors that constitute the rows of $V_{(n)}$; the solution of each of these least-squares problems corresponds to a row of $A^{(n)}$. In principal, each of the rows of $A^{(n)}$ can be solved for in parallel.

In our implementation, we use $A^n_{\otimes} G^T_{(n)}$ as an operator and solve for all of $A^{(n)}$ simultaneously. We do this by vectorizing the term within the norm of (3.5) and rewriting ϕ as

$$\begin{aligned}\phi(\mathcal{G}, A^{(1)}, \ldots, A^{(N)}) &= \tfrac{1}{2} \| \operatorname{diag}\left(A^n_{\otimes} G^T_{(n)}, \ldots, A^n_{\otimes} G^T_{(n)}\right) \operatorname{vec} A^{(n)T} - \operatorname{vec} V^T_{(n)} \|_2^2 \\ &= \tfrac{1}{2} \| \left(I_{(I_n)} \otimes (A^n_{\otimes} G^T_{(n)})\right) \operatorname{vec} A^{(n)T} - \operatorname{vec} V^T_{(n)} \|_2^2\end{aligned}$$

The resulting N subproblems (one for each $A^{(n)}$) are constrained linear least-squares problems over the vector $\operatorname{vec} A^{(n)T}$:

$$\begin{array}{ll} (\mathrm{NTF}_{A^{(n)}}) & \underset{A^{(n)}}{\text{minimize}} \quad \| \left(I_{(I_n)} \otimes (A^n_{\otimes} G^T_{(n)})\right) \operatorname{vec} A^{(n)T} - \operatorname{vec} V^T_{(n)} \|_2 \\ & \text{subject to} \quad A^{(n)} \geq 0, \quad \|A^{(n)}\|_1 \leq 1. \end{array}$$

Note that $A^n_{\otimes}$ is the contracted Kronecker product and does not include $A^{(n)}$, which is the optimization variable. Also, each $I_{(I_n)} \otimes (A^n_{\otimes} G^T_{(n)})$ in $(\mathrm{NTF}_{A^{(n)}})$ is a block diagonal matrix, with the submatrix $A^n_{\otimes} G^T_{(n)}$ appearing at each block entry. Because $A^n_{\otimes} G^T_{(n)}$ is only used as an operator, it is not necessary to form this matrix explicitly.

To isolate $\mathcal{G}$ for the $(N+1)$st subproblem, we use (3.4) to transform the objective as

$$\phi(\mathcal{G}, A^{(1)}, \ldots, A^{(N)}) = \tfrac{1}{2} \left\| A_{\otimes} \operatorname{vec} G_{(1)} - \operatorname{vec} V_{(1)} \right\|_2^2 . \tag{3.6}$$

The optimization over $\mathcal{G}$ is then reduced to the nonnegative linear least-squares problem over the vectorized 1-mode matricization of the tensor $\mathcal{G}$:

$$\begin{array}{ll} (\mathrm{NTF}_{\mathcal{G}}) & \underset{\mathcal{G}}{\text{minimize}} \quad \| A_{\otimes} \operatorname{vec} G_{(1)} - \operatorname{vec} V_{(1)} \|_2 \\ & \text{subject to} \quad \mathcal{G} \geq 0. \end{array}$$

The 1-norm constraint $\|A^{(n)}\|_1 \leq 1$ imposes a constraint on each of the columnwise sum of absolute values of $A^{(n)}$. Importantly, the componentwise nonnegativity constraint on $A^{(n)}$ can be leveraged to express the nondifferentiable 1-norm as a linear function: for any vector x,

$$\|x\|_1 = e^T x \qquad \text{if} \qquad x \geq 0. \tag{3.7}$$

Hence, the 1-norm constraint $\|A^{(n)}\|_1 \leq 1$ can be imposed by a collection of simple linear inequality constraints. We describe in §4.1 an approach that imposes these constraints implicitly.

3.2 Alternating least squares and block Gauss-Seidel

Algorithm 1 describes the basic ALS method for (NTF) that is based on solving a sequence of subproblems defined by $(\mathrm{NTF}_{A^{(n)}})$ and $(\mathrm{NTF}_{\mathcal{G}})$. The ALS approach is a special case of the block coordinate-descent method, also knows as the nonlinear block Gauss-Seidel (BGS) method. The nonlinear BGS method is originally an iterative method for solving systems of linear equations, and has been generalized to the minimization of nonlinear functions (see, e.g., [Kel95, Chapter 1], and [Ber99, §2.7]). At each iteration of the block Gauss-Seidel method, a subset of the variables are held fixed while the problem is minimized over the remaining variables. In the NMF and NTF cases, this partitioning leads to constrained linear least-squares subproblems.

These subproblem solution generate a sequence $\{A^{(1)}_k, A^{(2)}_k, \ldots, A^{(N)}_k, \mathcal{G}_k\}$. If the solution $A^{(n)}_k$ and $\mathcal{G}_k$ of each subproblem $(\mathrm{NTF}_{A^{(n)}})$ and $(\mathrm{NTF}_{\mathcal{G}})$ is uniquely attained, then every limit point of the sequence $\{A^{(1)}_k, A^{(2)}_k, \ldots, A^{(N)}_k, \mathcal{G}_k\}$ is a stationary point for (NTF). (See,

Algorithm 1: The alternating least-squares algorithm for (NTF)

Input: $\mathcal{V} \in \mathbb{R}^{I_1 \times \cdots \times I_N}$
Output: $\mathcal{G}_* \in \mathbb{R}^{J_1 \times \cdots \times J_N}$, $A_*^{(n)} \in \mathbb{R}^{I_n \times J_n}$ for $n = 1, \ldots, N$
Initialize $A_0^{(1)}, \ldots, A_0^{(n)} \geq 0$, $\mathcal{G}_0 \geq 0$
$k \leftarrow 0$
repeat
  for $n = 1, \ldots, N$ **do**
    $A_{k+1}^{(n)} \leftarrow$ solve $(\text{NTF}_{A^{(n)}})$
  $\mathcal{G}_{k+1} \leftarrow$ solve $(\text{NTF}_{\mathcal{G}})$
  $k \leftarrow k + 1$
until *converged*
$\mathcal{G}_* \leftarrow \mathcal{G}_k$, and $A_*^{(n)} \leftarrow A_k^{(n)}$ for $n = 1, \ldots, N$

e.g., [Ber99, §2.7] for a detailed discussion of BGS methods.) The convergence rate of BGS is linear [LT92]. No convexity assumption on the objective is needed, though the Cartesian product of the subproblem constraints must be convex, as it is in the NTF case.

The "uniquely attained" assumption is satisfied if the least-squares problems have full rank. One possible way to safeguard against a lack of uniqueness is to augment the subproblems with a convex regularization term that makes the subproblem solutions unique. Our implementation relies on an additional 1-norm regularization function—which although is not strictly convex and thus does not guarantee uniqueness of the solution—has the benefit that it can help encourage sparse solutions to the overall problem. We discuss this further in §4.

3.3 Regularizing for sparseness

An important feature of NTF is that it can decompose data into parts that have intuitive meaning and can easily be interpreted in terms of the original data. Hoyer [Hoy04] has observed that NMF tends to produce parts that are sparse—that is, the factors tend to have many small or zero entries. With the idea that a parsimonious representation of the data yields parts that are well defined, we introduce a mechanism for encouraging the sparsity of the factors obtained via NTF.

Our approach is based on regularizing (NTF) with an 1-norm penalty function. This nondifferentiable function has a well-observed property of pushing small values exactly to zero while leaving large (and significant) entries relatively undisturbed. Similar applications of 1-norm regularization include signal processing for the recovery of sparse signals; see, e.g., [CDS01, CRT06, CRT05, DT05].

We define the regularized NTF problem as

$$(\text{NTF}_{\text{sp}}) \qquad \underset{\mathcal{G}, A^{(1)}, \ldots, A^{(N)}}{\text{minimize}} \ \phi(\mathcal{G}, A^{(1)}, \ldots, A^{(N)}) + \gamma \psi(A^{(n)}, \mathcal{G}) \quad \text{subject to} \ \ \mathcal{G}, A^{(1)}, \ldots, A^{(N)} \geq 0,$$

where γ is a nonnegative regularization parameter and

$$\psi(\mathcal{G}, A^{(n)}) := \| \operatorname{vec} G_{(1)} \|_1 + \sum_{n=1}^{N} \| \operatorname{vec} A^{(n)} \|_1$$

is the 1-norm regularization function. The alternating least-squares approach derived in §3.1 extends easily to the regularized problem (NTF_{sp}). With (3.7), we can write the linear

least-squares subproblems ($\text{NTF}_{A^{(n)}}$) and ($\text{NTF}_{\mathcal{G}}$), respectively, as

$$\begin{array}{ll} \underset{A^{(n)}}{\text{minimize}} & \frac{1}{2}\|\left(I_{(I_n)} \otimes (A_{\otimes}^{n} G_{(n)}^{T})\right) \operatorname{vec} A^{(n)T} - \operatorname{vec} V_{(n)}^{T}\|_2^2 + \gamma e^T \operatorname{vec} A^{(n)T} \\ \text{subject to} & A^{(n)} \geq 0, \quad \|A^{(n)}\|_1 \leq 1, \end{array} \tag{3.8a}$$

and

$$\begin{array}{ll} \underset{\mathcal{G}}{\text{minimize}} & \frac{1}{2}\|A_{\otimes} \operatorname{vec} G_{(1)} - \operatorname{vec} V_{(1)}\|_2^2 + \gamma e^T \operatorname{vec} G_{(1)} \\ \text{subject to} & \mathcal{G} \geq 0. \end{array} \tag{3.8b}$$

Each subproblem now has an additional linear term. Note that we discard the constant linear terms that correspond to the other fixed variables.

There is an important distinction between the explicit matrix 1-norm constraint on $A^{(n)}$ and the vector 1-norm regularization on $\operatorname{vec} A^{(n)T}$, although the two are clearly related. The explicit constraint imposes a bound on the maximum column sum; the regularization term imposes a bound on the overall sum of the matrix entries.

3.4 Specialization to matrices (NMF)

For interest, and to draw a relationship to the existing NMF literature, we describe the linear least-squares subproblems that arise in the matrix case, corresponding to $N = 2$. The subproblems analogous to ($\text{NTF}_{A^{(n)}}$) and ($\text{NTF}_{\mathcal{G}}$) are

$$\begin{array}{lllll} (\text{NMF}_{A^{(1)}}) & \underset{A^{(1)}\geq 0}{\text{minimize}} & \|(I \otimes A^{(2)}G) \operatorname{vec} A^{(1)T} - \operatorname{vec} V^T\|_2 & \text{subject to} & \|A^{(1)}\|_1 \leq 1, \\ (\text{NMF}_{A^{(2)}}) & \underset{A^{(2)}\geq 0}{\text{minimize}} & \|(I \otimes A^{(1)}G) \operatorname{vec} A^{(2)T} - \operatorname{vec} V\|_2 & \text{subject to} & \|A^{(2)}\|_1 \leq 1, \\ (\text{NMF}_{G}) & \underset{G\geq 0}{\text{minimize}} & \|(A^{(2)} \otimes A^{(1)}) \operatorname{vec} G - \operatorname{vec} V\|_2. & & \end{array}$$

As discussed in the context of ($\text{NTF}_{A^{(n)}}$), the analogous problems (NMF_{W}) and (NMF_{G}) can each be solved as a set of independent linear least-squares problems with a different right-hand-side vector that corresponds to a row (or column) of V. Also, the 1-norm-regularization approach discussed in §3.3 extends immediately to the matrix case with the addition of a linear term to each of the subproblems.

4 Implementation

Our algorithm development up to this point has allowed for a general core tensor $\mathcal{G}$ and an arbitrary number of factors N. For our implementation of the NTF algorithm, we make the simplifying assumption that $N = 3$, which is the situation that arises most often in practice. However, we note that our implementation can be generalized to any N with only trivial modifications, although the subproblems will grow as more matrix factors are added (cf. (3.1)).

We discuss below a method for enforcing the 1-norm constraint $\|A^{(n)}\|_1 \leq 1$. It turns out that maintaining factors that are equilibrated is vital for the efficiency of our approach. We also briefly describe the algorithm used for solving the resulting bound-constrained linear-least-squares subproblems.

4.1 Scaling

In the implementation of our NTF method, we maintain $\mathcal{G}$ diagonal and remove the subproblem ($\text{NTF}_{\mathcal{G}}$) from the loop in Algorithm 1. Having removed $\mathcal{G}$ as one of the optimization variables, we may then use it as a degree of freedom to enforce the 1-norm constraint by ensuring that $\|A^{(n)}\|_1 = 1$. It turns out that keeping the factors equilibrated is a useful

device for maintaining iterates with a scale most favourable for an efficient solution of the underlying linear least-squares subproblems (cf. §3.1), as we describe below. A further advantage of maintaining factors that have unit norm is that it ensures that the iterates stay away from zero, which is necessarily a nonoptimal stationary point.

The role of $\mathcal{G}$ as a scaling device is most easily understood in the matrix case where $\mathcal{G}$ is a diagonal matrix (and not necessarily the identity). Let $G := D^{(1)}D^{(2)}$ be a product of diagonal matrices $D^{(1)}$ and $D^{(2)}$, and rewrite (1.2) as

$$A^{(1)}GA^{(2)T} = A^{(1)}D^{(1)}D^{(2)}A^{(2)T} = (A^{(1)}D^{(1)})(A^{(2)}D^{(2)})^T = \bar{A}^{(1)}\bar{A}^{(2)T}, \tag{4.1}$$

where

$$\bar{A}^{(1)} := A^{(1)}D^{(1)} \quad \text{and} \quad \bar{A}^{(2)} := A^{(2)}D^{(2)}.$$

The diagonal operators $D^{(1)}$ and $D^{(2)}$ simply rescale the columns of $A^{(1)}$ and $A^{(2)}$. Given factors $\bar{A}^{(1)}$ and $\bar{A}^{(2)}$, the freedom to choose G in (4.1) implies that we can choose any scale we like for the columns of $A^{(1)}$ and $A^{(2)}$, and instead absorb the scales of $\bar{A}^{(1)}$ and $\bar{A}^{(2)}$ into the core matrix G. In some sense, this is analogous to the singular value decomposition, which has orthogonal factors and a "core" matrix (the singular values) that express the scale of the matrix. The transpose on $A^{(2)}$ is an important ingredient in making the transformation in (4.1) possible.

In the tensor case, the core tensor $\mathcal{G}$ can be similarly used to ensure that the factors have a chosen scale. For each $n = 1, \ldots, N$, we rescale the columns of the factors $A^{(n)}$ with diagonal matrices $D^{(n)}$ as follows: define the diagonal tensor

$$\mathcal{G} := \mathcal{I} \times_1 D^{(1)} \times_2 \cdots \times_N D^{(N)},$$

(with $\mathcal{I}$ as the identity tensor), so that

$$\begin{aligned} &\mathcal{G} \times_1 A^{(1)} \times_2 \cdots \times_N A^{(N)} \\ &\quad = \left[\mathcal{I} \times_1 D^{(1)} \times_2 \cdots \times_N D^{(N)}\right] \times_1 \left[A^{(1)} \times_2 \cdots \times_N A^{(N)}\right] \\ &\quad = \mathcal{I} \times_1 \left[A^{(1)}D^{(1)} \times_2 \cdots \times_N A^{(N)}D^{(N)}\right] \\ &\quad = \mathcal{I} \times_1 \bar{A}^{(1)} \times_2 \cdots \times_N \bar{A}^{(N)}, \end{aligned} \tag{4.2}$$

where each $\bar{A}^{(n)} := A^{(n)}D^{(n)}$, and the second equality follows from [Kol06, Proposition 4.2(a)]. We define scaling matrices $D^{(n)}$ that ensure each $A^{(n)}$ is well scaled. Thus $\mathcal{G} \times_1 A^{(1)} \times_2 \cdots \times_N A^{(N)}$ is in fact a PARAFAC decomposition [FBH03] with scaled matrix factors; when $N = 2$ (and $\mathcal{G}$ is a matrix, (4.2) and (4.1) are equivalent expressions.

When $\mathcal{G}$ is diagonal, (3.2) can be used to express $(\text{NTF}_{A^{(n)}})$ as

$$\begin{array}{ll} \underset{A^{(n)}}{\text{minimize}} & \|\left(I_{(I_n)} \otimes (A^n_{\odot}D^T)\right) \operatorname{vec} A^{(n)T} - \operatorname{vec} V^T_{(n)}\|_2 \\ \text{subject to} & A^{(n)} \geq 0, \end{array} \tag{4.3}$$

where D is the product of the diagonal matrices $D^{(n)}$ in (4.2). After this linear least-squares subproblem is solved at iteration k, the solution $A^{(n)}_k$ is rescaled by

$$D^{(n)} = \operatorname{diag}\left(e^T A^{(n)}\right)^{-1}, \tag{4.4}$$

so that the resulting columns of the scaled matrix $A^{(n)}D^{(n)}$ each sum to one. Thus, $\left\|A^{(n)}D^{(n)}\right\|_1 = 1$. In effect, this iteration can be viewed as solving the subproblem for $\bar{A}^{(n)}$ and then choosing $A^{(n)}$ and $D^{(n)}$ so that $\bar{A}^{(n)} \equiv A^{(n)}D^{(n)}$. Importantly, this redefinition of the latest iterate continues to be feasible and does not cause the objective

Algorithm 2: The alternating least-squares algorithm (with scaling) for (NTF)

Input: $\mathcal{V} \in \mathbb{R}^{I_1 \times \cdots \times I_N}$
Output: $\mathcal{G}_* \in \mathbb{R}^{J_1 \times \cdots \times J_N}$, $A_*^{(n)} \in \mathbb{R}^{I_n \times J_n}$ for $n = 1, \ldots, N$
Initialize $A_0^{(1)}, \ldots, A_0^{(n)} \geq 0$, $D_0^{(n)} = I$ for $n = 1, \ldots, N$
$k \leftarrow 0$
repeat
  for $n = 1, 2, \ldots, N$ **do**
    $A^{(n)} \leftarrow$ solve (4.3)
    Compute $D_k^{(n)}$ from $A^{(n)}$ [compute column scales; see (4.4)]
    $A_{k+1}^{(n)} \leftarrow A^{(n)} D_k^{(n)}$ [compute scaled factor]
    $D_{k+1} \leftarrow D_k (D_k^{(n)})^{-1}$ [update diagonal core tensor]
  $k \leftarrow k + 1$
until *converged*
$(\mathcal{G}_*)_{i \cdots i} \leftarrow (D_k)_{ii}$ for $i = 1, \ldots, N$, and $A_*^{(n)} \leftarrow A_k^{(n)}$ for $n = 1, \ldots, N$

function to increase. Therefore, Proposition 2.7.1 of [Ber99] holds, and the rescaling does not interfere with the convergence of the Gauss-Seidel iterations. Algorithm 2 describes our implementation of this scaling strategy.

We can interpret this rescaling as a projection of each of the Gauss-Seidel iterates onto the convex set of nonnegative matrices with induced matrix 1-norm. (Note that if the entire matrix was scaled so that vec $A^{(n)}$ would have unit 2-norm, then we could not interpret the rescaling as a projection onto a convex set.)

4.2 Solving the least-squares subproblems

The computational kernel of the alternating least-squares algorithm is the solution of the nonnegative linear least-squares problems ($\text{NTF}_{A^{(n)}}$) and ($\text{NTF}_{\mathcal{G}}$) described in §3.1. The efficiency of the overall method ultimately depends on the efficient solution of the large-scale subproblems that can arise in this context. In the context of the large datasets that can arise in applications of NTF, we must be prepared to apply optimization methods that do not rely on matrix factorizations, which can be prohibitively expensive. Our approach is based on an implementation that uses matrices only as operators.

We give here a brief description of the software package BCLS [Fri06] used to solve the nonnegative least-squares subproblems. BCLS is a separate implementation for solving least-squares problems with bound constraints. We describe the BCLS algorithm in context of the generic problem

$$\begin{array}{ll} \underset{x \in \mathbb{R}^n}{\text{minimize}} & \frac{1}{2}\|Ax - b\|_2^2 + c^T x + \frac{1}{2}\gamma^2 \|x\|_2^2 \\ \text{subject to} & \ell \leq x \leq u, \end{array} \tag{4.5}$$

where A is an $m \times n$ matrix and b and c are m- and n-vectors. The n-vectors ℓ and u are lower and upper bounds on the variables x; γ is a nonnegative regularization parameter that can be used to control the norm of the final solution. A value of $\gamma = 0$ is permitted in the implementation and simply eliminates the regularization term.

The BCLS algorithm is based on a two-metric projection method (see, e.g., [Ber82, Chapter 2]). A partitioning of the variables is maintained at all times; variables that are well within the interior of the feasible set are labeled *free*, and variables that are at (or near) one of their bounds are labeled *fixed*. Conceptually, the variables x and the data A and c

are correspondingly partitioned into their free (B) and fixed (N) components:

$$x = \begin{bmatrix} x_B & x_N \end{bmatrix}, \quad c = \begin{bmatrix} c_B & c_N \end{bmatrix}, \quad \text{and} \quad A = \begin{bmatrix} A_B & A_N \end{bmatrix}. \tag{4.6}$$

At each iteration, the two-metric projection method generates independent descent directions Δx_B and Δx_N for the free and fixed components of x; these are generated from an approximate solution of the block-diagonal linear system

$$\begin{bmatrix} A_B^T A_B + \gamma^2 I & 0 \\ 0 & D \end{bmatrix} \begin{bmatrix} \Delta x_B \\ \Delta x_N \end{bmatrix} = A^T r - c - \gamma^2 x, \tag{4.7}$$

where $r = b - Ax$ is the current residual, and D is a diagonal matrix with strictly positive entries. The right-hand side of the above equation is the negative of the gradient of (4.5). Thus a Newton step is generated for the free variables x_B, and a scaled steepest-descent step is generated for the fixed variables x_N. The aggregate step $(\Delta x_B, \Delta x_N)$ is then projected into the feasible region and the first minimizer is computed along the piecewise linear projected-search direction (see, e.g., [CGT00] for a detailed description on projected search methods).

The linear system (4.7) is never formed explicitly. Instead, Δx_B is computed equivalently as a solution of the least-squares problem

$$\underset{\Delta x_B}{\text{minimize}} \quad \tfrac{1}{2}\|A_B \Delta x_B - r\|_2^2 + c_B^T \Delta x_B + \tfrac{1}{2}\gamma^2 \|x_B + \Delta x_B\|_2^2. \tag{4.8}$$

We find an approximate solution to (4.8) by applying the conjugate-gradient-type solver LSQR [PS82] to the problem

$$\underset{\Delta x_B}{\text{minimize}} \quad \left\| \begin{bmatrix} A_B \\ \beta I \end{bmatrix} \Delta x_B - \begin{bmatrix} r \\ \frac{1}{\beta} c_B - \frac{\gamma^2}{\beta} x_B \end{bmatrix} \right\|, \tag{4.9}$$

where $\beta = \max\{\gamma, \bar{\gamma}\}$ and $\bar{\gamma}$ is a small positive constant. (In principle, this least-squares problem could alternatively be solved via a *direct method*, e.g., the QR factorization; but BCLS is aimed at large-scale problems where such an approach may be infeasible.) If $\gamma < \bar{\gamma}$ (as it is with the ALS subproblems of §3.1), then the resulting step is effectively a modified Newton step. Although this can lead to slower convergence, it has the side effect of safeguarding against rank-deficient systems.

The scaling strategy described in §4.1 implies that the solution Δx_B will be well scaled. This is an especially favorable circumstance for CG-type solvers.

5 Numerical experiments

We implemented Algorithm 2 in MATLAB. Two separate interfaces are available. The first interface (lsNTF) implements the nonnegative tensor factorization for $N = 3$ and relies on the MATLAB Tensor Toolbox [BK06b, BK06a]. The second interface (lsNMF) implements the nonnegative matrix factorization (e.g., $N = 2$) and does not rely on the Tensor Toolbox.

We illustrate the performance of lsNTF and lsNMF on a set of images from the CBCL Face Database [BL06]. For the tensor case, we assemble 1000 gray-scale images, each 19×19 pixels, into a tensor $\mathcal{V}$ with dimensions $19 \times 19 \times 1000$. For the matrix case, we assemble the same images into a matrix V with dimensions $19^2 \times 1000$. For both cases we choose a fixed inner dimension of 15, which corresponds to $J_1 = J_2 = J_3 = 15$ in the tensor case and $J_1 = J_2 = 15$ in the matrix case. (Our choice of the inner dimension is arbitrary and empirical. Indeed, how to choose this dimension is an open problem, and out of the scope of this paper. For relevant background, see [LMV00, dSL07].)

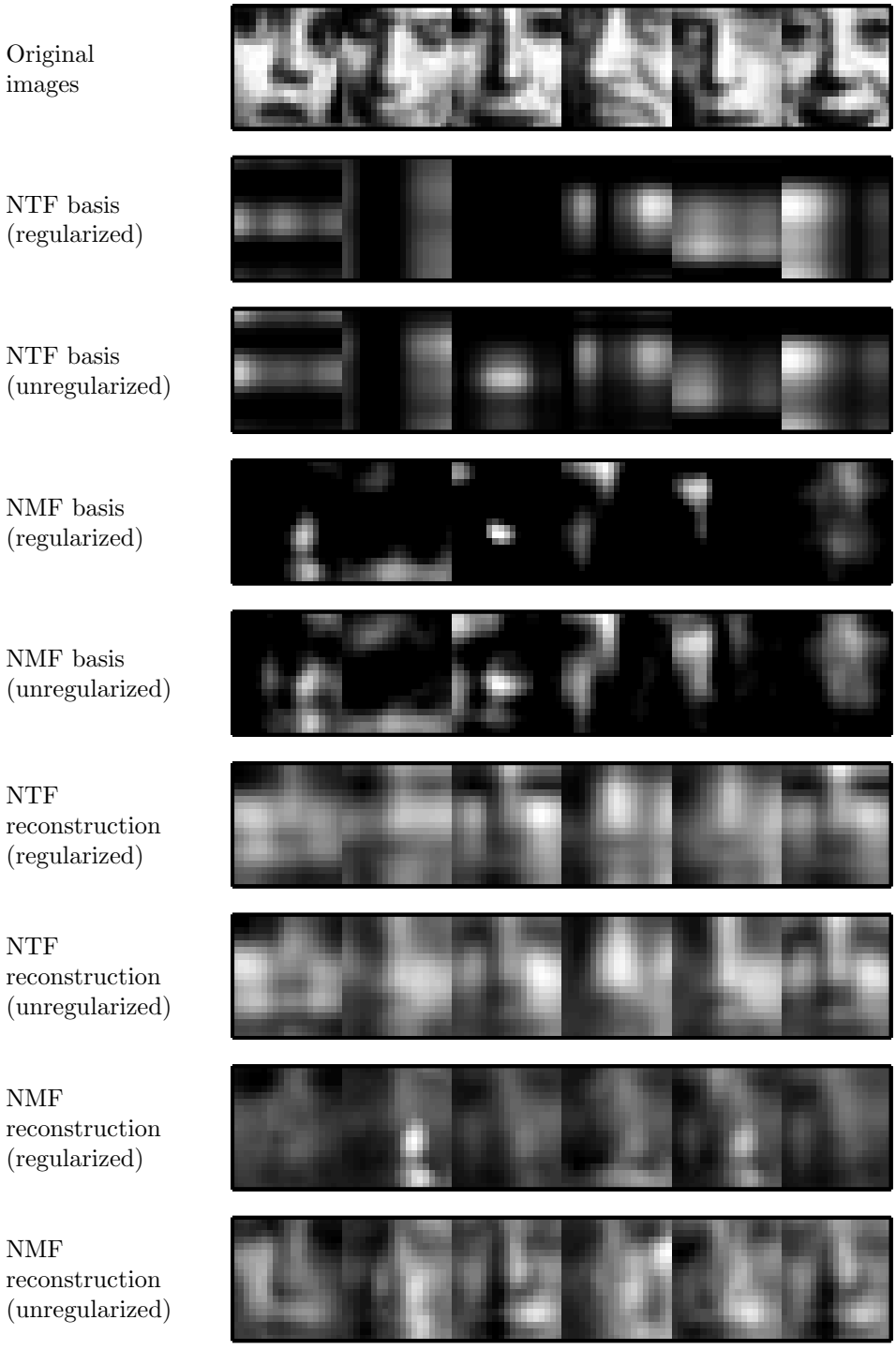

FIGURE 1: *The first row shows six of the original images in the CBCL test set. The next four rows show the bases computed from 1000 images in the dataset using the NMF and the NTF with and without 1-norm regularization. The last four rows display reconstructions of the original faces using the tensor and matrix factorizations with and without 1-norm regularization.*

Algorithm	Iterations	Optimality	Total time (sec)
lsNTF	6	1.9e−04	315
lsNMF	3	5.9e−05	357
projGradNMF	47	9.9e−01	260
multUpdateNMF	(1000)	(9.0e+00)	(320)

TABLE 1: *Performance of four algorithms on the nonnegative factorization of 1000 images with 15 leading factors*

Table 1 shows the relative performance of our matrix and tensor factorization implementations (lsNMF and lsNTF, respectively). For interest, we also show the results of solving the same NMF problem with a projected gradient method [Lin07] and multiplicative update method [LS01] (these are the rows labeled projGradNMF and multUpdateNMF). The multiplicative update method failed to converge within its allotted maximum of 1000 iterations. The numbers of iterations are not comparable across algorithms, but are shown only for interest. Most relevant are the last two columns, which show the optimality achieved (the norm of the reduced gradient) and the solution time to achieve that level of optimality, though we caution that CPU times are generally a poor gauge of an an algorithm's efficiency. Importantly, however, the timings in the table indicate that we are able to solve the NTF and NMF problems with very similar computing times. All runs were conducted on a 3.2 GHz Intel Pentium 4 running Linux 2.6.16 and MATLAB 7.2.

The images in Figure 1 illustrate the results of the NTF and NMF factorizations. The first row shows six typical images in the CBCL test set. The next four rows show six of the fifteen basis images computed using NTF and NMF, both with and without 1-norm regularization. The tensor factorization stores the bases in $\mathcal{G} \times_1 A^{(1)} \times_2 A^{(2)}$ (each basis image is stacked behind the other); analogously, the matrix factorization stores the vectorized bases in the columns of $A^{(1)}G$. The reconstructions from these bases are obtained by completing the multiplication with the final matrix factor in order to obtain the approximations (1.1) and (1.2). The remaining rows display the reconstruction of the original faces using the tensor and the matrix factorization with and without 1-norm regularization.

As we expect, the 1-norm regularization encourages sparse factors. In Figure 1 the NTF and NMF bases computed with 1-norm regularization are sparser (i.e., contain more black pixels) than the unregularized factors. In this experiment, the regularized NTF factors are about 10% sparser than the unregularized factors; in the NMF case, the regularized factors are nearly twice as sparse and the unregularized factors. In both cases we chose $\gamma = 10^{-2}$ for the regularization parameter.

An advantage of NTF over NMF is storage efficiency. In our implementation, both the core tensor $\mathcal{G}$ (for NTF) and the matrix G (for NMF) contain J nonzero elements ($J := J_1 = J_2 = J_3$ for $\mathcal{G}$ and $J := J_1 = J_2$ for G). Hence, we have to store $J(I_1+I_2+I_3+1)$ entries for NTF, and $J(I_1I_2+I_3+1)$ entries for NMF. In the example of Figure 1 with 1000 images of size 19×19 and an inner dimension of 15, the NTF requires storage for 15585 entries, and the NMF requires storage for 20430 entries. Thus, NMF requires just over 30% more storage than NTF on this example.

Our entire MATLAB implementation, including the scripts needed to reproduce Table 1 and Figure 1, can be obtained at `http://www.cs.ubc.ca/~mpf/lsntf`.

6 Looking ahead

We outline some of the important ingredients that need to be considered for the efficient implementation of an algorithm for computing the nonnegative tensor factorization. The

techniques we use for regularizing (§3.3) and scaling (§4.1) play important dual roles: they are useful for ensuring the efficiency of the method and also for encouraging solutions have desirable properties. Importantly, it seems that the tensor factorization can be computed without much more effort than is needed to compute the matrix factorization.

6.1 A Gauss-Newton approach

The alternating least-squares method given in Algorithm 1 has the attractive property that it decomposes the nonlinear problem into a sequence of well-structured subproblems for which there are effective solution methods. Although the asymptotic convergence rate of Gauss-Seidel methods is at most linear, the ALS algorithm still seems to perform effectively. An alternative to the ALS algorithm is to apply a Gauss-Newton method to (NTF), and optimize over all factors simultaneously (as opposed to one factor at a time).

The NTF problem can be reformulated as a generic nonlinear least-squares problem:

$$\underset{x}{\text{minimize}} \quad \tfrac{1}{2}\|c(x)\|_2^2 \quad \text{subject to} \quad x \geq 0.$$

In this case, $c(x)$ is an appropriate vectorization of $\mathcal{R}$ defined by (2.1). A variant of the Gauss-Newton method could be based on obtaining a correction Δx for the current iterate x via the solution of the regularized least-squares subproblem

$$\underset{\Delta x}{\text{minimize}} \quad \tfrac{1}{2}\|c(x) + J(x)\Delta x\|_2^2 + \tfrac{1}{2}\lambda\|\Delta x\|_2^2 \quad \text{subject to} \quad x + \Delta x \geq 0,$$

where J is the Jacobian of c, and λ is a positive damping parameter. Standard Levenberg-Marquadt rules can be used to update λ (see, e.g., [DS96]). With the simplifying assumption that $J_n \equiv J$ for each $n = 1, \ldots, N$, then the Jacobian for the NTF problem is

$$J(x) = \begin{bmatrix} D_1 & R_1 & B_1 \\ D_2 & R_2 & B_2 \\ \vdots & \vdots & \vdots \\ D_{I_3} & R_{I_3} & B_{I_3} \end{bmatrix},$$

where

$$X_\ell = \begin{bmatrix} X_{1\ell 1} & X_{1\ell 2} & \cdots & X_{1\ell J} \\ X_{2\ell 1} & \ddots & & X_{2\ell J} \\ \vdots & & \ddots & \vdots \\ X_{I_2\ell 1} & \cdots & & X_{I_2\ell J} \end{bmatrix}, \quad \ell = 1, \ldots, I_3, \quad X = (D, R, B),$$

and the matrices D_{ijk}, R_{ijk}, and B_{ijk} are defined by

$$D_{ijk} = g_{kkk}a_{ik}^2 a_{jk}^3 I_{(I_1)}, \quad R_{ijk} = g_{kkk}a_{jk}^3 A^{(1)}_{(:,k)} e_i^T, \quad B_{ijk} = g_{kkk}a_{ik}^2 A^{(1)}_{(:,k)} e_j^T.$$

The Jacobian J is a large matrix, but its regular structure makes it amenable to the efficient implementation of a matrix-vector product routine.

6.2 Multiple right-hand sides

As commented in §3.1, the ALS subproblem ($\text{NTF}_{A^{(n)}}$) is actually a set of independent linear least-squares problems that can in principle be solved in parallel. If the subproblems did not have nonnegativity constraints, then it would be possible to compute a QR factorization of $A^n_{\otimes} G^T_{(n)}$ and reuse it to solve for each row of $A^{(n)}$. However, the nonnegativity constraints imply that only subsets of the columns of $A^n_{\otimes} G^T_{(n)}$ participate in the solution of each least-squares problem. Because these subsets cannot be known in advance, such an approach is not viable. Still, we can consider applying the same bound-constrained least-squares technique described in §4.2 to solve each of these problems in parallel.

Acknowledgments

We are indebted to Tammy Kolda, who introduced us to the tensor factorization problem during her visit to UBC as a distinguished lecturer, and whos papers in this subject were invaluable for our work. Also, our sincerest thanks to two anonymous referees whos detailed and thoughtful comments led to a significant revision of this paper.

References

[Ber82] D. P. Bertsekas. *Constrained Optimization and Lagrange Multiplier Methods*. Academic Press, New York, 1982.

[Ber99] D. P. Bertsekas. *Nonlinear Programming*. Athena Scientific, Belmont, MA, second edition, 1999.

[BK06a] B. W. Bader and T. G. Kolda. MATLAB tensor classes for fast algorithm prototyping. *ACM Trans. Math. Software*, 2006.

[BK06b] B. W. Bader and T. G. Kolda. MATLAB tensor toolbox version 2.0. http://csmr.ca.sandia.gov/~tgkolda/TensorToolbox/, 2006.

[BL06] MIT Center For Biological and Computation Learning. CBCL Face Database #1. http://www.ai.mit.edu/projects/cbcl, 2006.

[CDS01] S. S. Chen, D. L. Donoho, and M. A. Saunders. Atomic decomposition by basis pursuit. *SIAM Rev.*, 43(1):129–159, 2001.

[CGT00] A. R. Conn, N. I. M. Gould, and Ph. L. Toint. *Trust-Region Methods*. MPS-SIAM Series on Optimization. Society of Industrial and Applied Mathematics, Philadelphia, 2000.

[CRT05] E. J. Candés, J. Romberg, and T. Tao. Stable signal recovery from incomplete and inaccurate measurements. To appear in Comm. Pure Appl. Math., 2005.

[CRT06] E. J. Candés, J. Romberg, and T. Tao. Robust uncertainty principles: exact signal reconstruction from highly incomplete frequency information. *IEEE Trans. Info. Theory*, 52(2):489–509, February 2006.

[dLdMV01] L. de Lathauwer, B. de Moor, and J. Vandewalle. A multilinear singular value decomposition. *SIAM J. Matrix Anal. Appl.*, 21:1253–1278, 2001.

[DS96] J. E. Dennis and R. B. Schnabel. *Numerical Methods for Unconstrained Optimization and Nonlinear Equations*. Classics in Applied Mathematics. Society of Industrial and Applied Mathematics, Philadelphia, 1996. Originally published: Prentice-Hall, New Jersey, 1983.

[dSL07] V. de Silva and L. Lim. Tensor rank and the ill-posedness of the best low-rank approximation problem. To appear in *SIAM J. Matrix Anal. Appl.*, 2007.

[DT05] D. L. Donoho and J. Tanner. Sparse nonnegative solution of underdetermined linear equations by linear programming. *Proc. Nat. Acad. Sci. USA*, 102(27):9446–9451, 2005.

[FBH03] N. K. M. Faber, R. Bro, and P. K. Hopke. Recent developments in CANDECOMP/PARAFAC algorithms: a critical review. *Chemometr. Intell. Lab.*, 65:119–137, 2003.

[Fri06] M. P. Friedlander. BCLS: A large-scale solver for bound-constrained least squares. Available at http://www.cs.ubc.ca/~mpf/bcls/, 2006.

[Hoy04] P. O. Hoyer. Non-negative matrix factorization with sparseness constraints. *Journal of Machine Learning Research*, 5:1457–1469, 2004.

[HPS05] T. Hazan, S. Polak, and A. Shashua. Sparse image coding using a 3d non-negative tensor factorization. *iccv*, 1:50–57, 2005.

[HS05] M. Heiler and C. Schnorr. Learning non-negative sparse image codes by convex programming. In *Tenth IEEE International Conference on Computer Vision*, volume 2, pages 1667–1674, October 2005.

[HTF02] T. Hastie, R. Tibshirani, and J. Friedman. *The Elements of Statistical Learning*. Springer-Verlag, 2002.

[Kel95] C. T. Kelley. *Iterative Methods for Linear and Nonlinear Equations*, volume 16 of *Frontiers in applied mathematics*. SIAM, Philadelphia, 1995.

[Kol06] T. G. Kolda. Multilinear operators for higher-order decompositions. Technical report, Sandia National Laboratories, 2006.

[Lin07] C.-J. Lin. Projected gradient methods for non-negative matrix factorization. Technical report, Department of Computer Science, National Taiwan University, 2007. To appear in *Neural Computation*.

[LMV00] L. D. Lathauwer, B. D. Moor, and J. Vandewalle. On the best rank-1 and rank-$(R_1, R_2, \ldots, R_N)$ approximation of higher-order tensors. *SIAM J. Matrix Anal. Appl.*, 21(4):1324–1342, 2000.

[LS99] D. D. Lee and H. S. Seung. Learning the parts of objects by non-negative matrix factorization. *Nature*, 401:788–791, 1999.

[LS01] D. D. Lee and H. S. Seung. Algorithms for non-negative matrix factorization. In Todd K.Leen, Thomas G. Dietterich, and Volker Tresp, editors, *Advances in Neural Information Processing Systems 13*, pages 556–562. MIT Press, 2001.

[LT92] Z. Q. Luo and P. Tseng. On the convergence of the coordinate descent method for convex differentiable minimization. *Journal of Optimization Theory and Applications*, 72:7–35, 1992.

[PS82] C. C. Paige and M. A. Saunders. LSQR: An algorithm for sparse linear equations and sparse least squares. *ACM Trans. Math. Software*, 8:43–71, 1982.

[PT94] P. Paatero and U. Tapper. Positive matrix factorization: A non-negative factor model with optimal utilization of error. *Envirometrics*, 5:111–126, 1994.

[SH05] A. Shashua and T. Hazan. Non-negative tensor factorization with applications to statistics and computer vision. In *Proceedings of ICCV*, 2005.

[SL01] A. Shashua and A. Levin. Linear image coding for regression and classification using the tensor-rank principle. In *In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2001.

[WW01] M. Welling and M. Weber. Positive tensor factorization. *Pattern Recog. Letters*, 22:1255–1261, 2001.