Department of Computer Science Technical Report TR-2006-21
October 2006, University of British Columbia

Computing nonnegative tensor factorizations

Michael P. Friedlander[*] Kathrin Hatz[†]

October 19, 2006

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). Also, the implementation provides a mechanism, based on ℓ_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 the case in which each component of $\mathcal{V}$ is nonnegative. We henceforth use the shorthand

[*]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).

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 $\mathcal{G}$, $A^{(1)}, \dots, A^{(N)}$ 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 $n = 1, \dots, N$, is a matrix; $\mathcal{G}$ is called the *core tensor.*) The factorization then satisfies

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

The properties of these factors and the operators $\times_1, \times_2, \dots$ are described in detail in §4. In particular, the factors are chosen as the solution of the nonlinear least-squares problem

$$\boxed{\text{(NTF)} \qquad \begin{array}{ll} \underset{\mathcal{G}, A^{(1)}, \dots, A^{(N)}}{\text{minimize}} & \tfrac{1}{2} \left\| \mathcal{G} \times_1 A^{(1)} \times_2 \cdots \times_N A^{(N)} - \mathcal{V} \right\|_F^2 \\ \text{subject to} & \mathcal{G}, A^{(1)}, \dots, A^{(N)} \geq 0. \end{array}}$$

The nonnegativity constraint in (NTF) can be motivated in several ways. Based on intuition, the parts of a dataset (represented by the factors in (1.1)) are thought to generally combine additively. 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, 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.

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 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 $A^{(1)}, \dots, A^{(N)}$. We note that our implementation can be easily extended to handle the case where $N > 3$ and $\mathcal{G}$ is optimized explicitly, as shown in (NTF). It turns out that keeping the factors equilibrated is critical for the efficiency of this method.

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. We describe this solver in §5.2.

2 Alternating least squares

The alternating least squares (ALS) approach is a special case of the block coordinate-descent method, also knows as the block Gauss-Seidel (BGS) method. 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 nonnegative linear least-squares subproblems.

We describe some of the main features of this approach in terms of the general optimization problem

$$\underset{x}{\text{minimize}} \quad f(x) \quad \text{subject to} \quad x \in X, \tag{2.1}$$

where $f : \mathbb{R}^n \to \mathbb{R}$ is a continuously differentiable function and the feasible set $X = X_1 \times \cdots \times X_m \subseteq \mathbb{R}^n$ is the Cartesian product of closed, nonempty, and convex subsets $X_i \subseteq \mathbb{R}^{n_i}$, for $i = 1, \dots, m$, with $\sum_{i=1}^m n_i = n$. Let $x \in \mathbb{R}^n$ be partitioned into m component subvectors $x_i \in \mathbb{R}^{n_i}$. Each iteration of the BGS method is then defined by

$$x_i^{k+1} = \arg\min_{y \in X_i} \; f(x_1^{k+1}, \dots, x_{i-1}^{k+1}, y, x_{i+1}^k, \dots, x_m^k), \tag{2.2}$$

for $i = 1, \dots, m$. These subproblem solution generate a sequence $\{x^k\}$ with $x^k = (x_1^k, \dots, x_m^k)$. If the solution x_i^{k+1} of each subproblem (2.2) is uniquely attained, then it is possible to show that every limit point of $\{x_k\}$ is a stationary point for (2.2). (See, e.g., [Ber99, §2.7] for a more detailed presentation of BGS methods.) The convergence rate of BGS is linear [LT92]. No convexity assumption on f is needed.

In the particular case where (2.1) is taken as (NTF), then (2.2) reduces to a linear least-squares problem and each X_i is the nonnegative orthant. The "uniquely attained" assumption is then equivalent to the requirement that the matrices of the least-squares problems have full rank. One possible way to safeguard against a lack of uniqueness is to augment the subproblems with a proximal-point term, and instead to solve

$$x_i^{k+1} = \arg\min_{y \in X_i} \; f(x_1^{k+1}, \dots, x_{i-1}^{k+1}, y, x_{i+1}^k, \dots, x_m^k) + \tfrac{1}{2}\tau_i \|y - x_i^k\|_2^2, \tag{2.3}$$

$i = 1, \dots, m$, at each iteration, where each τ_i is a positive scalar. The proximal-point term serves to convexify the objective and ensure that the subproblem solutions are unique. Every limit point of the sequence of vectors $\{x^k\}$ generated by (2.3) is a critical point of (2.1) [GS99].

We experimented with the proximal-point term but found that in practice it could slow down the convergence rate of the algorithm. Instead, our implementation relies on the ℓ_1 regularization function which is not strictly convex, and thus does not guarantee uniqueness of the solution. However, this heuristic has the added benefit that it can help encourage sparse solutions to the overall problem. We discuss this further in §5.

3 Tensors

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.

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 $n = 1, \dots, N$. (Note that we use n as an index variable throughout the remainder of the paper.)

3.1 The n-mode product

The n-mode product defines the multiplication of a tensor with a matrix. The 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 $J_1 \times \cdots \times J_{n-1} \times I_n \times J_{n+1} \times \cdots \times J_N$.

We may also define this product elementwise. 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. The n-mode product can then 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 $\mathcal{G}$.

3.2 The n-mode matricization

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

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

3.3 Notation

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.

We often make use of transformations that change tensors into matrices (n-mode matricization), and matrices into vectors (vectorization). We find it useful to use a lower case version of an upper case letter to denote the vectorizations of some matrices and of some n-mode tensor matricizations. Define vectorizations as follows:

matrix A	$a = \operatorname{vec} A,$
matrix A^T	$\bar{a} = \operatorname{vec} A^T,$
matrix $A^{(n)}$	$a^n = \operatorname{vec} A^{(n)},$
matrix $A^{(n)T}$	$\bar{a}^n = \operatorname{vec} A^{(n)T},$
n-mode matricization of the tensor $\mathcal{G}$	$g_n = \operatorname{vec} G_{(n)},$
... and its transpose	$\bar{g}_n = \operatorname{vec} G_{(n)}^T.$

The Kronecker product (denoted by $\otimes$) and the Khatri-Rao product (denoted by $\odot$; see [Kol06, §3.1]) 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

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

4 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

$$A^n_{\otimes} G^T_{(n)} \quad \text{has dimensions} \quad \prod_{\substack{i=1 \\ i \neq n}}^{N} I_i \times J_n \tag{4.1}$$

(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 (4.1) 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

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

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 (4.1). 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 §5, where we assume that $\mathcal{G}$ is diagonal.

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

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

of (NTF) with respect to each $A^{(n)}$ and $\mathcal{G}$. These derivatives are given by

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

where

$$\mathcal{R} := \mathcal{G} \times_1 A^{(1)} \times_2 \cdots \times_N A^{(N)} - \mathcal{V} \tag{4.2}$$

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

4.1 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

$$X_{(n)} = A^{(n)} G_{(n)} (A^n_{\otimes})^T \tag{4.3}$$

and

$$x_1 \equiv \operatorname{vec} X_{(1)} = A_{\otimes} g_1. \tag{4.4}$$

In order to isolate any given factor $A^{(n)}$, $n = 1, \ldots, N$, we use (4.3) to rewrite the objective of (NTF) as

$$\begin{aligned} \phi(\mathcal{G}, A^{(1)}, \ldots, A^{(N)}) &\equiv \tfrac{1}{2} \left\| \mathcal{G} \times_1 A^{(1)} \times_2 \cdots \times_N A^{(N)} - \mathcal{V} \right\|_F^2 \\ &= \tfrac{1}{2} \left\| A^{(n)} G_{(n)} (A^n_{\otimes})^T - V_{(n)} \right\|_F^2 \\ &= \tfrac{1}{2} \left\| A^n_{\otimes} G^T_{(n)} A^{(n)T} - V^T_{(n)} \right\|_F^2 . \end{aligned} \tag{4.5}$$

From (4.5) 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 (4.5) and rewriting ϕ as

$$\begin{aligned} \phi(\mathcal{G}, A^{(1)}, \ldots, A^{(N)}) &= \tfrac{1}{2} \| \operatorname{diag}(A^n_{\otimes} G^T_{(n)}, \ldots, A^n_{\otimes} G^T_{(n)}) \bar{a}^n - \bar{v}_n \|_2^2 \\ &= \tfrac{1}{2} \| \big(I_{(I_n)} \otimes (A^n_{\otimes} G^T_{(n)}) \big) \bar{a}^n - \bar{v}_n \|_2^2 \end{aligned}$$

(recall that $\bar{a}^n = \operatorname{vec} A^{(n)T}$ and $\bar{v}_n = \operatorname{vec} V^T_{(n)}$; cf. §3.3). The resulting N subproblems (one for each $A^{(n)}$) are standard nonnegative linear least-squares problems over the vector $\bar{a}^n$:

$$\boxed{(\mathrm{NTF}_{A^{(n)}}) \qquad \underset{\bar{a}^n}{\text{minimize}} \;\; \| \big(I_{(I_n)} \otimes (A^n_{\otimes} G^T_{(n)}) \big) \bar{a}^n - \bar{v}_n \|_2 \quad \text{subject to} \quad \bar{a}^n \geq 0.}$$

Note that 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 (4.4) to transform the objective as

$$\phi(\mathcal{G}, A^{(1)}, \ldots, A^{(N)}) = \tfrac{1}{2} \left\| A_{\otimes} g_1 - v_1 \right\|_2^2 . \tag{4.6}$$

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 $A_0^{(1)}, \ldots, A_0^{(n)}$, $\mathcal{G}_0 \geq 0$, $k \leftarrow 0$
repeat
    for $n = 1, \ldots, N$ do
        $A_{k+1}^{(n)} \leftarrow$ solve (NTF$_{A^{(n)}}$)
    $\mathcal{G}_{k+1} \leftarrow$ solve (NTF$_{\mathcal{G}}$)
    $k \leftarrow k + 1$
until converged
$\mathcal{G}_* \leftarrow \mathcal{G}_k$, and $A_*^{(n)} \leftarrow A_k^{(n)}$ for $n = 1, \ldots, N$
```

The optimization over $\mathcal{G}$ is then reduced to the nonnegative linear least-squares problem over the vector g_1:

$$\boxed{(\mathrm{NTF}_{\mathcal{G}}) \qquad \underset{g_1}{\text{minimize}} \;\; \|A_{\otimes} g_1 - v_1\|_2 \quad \text{subject to} \quad g_1 \geq 0.}$$

Algorithm 1 describes the basic ALS method for (NTF) that is based on solving a sequence of subproblems defined by (NTF$_{A^{(n)}}$) and (NTF$_{\mathcal{G}}$). The "vec" operator is then inverted to derive $A^{(1)}, \ldots, A^{(n)}$ and $\mathcal{G}$ from the solutions $\bar{a}^1, \ldots, \bar{a}^n$ and g_1, respectively, of (NTF$_{A^{(n)}}$) and (NTF$_{\mathcal{G}}$).

4.2 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 ℓ_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 ℓ_1 regularization include signal processing for the recovery of sparse signals; see, e.g., [CDS01, CRT04, CRT05, DT05].

An important side effect of regularizing the NTF problem is that it can help to keep the solution bounded. As we discuss in §5.1, the solution set of (NTF) is necessarily unbounded, which can be a source of numerical difficulties. We describe there a mechanism for ensuring that the computed factors are well scaled.

We define the regularized NTF problem as

$$\boxed{\begin{array}{ll} (\mathrm{NTF}_{\mathrm{sp}}) \qquad & \underset{\mathcal{G}, A^{(1)}, \ldots, A^{(N)}}{\text{minimize}} \;\; \phi(\mathcal{G}, A^{(1)}, \ldots, A^{(N)}) + \gamma \left(\sum_{n=1}^{N} \|a^n\|_1 + \|g_1\|_1 \right) \\ & \text{subject to} \;\; \mathcal{G}, A^{(1)}, \ldots, A^{(N)} \geq 0, \end{array}}$$

where γ is a nonnegative regularization parameter. (Recall that $a^n \equiv \operatorname{vec} A^{(n)}$, and so the term $\sum_{n=1}^{N} \|a^n\|_1$ can be interpreted as an ℓ_1-norm penalty on the columns of the factors $A^{(n)}$.) Importantly, the nonnegativity constraints can be leveraged to re-express the nondifferentiable ℓ_1 norm as a linear function. Note that for any vector x,

$$\|x\|_1 = e^T x \qquad \text{if} \qquad x \ge 0. \tag{4.7}$$

The alternating least-squares approach derived in §4.1 extends easily to the regularized problem (NTF$_{\text{sp}}$). With (4.7), we can write the linear least-squares subproblems (NTF$_{A^{(n)}}$) and (NTF$_{\mathcal{G}}$), respectively, as

$$\underset{\bar{a}^n}{\text{minimize}} \quad \tfrac{1}{2}\big\|\big(I_{(I_n)} \otimes (A^n_{\otimes} G^T_{(n)})\big)\bar{a}^n - \bar{v}_n\big\|_2^2 + \gamma e^T \bar{a}^n \quad \text{subject to} \quad \bar{a}^n \ge 0, \tag{4.8a}$$

and

$$\underset{g_1}{\text{minimize}} \quad \tfrac{1}{2}\|A_{\otimes} g_1 - v\|_2^2 + \gamma e^T g_1 \quad \text{subject to} \quad g_1 \ge 0. \tag{4.8b}$$

Each subproblem now has an additional linear term. Note that we discard the constant linear terms that correspond to the other fixed variables. Also, we use the fact that $\|\bar{a}^n\|_1 = \|a^n\|_1$.

4.3 Specialization to matrices (NMF)

When $N = 2$, the NTF problem reduces to the well-known matrix factorization problem. We find that specializing our development of the tensor factorization problem to the special matrix case clarifies our approach, and leads to a nonnegative matrix factorization that is slightly different from what typically appears in the literature.

For this section only, let

$$W := A^{(1)}, \quad H := A^{(2)}, \quad G := \mathcal{G}, \quad \text{and} \quad V := \mathcal{V};$$

recall that with $N = 2$, $\mathcal{G}$ and $\mathcal{V}$ are simply 2-mode tensors—i.e., matrices. Therefore,

$$G \times_1 W \times_2 H = W G H^T, \tag{4.9}$$

and the NTF problem reduces to

$$\boxed{\text{(NMF)} \qquad \begin{array}{ll} \underset{G,W,H}{\text{minimize}} & \tfrac{1}{2}\left\|WGH^T - V\right\|_F^2 \\ \text{subject to} & G, W, H \ge 0. \end{array}}$$

We note that in the literature, (NMF) typically appears with the objective $\frac{1}{2}\|WH - V\|_F^2$; the implication is that the core matrix $G \equiv I$. However, our formulation keeps G explicit. A second, apparently small, difference is the transpose on the factor H. These two differences have a crucial implication for our implementation, as we discuss in §5.1.

The linear least-squares subproblems that correspond to (NTF$_{A^{(n)}}$) and (NTF$_{\mathcal{G}}$) are

$$\begin{array}{llll} (\text{NMF}_W) & \underset{\bar{w}}{\text{minimize}} \; \tfrac{1}{2}\|(I \otimes HG)\bar{w} - \bar{v}\| & \text{subject to} & \bar{w} \ge 0, \\ (\text{NMF}_H) & \underset{\bar{h}}{\text{minimize}} \; \tfrac{1}{2}\big\|(I \otimes WG)\bar{h} - v\big\| & \text{subject to} & \bar{h} \ge 0, \\ (\text{NMF}_G) & \underset{g}{\text{minimize}} \; \tfrac{1}{2}\|(H \otimes W)g - v\| & \text{subject to} & g \ge 0, \end{array}$$

and analogous to the definitions in §3.3, we set $v := \operatorname{vec} V$, $\bar{v} := \operatorname{vec} V^T$, $\bar{w} := \operatorname{vec} W^T$, and $g := \operatorname{vec} G$.

As discussed in the context of (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 ℓ_1-regularization approach discussed in §4.2 extends immediately to the matrix case with the addition of a linear term to each of the subproblems.

5 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. (4.1)).

We discuss below a method for keeping the factors $A^{(n)}$ well scaled. 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 bound-constrained linear-least-squares subproblems.

5.1 Scaling

In the implementation of our NTF method, we maintain $\mathcal{G}$ diagonal and remove the subproblem (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 keep the factors equilibrated. This device is needed because the solution set of (NTF) is necessarily unbounded. To see this, note that if $\{\mathcal{G}, A^{(1)}, \dots, A^{(n)}\}$ is any solution of (NTF), then

$$\left\{\mathcal{G}, \alpha A^{(1)}, \frac{1}{\alpha}A^{(2)}, A^{(3)}, \dots, A^{(N)}\right\},$$

for example, is also a solution of (NTF) for every $\alpha > 0$. This is true for any scaling of the factor $A^{(n)}$ or of the core tensor $\mathcal{G}$. On first glance, it would seem that this property of (NTF) would be a significant obstacle for the convergence of any algorithm. However, we can use this extra degree of freedom as a device for maintaining iterates with a scale most favourable for an efficient solution of the underlying linear least-squares subproblems (cf. §4.1). In particular, we are at liberty to choose a scaling of each factor $A^{(n)}$ in order to ensure that every column of $A^{(n)}$ has unit norm, which is especially useful in our implementation, 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 $G := D_W D_H$ be a product of diagonal matrices D_W and D_H, and rewrite (4.9) as

$$WGH^T = WD_W D_H H^T = (WD_W)(HD_H)^T = \bar{W}\bar{H}^T, \tag{5.1}$$

where and $\bar{W} := WD_W$ and $\bar{H} := HD_H$. The diagonal operators D_w and D_h simply rescale the columns of W and H. Given factors $\bar{W}$ and $\bar{H}$, the freedom to choose G in (5.1) implies that we can choose any scale we like for the columns of W and H, and instead absorb the scales of $\bar{W}$ and $\bar{H}$ 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 H is an important ingredient in making the transformation in (5.1) 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 $n = 1, \dots, N$, we rescale the columns of the factors $A^{(n)}$ with diagonal matrices $D^{(n)}$ as follows: define the diagonal tensor

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

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 $A_0^{(1)}, \ldots, A_0^{(n)} \geq 0$, $\mathcal{G}_0 = \mathcal{I}$, $k \leftarrow 0$
repeat
    for $n = 1, 2, \ldots, N$ do
        $A^{(n)} \leftarrow$ solve $(\mathrm{NTF}_{A^{(n)}})$
        Compute $D_{k+1}^{(n)}$ for $A^{(n)}$                    [compute column scales; see (5.3)]
        $A_{k+1}^{(n)} \leftarrow A^{(n)} D_{k+1}^{(n)}$                    [compute scaled factor]
        $\mathcal{G}_{k+1} \leftarrow \mathcal{G}_k \times_n (D_{k+1}^{(n)})^{-1}$          [update diagonal core tensor]
    $k \leftarrow k + 1$
until converged
$\mathcal{G}_* \leftarrow \mathcal{G}_k$, and $A_*^{(n)} \leftarrow A_k^{(n)}$ for $n = 1, \ldots, N$
```

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

$$\begin{aligned}
&\mathcal{G} \times_1 A^{(1)} \times_2 \cdots \times_N A^{(N)} \\
&\qquad = \left[\mathcal{I} \times_1 D^{(1)} \times_2 \cdots \times_N D^{(N)}\right] \times_1 \left[A^{(1)} \times_2 \cdots \times_N A^{(N)}\right] \\
&\qquad = \mathcal{I} \times_1 \left[A^{(1)} D^{(1)} \times_2 \cdots \times_N A^{(N)} D^{(N)}\right] \\
&\qquad = \mathcal{I} \times_1 \bar{A}^{(1)} \times_2 \cdots \times_N \bar{A}^{(N)},
\end{aligned} \tag{5.2}$$

where each $\bar{A}^{(n)} := A^{(n)} D^{(n)}$. 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; it is exactly analogous to (5.1).

After a linear least-squares subproblem is solved at iteration k, the solution $A_k^{(n)}$ is rescaled by

$$D^{(n)} = \operatorname{diag}\left(e^T A^{(n)}\right)^{-1}, \tag{5.3}$$

so that the resulting columns of the scaled matrix $A^{(n)} D^{(n)}$ each sum to one. Thus, $\left\|A^{(n)} D^{(n)}\right\|_1 = 1$. We can interpret this rescaling as a projection of each of the Gauss-Seidel iterates onto the convex set of nonnegative matrices with induced ℓ_1-matrix norm. (We might be tempted to simply scale the entire matrix so that $\operatorname{vec} A^{(n)}$ would have unit ℓ_2-norm, but then we could not interpret the rescaling as a projection onto a convex set.) It can be shown that the Gauss-Seidel iterations still converge under projections to convex sets. Algorithm 2 describes our implementation of this scaling strategy.

5.2 Solving the least-squares subproblems

The computational kernel of the alternating least-squares algorithm is the solution of the nonnegative linear least-squares problems $(\mathrm{NTF}_{A^{(n)}})$ and $(\mathrm{NTF}_{\mathcal{G}})$ described in §4.1. 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 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

$$\begin{array}{ll} \underset{x\in\mathbb{R}^n}{\text{minimize}} & \tfrac{1}{2}\|Ax-b\|_2^2 + c^Tx + \tfrac{1}{2}\gamma^2\|x\|_2^2 \\ \text{subject to} & \ell \le x \le u, \end{array} \tag{5.4}$$

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:

$$x = \begin{bmatrix} x_B & x_N \end{bmatrix}, \quad c = \begin{bmatrix} c_B & c_N \end{bmatrix}, \quad \text{and} \quad A = \begin{bmatrix} A_B & A_N \end{bmatrix}. \tag{5.5}$$

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

$$\begin{bmatrix} A_B^T A_B + \gamma^2 I & 0 \\ 0 & D \end{bmatrix} \begin{bmatrix} \Delta x_B \\ \Delta x_N \end{bmatrix} = A^T r - c - \gamma^2 x, \tag{5.6}$$

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 (5.4). 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 (5.6) is never formed explicitly. Instead, Δx_B is computed equivalently as a solution of the least-squares problem

$$\underset{\Delta x_B}{\text{minimize}} \quad \tfrac{1}{2}\|A_B \Delta x_B - r\|_2^2 + c_B^T \Delta x_B + \tfrac{1}{2}\gamma^2 \|x_B + \Delta x_B\|_2^2. \tag{5.7}$$

We find an approximate solution to (5.7) by applying the conjugate-gradient-type solver LSQR [PS82] to the problem

$$\underset{\Delta x_B}{\text{minimize}} \quad \left\| \begin{bmatrix} A_B \\ \beta I \end{bmatrix} \Delta x_B - \begin{bmatrix} r \\ \frac{1}{\beta} c_B - \frac{\gamma^2}{\beta} x_B \end{bmatrix} \right\|, \tag{5.8}$$

where $\beta = \max\{\gamma, \bar{\gamma}\}$ and $\bar{\gamma}$ is a small positive constant. If $\gamma < \bar{\gamma}$ (as it is with the ALS subproblems of §4.1), 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 §5.1 implies that the solution Δx_B will be well scaled. This is an especially favorable circumstance for CG-type solvers.

6 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

Algorithm	Iterations	Optimality	Total time (sec)
lsNTF	6	4.9e−04	315
lsNMF	3	2.2e−04	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*

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 grayscale 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.

Table 1 shows the relative performance of our matrix and tensor factorization implementations (lsNMF and lsNTF, respectively). Importantly, we are able to solve the NTF and NMF problems with very similar computing times.

For interest, we also show in Table 1 the results of solving the same NMF problem with a projected gradient method [Lin05] and multiplicative update method [LS01] (these are the rows labeled projGradNMF and multUpdateNMF). 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. The multiplicative update method failed to converge within its allotted maximum of 1000 iterations. All runs were conducted on a 3.2 GHz Intel Pentium 4 running Linux 2.6.16 and MATLAB 7.2.

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

7 Discussion

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 (§4.2) and scaling (§5.1) 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.

7.1 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:

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

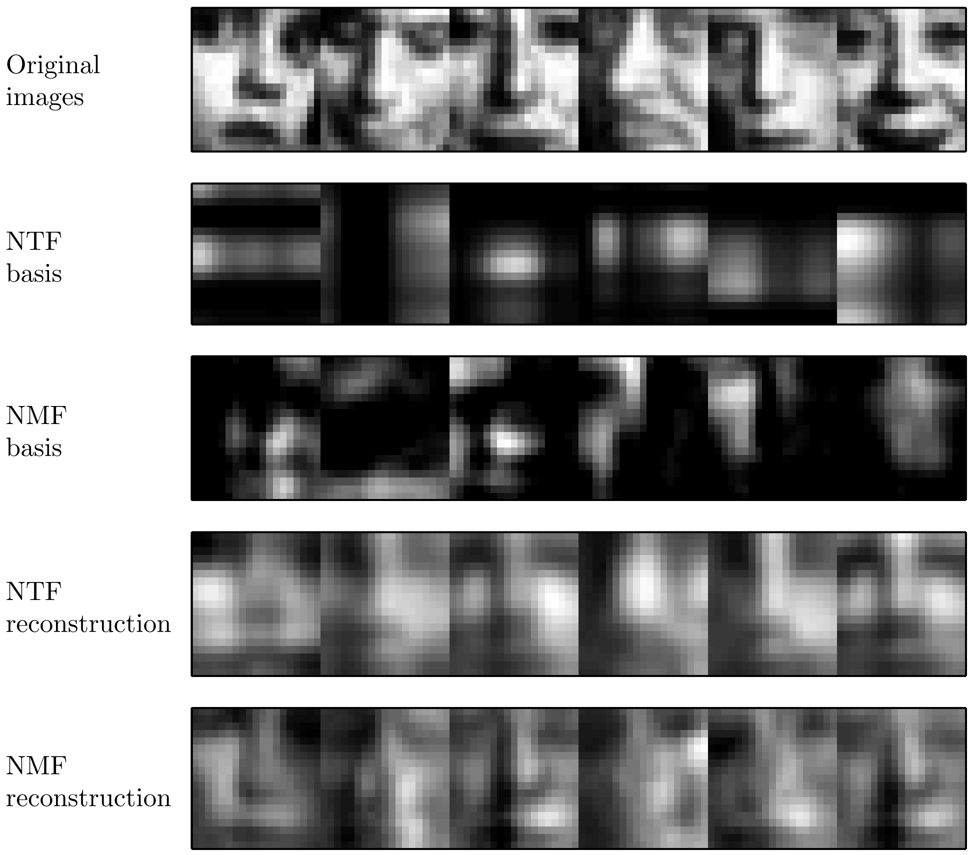

FIGURE 1: *The first row shows six of the original images in the CBCL test set. The second and third rows show the bases computed from 1000 images in the dataset using the NTF and NMF methods. The last two rows show reconstructions of the original faces using the tensor and matrix factorizations.*

In this case, $c(x)$ is an appropriate vectorization of $\mathcal{R}$ defined by (4.2). A variant of the Gauss-Newton method could be based on obtaining a correction $\mathit{\Delta} x$ for the current iterate x via the solution of the regularized least-squares subproblem

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

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

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

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

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

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

7.2 Multiple right-hand sides

As commented in §4.1, the ALS subproblem (NTF$_{A^{(n)}}$) is actually a set of independent linear least-squares problems that can in pricinple 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 §5.2 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.

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.

[CRT04] E. J. Candés, J. Romberg, and T. Tao. Robust uncertainty principles: Exact signal reconstruction from highly incomplete frequency information. Technical report, Applied and Computational Analysis, California Institute of Technology, 2004.

[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.

[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.

[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.

[GS99] L. Grippo and M. Sciandrone. On the convergence of the block nonlinear gauss-seidel method under convex constraints. *Operations Research Letter 26*, pages 127–136, 1999.

[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. Technical report, The Hebrew University, 2005.

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

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

[Lin05] C.-J. Lin. Projected gradient methods for non-negative matrix factorization. Technical report, Department of Computer Science, National Taiwan University, 2005.

[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 ans 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 *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.