UNIVERSITÄT SALZBURG

Nonnegative Matrix Factorization: Algorithms and Parallelization

Gabriel Okša Martin Bečka Marián Vajteršic

Technical Report 2010-05 June 2010

Department of Computer Sciences

Jakob-Haringer-Straße 2
5020 Salzburg
Austria
www.cosy.sbg.ac.at

Technical Report Series

Nonnegative Matrix Factorization: Algorithms and Parallelization

Gabriel Okša, Martin Bečka and Marián Vajteršic

Abstract. *An alternative to singular value decomposition (SVD) in the information retrieval is the low-rank approximation of an original non-negative matrix A by its non-negative factors U and V. The columns of U are the feature vectors with no non-negative components, and the columns of V store the non-negative weights that serve for the combination of feature vectors. First experiments show that restricting the decomposition of a non-negative matrix A to non-negative factors leads to the better efficiency than the low-rank approximation using the SVD of A. Also, the non-negative feature vectors can be more easily interpreted in terms of a coded information than the left (right) singular vectors (which usually contain also negative components). We describe basic serial iterative algorithms for minimizing the difference $A - UV$ in the Frobenius norm together with possible additional constraints. Next, the analysis of a possible parallelization of serial algorithms for distributed parallel architecture is provided.*

1 Introduction

Our society can be characterized by an increasing amount of data coming form an advanced technology, like antennas, sensors on satellites, images fromm space and from human body, etc. This continuing stream of data should be evaluated and interpreted quickly and reliably. Since incoming data can be inexact, it is often important to represent them using some compressing mechanism, so that their ambiguity is reduced. In signal and image processing community, such an approach is well-known for years as the *noise removal problem*. The common ground of such techniques is to replace the original data by a lower dimensional representation obtained via subspace approximation. In statistics, factor analysis and principal component analysis are two methods with a long history that accomplish the reduction of variables and detection of structures among them.

The acquired data is often nonnegative and the low-rank representation is further required to preserve the nonnegativity in order to avoid some contradictions and/or to make the data interpretation easier. Classical tools, like the Singular Value Decomposition (SVD), cannot guarantee to maintain the nonnegativity. For example, even for a nonnegative matrix A its SVD consists of nonnegative singular values, but the components of right and left singular vectors can be of positive or negative sign (or zero). When one is interested in maintaining

the nonnegativity also in the low-rank approximation, than the so-called *Nonnegative Matrix Factorization* (NNMF) problem must be solved [2].

NNMF problem. Given a nonnegative matrix $A \in \mathbb{R}^{m\times n}$ and a positive integer $k < \min\{m,n\}$, find nonnegative matrices $U \in \mathbb{R}^{m\times k}$ and $V \in \mathbb{R}^{k\times n}$ that minimize the functional

$$f(U,V) = \frac{1}{2}\|A - UV\|_{\mathrm{F}}^2. \tag{1}$$

Notice that the product is an approximate factorization of A of rank at most k. The choice of k is problem dependent and in most cases $k \ll \min\{m,n\}$ so that UV can be interpreted as the compressed form of data in A.

Numerical algorithms that solve the minimization problem (1) extract the underlying features as basis vectors (columns) in U, which can be then used for identification and classification. Since the entries in U and V are nonnegative, any multiplication of basis vectors in U by a column from V is a non-subtractive combination of parts to form the whole. Features may be, for example, parts of faces in image data, topics or clusters in textual data, specific absorption characteristics in hyperspectral data, etc.

From the mathematical point of view, the important question is that of existence of local minima due to the non-convexity of $f(U,V)$ in both U and V, and a lack of a unique solution: when (U,V) is a solution then also $(UD, D^{-1}V)$ is also a solution for any nonnegative matrix D of order k whose inverse is also nonnegative (take for D, for example, any diagonal matrix with nonnegative entries). Of course, one is interested in finding the global minimum (if it exists) in solving (1). However, even local minima can provide a useful data compression and feature extraction.

From a mathematical point of view, the existence of a solution of (1) is closely connected to the fulfillment of so-called *first order optimality conditions*; see [3]. The cone $\mathbb{R}_+^{m\times k}$ of nonnegative matrices in $\mathbb{R}^{m\times k}$ can be defined as the set

$$\mathbb{R}_+^{m\times k} = \{F \in \mathbb{R}^{m\times k} | F = E.*E, E \in \mathbb{R}^{m\times k}\},$$

where $M.*N = [m_{ij}n_{ij}]$ denotes (as in MATLAB) the Hadamard product of two matrices. The expression $E.*E$ is one way how to parametrize nonnegative matrices over the open set $\mathbb{R}^{m\times k}$. This parametrization is differentiable and the NNMF problem can now be expressed as the minimization of

$$f(U,V) = \frac{1}{2}\|A - (U.*U)(V.*V)\|_{\mathrm{F}}^2, \tag{2}$$

where variables (U,V) are from the open set $\mathbb{R}^{m\times k} \times \mathbb{R}^{k\times n}$. We see that this parametrization transforms the constrained optimization over the cones into a problem with no constraints at all.

Consider f as a differentiable functional over the space $\mathbb{R}^{m\times k} \times \mathbb{R}^{k\times n}$ with the product Frobenius scalar product,

$$[(X_1,Y_1),(X_2,Y_2)] = [X_1,X_2] + [Y_1,Y_2],$$

for all $X_1, X_2 \in \mathrm{R}^{m\times k}$ and $Y_1, Y_2 \in \mathrm{R}^{k\times n}$. Then the Fréchet derivative of $f(U,V)$ can be computed component by component. In particular, the partial derivative of f with respect to U acting on an arbitrary $E \in \mathrm{R}^{m\times k}$ is given by

$$\begin{aligned}\frac{\partial f}{\partial U}.E &= [-(E.*U + U.*E)(V.*V), \delta(U,V)] \\ &= [-2E, U.*(\delta(U,V)(V.*V)^T)],\end{aligned}$$

where, for convenience,

$$\delta(U,V) := A - (U.*U)(V.*V).$$

Similarly, the partial derivative of f with respect to V acting on an arbitrary $F \in \mathrm{R}^{k\times n}$ is given by

$$\frac{\partial f}{\partial V}.F = [-2F, V.*((U.*U)^T\delta(U,V))].$$

By the Riesz representation theorem, the gradient of f at (U,V) can be expressed as

$$\nabla f(U,V) = \left(-2U.*(\delta(U,V)(V.*V)^T), -2V.*((U.*U)^T\delta(U,V))\right),$$

which is the mathematical object in the product space $\mathbb{R}^{m\times k} \times \mathbb{R}^{k\times n}$.

Now it is possible to characterize the first order optimality condition for the NNMF problem by setting both gradient components equal to zero.

If (W,H) is a local minimizer of the objective functional f in (2), then the equations

$$\begin{aligned}W.*(\delta(W,H)(H.*H)^T) &= 0 \in \mathbb{R}^{m\times k}, \\ H.*((W.*W)^T\delta(W,H)) &= 0 \in \mathbb{R}^{k\times n}.\end{aligned} \tag{3}$$

Consequently, the necessary condition for $(U,V) \in \mathbb{R}_+^{m\times k} \times \mathbb{R}_+^{k\times n}$ to solve the NNMF problem is given by two equations,

$$\begin{aligned}U.*((A-UV)V^T) &= 0 \in \mathbb{R}^{m\times k}, \\ V.*(U^T(A-UV)) &= 0 \in \mathbb{R}^{k\times n}.\end{aligned} \tag{4}$$

First formulation in (3) works over the open set $\mathbb{R}^{m\times k} \times \mathbb{R}^{k\times n}$ with no constraints at all. In the second formulation (4), it is *assumed* that the components of U and V are nonzero. If this assumption is violated then it must be restored by some *projection* step; for example, all negative components of U and V are set to zero.

It is interesting to note the complementarity condition of zeros in (4): if the (i,j) entry of U is not zero, then the corresponding entry in the product $(A-UV)V^T$ must be zero, end *vice versa*. In the optimization literature, this is known as a characterization of the *Kuhn-Tucker condition* for minimization of (1) subject to the nonnegativity constraint [6]. In particular, the following inequalities are also necessary.

The two matrices $-(A-UV)V^T$ and $-U^T(A-UV)$ are precisely the Lagrangian multipliers specified in the Kuhn-Tucker condition. At a solution (U,V) of the NNMF problem (1), it is

necessary that

$$\begin{aligned} (A - UV)V^T &\leq 0 \in \mathbb{R}^{m\times k}, \\ U^T(A - UV) &\leq 0 \in \mathbb{R}^{k\times n}. \end{aligned} \tag{5}$$

Next section contains a description of basic serial algorithms that are used for the solution of (1).

2 Basic algorithms for NNMF

2.1 ADI Newton iteration

Any critical point of the NNMFD problem must satisfy the nonlinear matrix equations (3). Hence, the Newton method seems to be a reasonable choice for solving this problem.

The size of A can be very large, and to compute *both* factors U and V form (4) can be computationally very demanding. A common approach is to *alternate* between U and V by fixing the other. This idea of *alternating direction iteration* (ADI) is used also in other fields of numerics.

We may start by fixing V in (4) and solve the system

$$U. * (B - UC) = 0 \tag{6}$$

for a nonnegative matrix $U \in \mathbb{R}_+^{m\times k}$, where both $B := AV^T \in \mathbb{R}^{m\times k}$ and $C := VV^T \in \mathbb{R}^{k\times k}$ are fixed and nonnegative matrices. We then fix U and solve next the system

$$V. * (R - SV) = 0 \tag{7}$$

for a nonnegative matrix $V \in \mathbb{R}_+^{k\times n}$, where both $R := U^T A \in \mathbb{R}^{k\times n}$ and $S := U^T U \in \mathbb{R}^{k\times k}$ are fixed and nonnegative matrices. Since k is usually small, the size of matrices C and S is relatively small, too. This is one sweep of the *outer loop iteration*.

Taking $U = BC^{-1}$ is (6) is not good enough because such U can have negative entries. Extra effort is needed to satisfy the complementary conditions in (4) and the inequalities in (5). Also, it is not clear under which conditions the outer loop iteration converges.

Interestingly, each row $u_i^T, 1 \leq i \leq m$, of U in matrix equation (6) is not related to any other rows. Hence, any u^T gives rise to a nonlinear system of equations

$$u_i^T . * (b_i^T - u_i^T C) = 0, \quad 1 \leq i \leq m. \tag{8}$$

Similarly if U is fixed, then V can be solved by columns in parallel:

$$v_j . * (r_j - Sv_j) = 0, \quad 1 \leq j \leq n. \tag{9}$$

We now omit the row (column) index and work with *any* row of U (column of V). To guarantee the nonnegativity of u^T, we rewrite (8) in the form

$$\psi(e) = (C(e. * e) - b). * e = 0, \tag{10}$$

using the fact that C is symmetric and by taking $u = e. * e, e \in \mathbb{R}^k$. Then the Fréchet derivative of ψ acting on $h \in \mathbb{R}^k$ is given by

$$\psi'(e).h = 2(C(e. * h)). * e + (C(e. * e) - b). * h,$$

which can be written using the matrix-vector multiplication as

$$\psi'(e).h = [2\,\mathrm{diag}(e)\,C\,\mathrm{diag}(e) + \mathrm{diag}(C(e. * e) - b)]\ h.$$

In other words, we have found the Jacobian matrix of ψ. Then a standard Newton scheme can be applied to solve $\psi(e) = 0$ as follows.

Newton scheme for any row $u = e. * e$ of U

Given e^0 such that $C(e^{(0)}. * e^{(0)}) - b \geq 0$, do for $\ell = 0, 1, \ldots$ until convergence:

1. Compute the actual residual $r^{(\ell)} = C(e^{(\ell)}. * e^{(\ell)}) - b$;
2. Solve for h from the linear system

$$\left[2\,\mathrm{diag}(e^{(\ell)})\,C\,\mathrm{diag}(e^{(\ell)}) + \mathrm{diag}(r^{(\ell)})\right]\ h = -r^{(\ell)}. * e^{(\ell)}; \tag{11}$$

3. Update $e^{(\ell+1)} = e^{(\ell)} + \alpha^{(\ell)} h$;
4. end

With $|e^{(0)}|$ large enough, the step size $\alpha^{(\ell)}$ is adapted so as to maintain $r^{(\ell)} \geq 0$ for all ℓ. The norm of residual, $\|r^{(\ell)}\|$, is checked for convergence. At the convergence, the row vector $u = e. * e$ is the nonnegative solution to (8) (notice that iterates $e^{(\ell)}$ may have also negative components!). All rows (m) of U can be computed in parallel. At the end, for a fixed V, we obtain a nonnegative solution U to (4), which also satisfies the inequality requirement (5). Exchanging roles of U and V, similar Newton scheme can be derived for each column of V independently when fixing U.

It should be emphasized that it is not clear at all under which conditions the outer loop iteration converges. Even if it converges, the Newton iteration only finds critical points satisfying the first order optimality conditions. In particular, without the second order information (Hessian), the iteration does not distinguish a minimizer from a maximizer. It is also possible that the outer iteration will converge to a saddle point.

2.2 Projected Newton iteration

The objective function in the NNMF problem formulation (1) is separable in columns. For a fixed $U \in \mathbb{R}^{m\times k}$, each single column of V is an object of a least squares (LS) minimization for an objective function

$$\phi(v) = \frac{1}{2}\|a - Uv\|_2^2, \text{ s.t. } v \geq 0, v \in \mathrm{R}^p), \tag{12}$$

where a is the corresponding column of A. Such a nonnegative least squares (NNLS) problem is well known in literature; see [6]. Also MATLAB has the function `LSQNONNEG` that can be used for finding v.

The next step consists in fixing V and formulating the nonnegative LS step for each row u^T of U. Hence, this is another outer loop iteration, similar to the previous one, but using the NNLS solver instead of the linear system one.

2.3 Multiplicative update algorithms

Multiplicative update algorithms (MUAs) use the so-called *reduced quadratic model approach*, where the idea is to replace the quadratic function $\phi(v)$ defined in (12) by a sequence of simpler quadratic functions. In more detail, near any given v^c, the quadratic function $\phi(v)$ can be expressed up to the first-order derivative (gradient) as

$$\phi(v) = \phi(v^c) + (v - v^c)^T \nabla\phi(v^c) + \frac{1}{2}(v - v^c)^T\, U^T U\, (v - v^c). \tag{13}$$

This expansion is approximated by a *simpler* quadratic model, where the matrix $U^T U$ is replaced by some *diagonal* matrix D that depends on v^c:

$$\tilde{\phi}(v; v^c) = \phi(v^c) + (v - v^c)^T \nabla\phi(v^c) + \frac{1}{2}(v - v^c)^T\, D(v^c)\, (v - v^c). \tag{14}$$

The minimizer of $\phi(v)$ is approximated by the minimizer v^+ of $\tilde{\phi}(v; v^c)$. The choice of $D(v^c)$ is specific for a given algorithm.

First MUA was designed by Lee and Seung in [7]. Their choice of diagonal elements d_i of $D(v^c)$ was as follows:

$$d_i := \frac{(U^T U v^c)_i}{v_i^c}, \quad 1 \leq i \leq k.$$

This specific choice has four important consequences. First, is can be shown [8] that

$$(v - v^c)^T\, (D(v^c) - U^T U)\, (v - v^c) \geq 0$$

for all v. Hence, the matrix $D(v^c) - U^T U$ is positive semidefinite, implying that $\phi(v) \leq \tilde{\phi}(v; v^c)$ for all v. Second, the minimum of any quadratic function always has a closed form solution, but for the diagonal $D(v^c$, the minimum v^+ of $\tilde{\phi}(v; v^c)$ is very simple:

$$v^+ := v^c - D^{-1}(v^c)\, (U^T U v^c - U^T a). \tag{15}$$

Third, it follows from the definition of $D(v^c$ that the entries of v^+ are given by

$$v_i^+ = \frac{(U^T a)_i}{(U^T U v^c)_i} v_i^c, \quad 1 \leq i \leq p.$$

Therefore, if v^c is nonnegative, the components of v^+ remain nonnegative. Finally,

$$\phi(v^+) \leq \tilde{\phi}(v^+; v^c) \leq \tilde{\phi}(v^c; v^c) = \phi(v^c),$$

which means that v^+ is an improved update of v^c.

Repeating the above process for each individual column of V and assembling all columns together, the updated matrix $V^+ = [v_{ij}^+]$ from a given nonnegative matrix $V^c = [v_{ij}^c]$ and a fixed nonnegative matrix U can be defined by the multiplicative rule:

$$v_{ij}^+ := \frac{(U^T A)_{ij}}{(U^T U V^c)_{ij}} v_{ij}^c. \tag{16}$$

By interchanging the roles of v and u (where u^T is a row of U), similar updates can be found for each row of U when fixing V. In summary, the updated matrix $U^+ = [u_{ij}^+]$ from a given nonnegative matrix $U^c = [u_{ij}^c]$ and a fixed nonnegative matrix V can be defined by another multiplicative rule:

$$u_{ij}^+ := \frac{(AV^T)_{ij}}{(U^c V V^T)_{ij}} u_{ij}^c. \tag{17}$$

Using the MATLAB notation, this algorithm is listed below.

Multiplicative Update Algorithm (MUA)

1. $U = \text{rand}(m, k)$; % initialize U as random dense matrix
2. $V = \text{rand}(k, n)$; % initialize V as random dense matrix
3. for $i = 1 : \text{maxiter}$
   $V = V. * (U^T A)./(U^T U V + 10^{-9})$;
   $U = U. * (AV^T)./(UVV^T + 10^{-9})$;
4. end

Steps 1 and 2 generate initial random matrices with nonnegative, random entries in interval $(0, 1)$. The constant 10^{-9} in both denominators prevents form dividing by zero.

The form of a multiplicative update in the above algorithm ensures that if the initial matrices U and V are strictly positive, then these matrices remain strictly positive throughout the iterations. Moreover, if the sequence of iterates (U, V) converges to $(U^\star, V^\star)$ and $U^\star > 0$, $V^\star > 0$, then $(\partial f/\partial U)((U^\star, V^\star) = 0$ and $(\partial f/\partial V)((U^\star, V^\star) = 0$, where $f(\cdot, \cdot)$ is the functional from (1). Unfortunately, this does *not* imply that $(U^\star, V^\star)$ is a local minimum since it can also bee a saddle point. The convergence of MUA was analyzed in detail in [4]. In summary, when the algorithm has converged to a limit point in the interior of the feasible region, this point is

a stationary point that may or may not be a local minimum. When the limit point lies on the boundary of the feasible region, its stationarity cannot be determined.

The computational experience has shown that when the MUA converges (which is often in practice), the convergence can be very slow. It requires many more iterations than alternatives discussed below and the work per iteration step is $O(mnk)$.

2.4 Gradient descent algorithms

Algorithms of this class repeatedly apply update rules of the form shown below.

Gradient Descent Algorithm (GDA)

1. $U = \text{rand}(m, k)$; % initialize U as random dense matrix
2. $V = \text{rand}(k, n)$; % initialize V as random dense matrix
3. for $i = 1 : \text{maxiter}$
   $V = V - \epsilon_V \frac{\partial f}{\partial V}$;
   $U = U - \epsilon_U \frac{\partial f}{\partial U}$;
4. end

Two parameters ϵ_U and ϵ_V depend on iteration step and partial derivatives of functional f are the same as in MUA. The crucial choice in this class of algorithms are the sizes of ϵ_V and ϵ_U. For example, one strategy chooses both parameters equal 1 at the beginning and then halves this value in each iteration. However, using this approach there is no guarantee that the elements of U and V remain nonnegative through the computation. Therefore, a common practice after an iteration step is to project both matrices U and V into the nonnegative orthant simply by setting all negative elements to zero.

The convergence analysis depends very much on the strategy of choosing both parameters ϵ_U and ϵ_V. Adding a projection step, the convergence analysis is even more difficult. Gradient descent methods that use a simple geometric rule for the stepsize (e.g., powering a fraction or scaling by a fraction at each iteration) can produce a poor factorization. Moreover, the method is very sensitive to the initialization of U and V, and the factorization can converge to matrices that are not far away from the initial guess.

2.5 Alternating least squares algorithms

Alternating Least Squares Algorithms (ALSA) are based on the observation that while the optimization problem (1) is not convex in both U and V, it is convex in either U or V. Thus, given one matrix, the other matrix can be found with a simple least squares (LS) computations.

Hence, in these algorithms, one LS step is followed by another LS step in alternating fashion (hence the name ALS). Basic algorithmic structure is listed below.

Alternating Least Squares Algorithms (ALSA)

1. $U = \text{rand}(m, k)$; % initialize U as random dense matrix
2. for $i = 1 : \text{maxiter}$
   Solve for V: $U^T U V = U^T A$; % first LS step
   Set all negative elements in V to 0; % projection
   Solve for U: $V V^T U^T = V A^T$; % second LS step
   Set all negative elements in U to 0; % projection
3. end

The algorithm is based on the solution of normal matrix equations for U and V. When a new V is found (updated) from the first system, the matrix VV^T is computed serving as a system matrix in the second system. When U is computed from the second system, $U^T U$ must be formed for the first system, and so on. Simple projection technique used in ALSA allows the iterates some additional flexibility not available in other classes of NNMF algorithms. For example, in MUA, once an element in U or V becomes 0, it *must* remain 0 for all iterations. This "locking" of zeros is restrictive, because once the algorithm starts heading down a path towards a "poor" fixed point, it *must* continue in that way. ALSAs are more flexible, allowing the iterative process to escape from a poor path.

ALSAs can be very fast. The implementation shown above requires slightly less work than an SVD implementation. Possible improvements incorporate sparsity and nonnegativity constraints described in next section.

With respect to the convergence, ALSA is actually a well-known simple optimization technique under various names in statistics: alternating variables, coordinate search, or method of local variation [9]. When the nonnegativity of both factors is properly enforced, for example, by the nonnegative least squares (NNLS) algorithm of Lawson and Hanson [6], then ALSA will converge to a local minimum. However, solving nonnegatively constrained least squares problems rather than unconstrained ones at each iteration greatly increases cost per iteration step.

2.6 General comments

For an NNMF algorithm of any class, one should input the fixed point solution into optimality condition to determine if it is indeed a minimum. In fact, the NNMF problem (1) does not have a unique global minimum due to a possible scaling and permutations. Therefore, some algorithms enforce row or column normalizations at each iteration. One can also use several different initializations using, e.g., the Monte Carlo method, and run them in parallel.

The rate of convergence of these algorithms is an open research problem. There exist only some bounds of rates in very special circumstances.

The quality of an approximate solution of (1) can be measured against the optimal rank-k reduction $U_k\Sigma_k V_k$ provided by the partial SVD of matrix A. Ideally, for a given NNMF algorithm with solution $(U^\star, V^\star)$ one would like to prove something like

$$1 - \epsilon \leq \frac{\|A - U^\star V^\star\|}{\|A - U_k\Sigma_k V_k\|} \leq 1$$

for some sufficiently small constant ϵ that depends on the parameters of the particular NNMF algorithm. Only a small number of results are known in this field.

To finish iterations, the squared Frobenius norm $\|A - UV\|_{\mathrm{F}}^2$ must be computed either in each iteration or periodically and compared with tol, $0 < \mathrm{tol} \ll 1$. Hence, an efficient computation of $\|A - UV\|_{\mathrm{F}}^2$ is desirable. The following expression

$$\|A - UV\|_{\mathrm{F}}^2 = \mathrm{trace}(A^T A) - 2\,\mathrm{trace}(V^T(U^T A)) + \mathrm{trace}(V^T(U^T UV))$$

contains an efficient order of matrix multiplications and also allows for computing the term $\mathrm{trace}(A^T A) = \|A\|_{\mathrm{F}}^2$ only once. Further, matrix products $U^T A$ and $U^T UV$ are usually needed in the updating step so that they can be re-used 'for free' in checking the convergence criterion. As a result, the complexity of this residual computation can be reduced to $O(nk)$. As a secondary convergence criterion, a maximum number of iterations is usually used, but this parameter is very problem-dependent.

3 Application-dependent auxiliary constraints

The NNMF problem formulation (1) can be sometimes extended to include auxiliary constraints on U and/or V. The purpose is often to compensate for uncertainties in the data, to enforce desired characteristics of a solution or to impose some *a priori* knowledge about the application at hand. *Penalty terms* are typically used, extending the cost function of Eq. (1) to the form

$$f(U, V) = \frac{1}{2}\|A - UV\|_{\mathrm{F}}^2 + \alpha J_1(U) + \beta J_2(V). \tag{18}$$

Here, $J_1(U)$ and $J_2(V)$ are the penalty terms depending on application, and α, β are small regularization parameters that balance the trade-off between the approximation error and the constraints.

Smoothness constraints are quite often enforced to somehow *regularize* the computed solution in the presence of noise in the data. For example, the term

$$J_1(U) = \|U\|_{\mathrm{F}}^2$$

penalizes matrices U of large Frobenius norm, i.e., it implicitly requires the norms of columns (rows) being 'small'. In practice, the columns of U are often normalized in order to maintain U away from zero. This form of regularization is known as *Tikhonov regularization* in inverse problems. More general form of regularization is defined by $J_1(U) = \|LU\|_{\mathrm{F}}^2$, where L is a regularization operator (e.g., the Laplacian operator). Smoothness constraints can be applied also to V depending on an application.

Sparsity constraints on either U or V can be similarly imposed. The notion of sparsity refers to the fact (or desire) that the data is well represented only by a limited number of different features. Measures for sparsity include, for example, the ℓ^p-norms for $0 < p \leq 1$ and Hoyer's measure [5] for vector x of dimension n,

$$\text{sparseness}(x) = \frac{\sqrt{n} - \|x\|_1/\|x\|_2}{\sqrt{n} - 1}. \tag{19}$$

Recall that $\|x\|_1 = \max_{1 \leq i \leq n}(|x_i|)$ and $\|x\|_2$ is the Euclidean norm of x. The Hoyer measure can be imposed as a penalty term in the form

$$J_2(V) = (\omega \|\text{vec}(V)\|_2 - \|\text{vec}(V)\|_1)^2,$$

where $\omega = \sqrt{kn} - (\sqrt{kn} - 1)\gamma$ and $\text{vec}(\cdot)$ is the operator that transforms a matrix into a vector by stacking its columns. The desired sparseness in H is specified by setting γ to a value between 0 and 1.

In some applications, the constraints must be met ensuring that the solution is physically realizable. One such physical constraint requires mixing coefficients v_{ij} to sum to one, i.e., $\sum_i v_{ij} = 1$ for all j. Enforcing such a physical constraint can significantly improve the determination of inherent features, when the data are in fact linear combinations of these features [10]. This additivity condition can be formulated as the penalty term

$$J_2(H) = \|V^T e_1 - e_2\|_2^2,$$

where e_1 and e_2 are vectors of ones of dimension k and n, respectively. This is equivalent to condition that V is a column-stochastic matrix, or that the minimization of (18) seeks solutions U whose columns form a convex set containing the data vectors in A.

When the constraints are included in form of penalty terms $J_1(U)$ and $J_2(V)$, the update rules for gradient descent method will change. Assuming that $J_1(U)$ and $J_2(V)$ have partial derivatives with respect to u_{ij} and v_{ij}, respectively, the update rules can be formulated as

$$\begin{aligned} u_{ij}^{(t)} &= u_{ij}^{(t-1)} \frac{(AV^T)_{ij}}{(U^{(t-1)}VV^T)_{ij} + \alpha(\partial J_1(U)/\partial u_{ij})}, \\ v_{ij}^{(t)} &= v_{ij}^{(t-1)} \frac{(U^T A)_{ij}}{(U^T U V^{(t-1)})_{ij} + \beta(\partial J_2(V)/\partial v_{ij})}. \end{aligned} \tag{20}$$

The extended cost function is non-increasing with these update rules for sufficiently small values of α and β; see [10]. For example, when enforcing the smoothness of both W and H, the updating rules in MUA become:

$$\begin{aligned} V &= V. * (U^T A)./(U^T U V + \beta V + 10^{-9}), \\ U &= U. * (AV^T)./(UVV^T + \alpha U + 10^{-9}). \end{aligned}$$

4 Parallelization strategy

A suitable candidate for parallelization seems to be the ADI Newton iteration method, for which we present a parallelization strategy for a distributed memory system of processors (e.g, a cluster) using the Message Passing Interface.

The main processing module, which is run in each processor, is depicted as Algorithm 1 (A1). Here it is assumed that we have p processors. The original matrix A of order $m \times n$ is cut

Algorithm 1 Parallel ADI Newton iteration algorithm

```
procedure ADI-Newton ( m, n, k, mp, np, Arow, Acol, Urow, Vcol )
    repeat
      ParMatMul ( 'FALSE', m, k, np, Acol, Vcol, B )        ▷ B := AV^T
      ParMatMul ( 'FALSE', k, k, np, Vcol, Vcol, C )        ▷ C := VV^T
      NewtonForU ( m, k, mp, B, C, Urow )
      ParMatMul ( 'TRUE', k, n, mp, Urow, Arow, R )         ▷ R := U^T A
      ParMatMul ( 'TRUE', k, k, mp, Urow, Urow, S )         ▷ S := U^T U
      NewtonForV ( k, n, np, R, S, Vcol )
    until Convergence (m, n, k, mp, np, Acol, Urow, Vcol)
end procedure
```

row-wise and column-wise into p block rows and block columns of size $m/p \times n$ and $m \times n/p$, respectively. Similarly, the computed matrix U of order $m \times k$ is cut row-wise and column-wise into p block rows and block columns of size $m/p \times k$ and $m \times k/p$, respectively. However, the computed matrix V of order $k \times n$ is cut only column-wise into p block columns of order $k \times n/p$. Notice that the number of processors p should be chosen less than k. Each processor then contains one block row and one block column of A, denoted by *Arow* and *Acol*, respectively, one block row and one block column of U denoted by *Urow* and *Ucol*, respectively, and one block column of V denoted *Vcol*. All matrix blocks in each processor remain stationary during computation.

This data distribution enables to compute in parallel matrix products needed in the outer loop of A1. Parallel matrix multiplication procedure `ParMatMul` is called four times in each outer iteration step and its structure is depicted in Algorithm 2 (A2). The usage of a transposed

Algorithm 2 Parallel Matrix Multiplication

```
procedure ParMatMul ( TRANSA, m, n, k, A, B, C )
    if TRANSA then                          ▷ local serial matrix multiplication
        D ← A^T * B
    else
        D ← A * B^T
    end if
    ALLREDUCE(D,C,m*n,SUM)                  ▷ global sum over all procs
end procedure
```

matrix block (i.e., block column) is controlled by the logical variable *TANSA*. Two matrix blocks are multiplied locally in each processor and the local results are then collected globally into matrix C, which is stored in each processor. This approach excludes communications between processors except one global reduction at the end of matrix multiplication.

After computing matrices $B \in \mathbb{R}^{m \times k}$ and $C \in \mathbb{R}^{k \times k}$ and storing them in each processor, the outer loop in A1 continues by calling the function `NewtonForU` (A3), which updates in parallel the rows of *Urow* in each processor. Locally, the computation proceeds serially for *mp* rows of

Algorithm 3 Newton scheme for local rows of U

procedure NEWTONFORU ($m, k, mp, B, C, Urow$)
  for $i = 1, mp$ **do**
    $C(e. * e) - B_{offset+i} \geq 0$ ▷ find such e
    $r = C(e. * e) - B_{offset+i}$ ▷ compute the residual
    while $\|r\| > \epsilon$ **do**
      $[2\,\mathrm{diag}(e)\,C\,\mathrm{diag}(e) + \mathrm{diag}(r)]\,h = -r. * e$ ▷ solve for h
      $e \leftarrow e + \alpha\, h$ ▷ update of e, α is the step size
      $r = C(e. * e) - B_{offset+i}$ ▷ compute the residual
    end while
    $Urow_i = e. * e$
  end for
end procedure

Urow and implements the Newton scheme, which requires the solution of one small, symmetric linear system of size k in each iteration step. Since $k \ll m, n$, it is possible to use some direct method (e.g., the Gaussian elimination with partial pivoting) for solving this linear system. As a whole, F2 is an inner iteration loop (run serially for each row of *Urow*), where the convergence criterion is based on the residual $r = C(e. * e) - B_{\mathrm{offset}+i}$. Notice that the inner iterations do not require any communication between processors.

When the upadte of U is done, very similar computations are performed for columns of V. To finish the outer loop, one has to check the convergence citerion formulated in the function `Convergence` (A4). First, the whole matrix U of order $m \times k$ is collected in each processor

Algorithm 4 Test of convergence

function CONVERGENCE $(m, n, k, mp, np, Ac, Urow, Vcol)$
  ALLGATHER(Urow,mp*k,U) ▷ collection of distributed U
  $localnorm \leftarrow \|Acol - U * Vcol\|_{\mathrm{F}}^2$
  ALLREDUCE(localnorm,f,1,SUM)
  return $\frac{1}{2}f$
end function

using the global communication `ALLGATHER`.Then, the square of Frobenius norm of the local residual $(Acol - U * Vcol)$ is computed, summed-up over all PEs and stored in each PE. The computation in all PEs finishes when $f < \epsilon$ for some $\epsilon \ll 1$.

5 Conclusion

We have presented basic algorithms for the nonnegative matrix factorization. This type of matrix compression becomes increasingly popular with researchers in many field of science and engineering, because many applications can be modeled by a non-negative linear combination of non-negative features. Then we know *a priori* that the NNMF is more suitable for data analysis and presentation than the SVD. We suggested also a general approach for a parallelization strategy using the Newton ADI method as an example.

References

[1] Berry M. W., Gillis N., Glineur F., Document classification using nonnegative matrix factorization and underapproximation, IEEE Press, 2009.

[2] Berry M. W., Browne M., Langville A. N., Pauca V. P., Plemmons R. J., Algorithms and applications for approximate nonnegative matrix factorization, *C*omputational Statistics and Data Analysis 52 (2007) 155-173.

[3] Chu M., Diele F., Plemmons R., Ragni S., Optimality, computation and interpretation of nonnegative matrix factorizations, preprint.

[4] Finesso L., Spreij P., Approximate nonnegative matrix factorization via alternating minimization, In: *Proc. 16th International Symposium on Mathematical Theory of Networks and Systems*, July 5-9, 2004, Leuven, Belgium.

[5] Hoyer P., Non-negative matrix factorization with sparseness constraints, *J. Mach. Learning Res.*, 5 (2004) 1457-1469.

[6] Lawson C., Hanson R., *Solving Least Squares Problems*, 1995, SIAM, Philadelphia.

[7] Lee D., Seung H., Learning the parts of objects by nonnegative matrix factorization, *Nature* 401 (1999) 788-791.

[8] Lee D., Seung H., Algorithms for non-negative matrix factorization, *Adv. Neural Inform. Process. Systems* 13 (2001) 556-562.

[9] Nocedal J., Wright S., *Numerical Optimization*, 1999, Springer Verlag, Berlin.

[10] Pauca V., Piper, J., Plemmons R., Nonnegative matrix factorization for spectral data analysis, *Lin. Algebra Appl.* 416 (2006) 29-47.

[11] Shahnaz F., Berry M. W., Pauca V. P., Plemmons R. J., Document clustering using nonnegative matrix factorization, *Information Processing and Management* 42 (2006) 373-386.