Nonlinear Least Squares Updating of the Canonical Polyadic Decomposition

Michiel Vandecappelle, Nico Vervliet, and Lieven De Lathauwer

Abstract—**Current batch tensor methods often struggle to keep up with fast-arriving data. Even storing the full tensors that have to be decomposed can be problematic. To alleviate these limitations, tensor updating methods modify a tensor decomposition using efficient updates instead of recomputing the entire decomposition when new data becomes available. In this paper, the structure of the decomposition is exploited to achieve fast updates for the canonical polyadic decomposition whenever new slices are added to the tensor in a certain mode. A batch NLS-algorithm is adapted so that it can be used in an updating context. By only storing the old decomposition and the new slice of the tensor, the algorithm is both time- and memory efficient. Experimental results show that the proposed method is faster than batch ALS and NLS methods, while maintaining a good accuracy for the decomposition.**

I. Introduction

TENSOR decompositions are powerful tools for various applications in machine learning and signal processing [1]–[3]. Tensors are higher-order extensions of vectors and matrices. They allow one to store and analyze large and higher-order datasets with the use of compact and meaningful tensor decompositions. As such, tensor tools are promising for big data applications. Several algebraic and optimization-based algorithms have been developed for tensor decompositions: See for instance [4]–[6]. Recently, dedicated methods have been designed for large, sparse, or incomplete tensors: See for example [7]–[9] and references therein. Also, support for structured and coupled tensor decompositions has been added to tensor toolboxes such as Tensorlab [10], [11].

Mainly batch methods are used to handle higher-order data, but they make one very important assumption: they assume that the full tensor is available at the start and that it does not change afterwards. As a result, these methods always compute a decomposition for the whole tensor. Yet, in real-time applications, tensors do not have to be immutable: the tensor entries might change gradually (or abruptly) over time or the tensor may grow (or shrink) in one or more modes. A tensor might even be so large that a decomposition for the full tensor cannot be computed at once, but must be built up from the decomposition of its subtensors, e.g., slice by slice. The available time to compute a decomposition may also be limited. In such cases, one would like to update the tensor decomposition using only the data that has changed. Such efficient updating methods for tensor decompositions are currently being investigated [12]–[14]. While the decomposition might lose some accuracy, the speed and memory efficiency of updating methods may give them an advantage in practice.

In this paper, we exploit the structure of the Canonical Polyadic Decomposition (CPD) of a tensor to obtain a Nonlinear Least Squares (NLS) updating algorithm. We apply the framework for the computation of structured tensor decompositions proposed in [15], [16] to modify the CPD when new tensor slices become available and old slices become outdated. This yields a CPD updating method that is more efficient than the existing batch methods, while maintaining a good accuracy. Additionally, the method only has to store the old decomposition and the new tensor slice in every updating step, thus making it both time- and memory-efficient. The method also admits arbitrary windowing strategies and can be used for tensors that have a dynamic tensor rank.

We fix notation and give basic definitions in Section II. We derive our method in Section III, discuss numerical experiments in Section IV and conclude in Section V.

II. Notation and definitions

Scalars, vectors and matrices are denoted by lowercase (a), bold lowercase ($\mathbf{a}$) and bold uppercase letters ($\mathbf{A}$), respectively. We refer to tensors by using letters in calligraphic script ($\mathcal{T}$). An Nth order tensor has N different modes. The outer product of N vectors, denoted by $\mathbf{v}^{(1)} \otimes \cdots \otimes \mathbf{v}^{(N)}$, is a natural extension of the outer product of two vectors. The result is an Nth-order tensor $\mathcal{T}$ of which each entry is defined as follows: $t_{i_1 \ldots i_N} = v^{(1)}_{i_1} v^{(2)}_{i_2} \cdots v^{(N)}_{i_N}$. For simplicity of notation, third-order tensors will be used throughout the rest of the paper. A mode-n fiber $\mathbf{t}_{ij:}$, $\mathbf{t}_{i:k}$ or $\mathbf{t}_{:jk}$ of a tensor $\mathcal{T} \in \mathbb{R}^{I\times J\times K}$ is a vector obtained by fixing all indices but the nth. Likewise, a mode-(m,n) slice $\mathbf{T}_{i::}$, $\mathbf{T}_{:j:}$ or $\mathbf{T}_{::k}$ is a matrix obtained by fixing all but the mth and nth index. The Frobenius norm is denoted by $||.||_F$ and the Kronecker, Khatri–Rao and Hadamard products of matrices by $\otimes$, $\odot$ and $*$, respectively, where

$$\mathbf{A} \otimes \mathbf{B} = \begin{bmatrix} a_{11}\mathbf{B} & a_{12}\mathbf{B} & \cdots \\ a_{21}\mathbf{B} & a_{22}\mathbf{B} & \cdots \\ \vdots & \vdots & \ddots \end{bmatrix}, \mathbf{A} \odot \mathbf{B} = \left[\mathbf{a}_1 \otimes \mathbf{b}_1, \ldots, \mathbf{a}_R \otimes \mathbf{b}_R \right],$$

Funding: Michiel Vandecappelle is supported by an SB Grant from the Research Foundation – Flanders (FWO) and Nico Vervliet is supported by an Aspirant Grant from the Research Foundation – Flanders (FWO). Research furthermore supported by: (1) Flemish Government: FWO: projects: G.0830.14N, G.0881.14N; (2) EU: The research leading to these results has received funding from the European Research Council under the European Union's Seventh Framework Programme (FP7/2007-2013) / ERC Advanced Grant: BIOTENSORS (n° 339804). This paper reflects only the authors' views and the Union is not liable for any use that may be made of the contained information; (3) KU Leuven Internal Funds C16/15/059.

KU Leuven, Dept. of Electrical Engineering ESAT/STADIUS, Kasteelpark Arenberg 10, bus 2446, B-3001 Leuven, Belgium; and Group Science, Engineering and Technology, KU Leuven - Kulak, E. Sabbelaan 53, 8500 Kortrijk, Belgium (Michiel.Vandecappelle@kuleuven.be, Nico.Vervliet@kuleuven.be, Lieven.DeLathauwer@kuleuven.be).

and $(\mathbf{A} * \mathbf{B})_{ij} = a_{ij}b_{ij}$. Herein $\mathbf{A} = [\mathbf{a}_1 \cdots \mathbf{a}_R]$ and $\mathbf{B} = [\mathbf{b}_1 \cdots \mathbf{b}_R]$. The mode-$n$ product of a tensor and a matrix is denoted by $\cdot_n$ and defined as $(\mathcal{T} \cdot_n \mathbf{G})_{i_1,\ldots,i_{n-1}j i_{n+1} i_N} = \sum_n t_{i_1,\ldots,i_N} g_{j i_n}$, where the nth dimension of $\mathcal{T}$ is equal to the number of columns of $\mathbf{G}$. The transpose of a matrix $\mathbf{M}$ is written as $\mathbf{M}^{\mathrm{T}}$ and its Moore–Penrose pseudoinverse as $\mathbf{M}^{\dagger}$. $\mathrm{vec}(\mathbf{X})$ denotes the vectorization of the matrix $\mathbf{X}$, i.e., putting the columns of $\mathbf{X}$ below each other and $\mathrm{diag}(\mathbf{x})$ forms a square matrix that has $\mathbf{x}$ as its diagonal.

A third-order tensor has rank 1 if it is the outer product of three non-zero vectors. The rank of a tensor is the minimal number of terms that is needed to write the tensor as a linear combination of rank-1 tensors. This leads to the CPD of a tensor, which decomposes a rank-R tensor $\mathcal{T}$ as a linear combination of R rank-1 terms:

$$\mathcal{T} = \sum_{r=1}^{R} \mathbf{a}_r \otimes \mathbf{b}_r \otimes \mathbf{c}_r.$$

The vectors $\mathbf{a}_r$, $\mathbf{b}_r$ and $\mathbf{c}_r$ are usually collected into factor matrices $\mathbf{A}$, $\mathbf{B}$ and $\mathbf{C}$, as follows: $\mathbf{A} = [\mathbf{a}_1 \ldots \mathbf{a}_R]$, and similarly for $\mathbf{B}$ and $\mathbf{C}$. The CPD is written as $\mathcal{T} = [\![\mathbf{A}, \mathbf{B}, \mathbf{C}]\!]_R$, or as $\mathcal{T} = [\![\mathbf{A}, \mathbf{B}, \mathbf{C}]\!]$ if the rank R is clear from the context. If the tensor $\mathcal{T}$ is unfolded to a matrix in the third mode by combining all its mode-3 fibers as columns of a matrix $\mathbf{T}_{(3)}$, the CPD can also be written as $\mathbf{T}_{(3)} = \mathbf{C}(\mathbf{B} \odot \mathbf{A})^{\mathrm{T}}$ [5].

III. NLS UPDATING

NLS algorithms have been shown to perform well for the computation of the CPD of a tensor, as they are efficient and very robust for more difficult decompositions [5]. In this section, we convert the batch NLS algorithm to an updating method that maintains most of these nice properties, while being more time- and memory-efficient.

Consider at time k a third order tensor $\mathcal{T} \in \mathbb{R}^{I \times J \times K}$, with rank-$R$ CPD $[\![\mathbf{X}, \mathbf{Y}, \mathbf{Z}]\!]$. At time $k+1$, a frontal slice $\mathbf{M}$ is added to $\mathcal{T}$ in the third mode, as shown in Figure 1, forming a tensor $\mathcal{T}^{(\mathrm{up})}$. Instead of recomputing the entire CPD to include the new slice, we want to perform an efficient update of the decomposition, both in time and memory, to obtain the CPD $[\![\mathbf{A}, \mathbf{B}, \mathbf{C}]\!]$ of $\mathcal{T}^{(\mathrm{up})}$. Assuming the model stays approximately the same, the old factor matrices are used to initialize the algorithm, but $\mathbf{Z}$ is extended with a new row $\mathbf{c}_{\mathrm{new}}^{\mathrm{T}}$: thus $\mathbf{A} = \mathbf{X}$, $\mathbf{B} = \mathbf{Y}$ and $\mathbf{C} = \begin{bmatrix} \mathbf{Z} \\ \mathbf{c}_{\mathrm{new}}^{\mathrm{T}} \end{bmatrix}$. $\mathbf{c}_{\mathrm{new}}$ can be obtained from the new slice by computing the least squares solution of $(\mathbf{Y} \odot \mathbf{X})\mathbf{c}_{\mathrm{new}} = \mathrm{vec}(\mathbf{M})$:

$$\begin{aligned} \mathbf{c}_{\mathrm{new}} &= (\mathbf{Y} \odot \mathbf{X})^{\dagger} \mathrm{vec}(\mathbf{M}) \\ &= \left[(\mathbf{Y} \odot \mathbf{X})^{\mathrm{T}} (\mathbf{Y} \odot \mathbf{X})\right]^{-1} (\mathbf{Y} \odot \mathbf{X})^{\mathrm{T}} \mathrm{vec}(\mathbf{M}) \\ &= [(\mathbf{Y}^{\mathrm{T}}\mathbf{Y}) * (\mathbf{X}^{\mathrm{T}}\mathbf{X})]^{-1} (\mathbf{Y} \odot \mathbf{X})^{\mathrm{T}} \mathrm{vec}(\mathbf{M}). \end{aligned} \quad (1)$$

As R is typically small, computing the inverse of the $R \times R$-matrix $[(\mathbf{Y}^{\mathrm{T}}\mathbf{Y}) * (\mathbf{X}^{\mathrm{T}}\mathbf{X})]$ is not expensive (and singularity of this matrix points to an overestimation of R), while $(\mathbf{Y} \odot \mathbf{X})^{\mathrm{T}} \mathrm{vec}(\mathbf{M})$ can be obtained without explicitly forming the Khatri–Rao product [17].

In the remainder of this section, we describe an NLS method that can efficiently update an existing CPD when a new slice is added to the tensor, while only the factor matrices and the new tensor slice are stored. The method builds on the framework for the efficient representation of structured tensors described in [15], [16]. We exploit the structure of the CPD to derive efficient expressions for the objective function, gradient and Gramians that are needed in the NLS method. Windowing strategies are also discussed, as are dynamic tensor ranks. The section ends with an analysis of the complexity of the algorithm. The full algorithm is given in Algorithm 1.

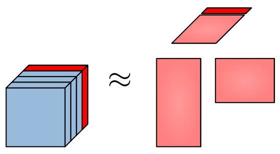
Figure 1. Example of the tensor updating procedure. Left: the original tensor is extended with an extra slice (red) in the third mode. Right: the CPD of the tensor is updated by adding a new vector (red) to the factor matrix in the third mode and modifying the existing factor matrices (pink).

A. Objective function

We compute the CPD $[\![\mathbf{A}, \mathbf{B}, \mathbf{C}]\!]$ of the updated tensor $\mathcal{T}^{(\mathrm{up})}$ by minimizing the following objective function:

$$\min_{\mathbf{A},\mathbf{B},\mathbf{C}} f = \min_{\mathbf{A},\mathbf{B},\mathbf{C}} \frac{1}{2} \| [\![\mathbf{A}, \mathbf{B}, \mathbf{C}]\!] - \mathcal{T}^{(\mathrm{up})} \|_F^2.$$

We partition the factor matrix $\mathbf{C}$ as $\begin{bmatrix} \overline{\mathbf{C}} \\ \underline{\mathbf{c}} \end{bmatrix}$, with $\underline{\mathbf{c}}$ the last row of $\mathbf{C}$ and rewrite f as

$$f = \frac{1}{2} \| [\![\mathbf{A}, \mathbf{B}, \overline{\mathbf{C}}]\!] - \mathcal{T} \|_F^2 + \frac{1}{2} \| [\![\mathbf{A}, \mathbf{B}, \underline{\mathbf{c}}]\!] - \mathbf{M} \|_F^2,$$

which can be expanded to

$$\begin{aligned} f = &\frac{1}{2} \| [\![\mathbf{A}, \mathbf{B}, \overline{\mathbf{C}}]\!] \|_F^2 - \left\langle [\![\mathbf{A}, \mathbf{B}, \overline{\mathbf{C}}]\!], \mathcal{T} \right\rangle + \frac{1}{2} \|\mathcal{T}\|_F^2 \\ &+ \frac{1}{2} \| [\![\mathbf{A}, \mathbf{B}, \underline{\mathbf{c}}]\!] \|_F^2 - \langle [\![\mathbf{A}, \mathbf{B}, \underline{\mathbf{c}}]\!], \mathbf{M} \rangle + \frac{1}{2} \|\mathbf{M}\|_F^2. \end{aligned}$$

One can note that $\frac{1}{2} \| [\![\mathbf{A}, \mathbf{B}, \overline{\mathbf{C}}]\!] \|_F^2 + \frac{1}{2} \| [\![\mathbf{A}, \mathbf{B}, \underline{\mathbf{c}}]\!] \|_F^2 = \frac{1}{2} \| [\![\mathbf{A}, \mathbf{B}, \mathbf{C}]\!] \|_F^2$. The full tensor $\mathcal{T}$ is not stored during the updating process for memory efficiency. Instead, we work with its CPD approximation $[\![\mathbf{X}, \mathbf{Y}, \mathbf{Z}]\!]$, which is the best available guess of $\mathcal{T}$. Using this CPD approximation, we obtain the following objective function:

$$\begin{aligned} f \approx &\frac{1}{2} \| [\![\mathbf{A}, \mathbf{B}, \mathbf{C}]\!] \|_F^2 - \left\langle [\![\mathbf{A}, \mathbf{B}, \overline{\mathbf{C}}]\!], [\![\mathbf{X}, \mathbf{Y}, \mathbf{Z}]\!] \right\rangle \\ &+ \frac{1}{2} \| [\![\mathbf{X}, \mathbf{Y}, \mathbf{Z}]\!] \|_F^2 - \langle [\![\mathbf{A}, \mathbf{B}, \underline{\mathbf{c}}]\!], \mathbf{M} \rangle + \frac{1}{2} \|\mathbf{M}\|_F^2. \end{aligned} \quad (2)$$

Equation (2) can be further simplified to avoid the construction of full tensors by exploiting the structure of the CPD. First,

$$\begin{aligned} \| [\![\mathbf{A}, \mathbf{B}, \mathbf{C}]\!] \|_F^2 &= \mathrm{vec}([\![\mathbf{A}, \mathbf{B}, \mathbf{C}]\!])^{\mathrm{T}} \mathrm{vec}([\![\mathbf{A}, \mathbf{B}, \mathbf{C}]\!]) \\ &= \mathbf{1}^{\mathrm{T}} (\mathbf{C} \odot \mathbf{B} \odot \mathbf{A})^{\mathrm{T}} (\mathbf{C} \odot \mathbf{B} \odot \mathbf{A}) \mathbf{1} \\ &= \mathbf{1}^{\mathrm{T}} [(\mathbf{A}^{\mathrm{T}}\mathbf{A}) * (\mathbf{B}^{\mathrm{T}}\mathbf{B}) * (\mathbf{C}^{\mathrm{T}}\mathbf{C})] \mathbf{1}, \end{aligned} \quad (3)$$

where $\mathbf{1}$ is a vector of length R consisting of only ones. The term $\frac{1}{2} \| [\![\mathbf{X}, \mathbf{Y}, \mathbf{Z}]\!] \|_F^2$ can be simplified in the same manner. The inner product $\left\langle [\![\mathbf{A}, \mathbf{B}, \overline{\mathbf{C}}]\!], [\![\mathbf{X}, \mathbf{Y}, \mathbf{Z}]\!] \right\rangle$ can similarly be written as $\mathbf{1}^{\mathrm{T}}[(\mathbf{A}^{\mathrm{T}}\mathbf{X})*(\mathbf{B}^{\mathrm{T}}\mathbf{Y})*(\overline{\mathbf{C}}^{\mathrm{T}}\mathbf{Z})]\mathbf{1}$. Because $[\![\mathbf{A},\mathbf{B},\underline{\mathbf{c}}]\!] = \mathbf{A}\mathrm{diag}(\underline{\mathbf{c}})\mathbf{B}^T$, both $[\![\mathbf{A},\mathbf{B},\underline{\mathbf{c}}]\!]$ and $\mathbf{M}$ are matrices and their inner product is simply $\mathbf{1}^{\mathrm{T}}[(\mathbf{A}\mathrm{diag}(\underline{\mathbf{c}})\mathbf{B}^T) * \mathbf{M}]\mathbf{1}$.

B. Gradient and Gramian

For an NLS algorithm, efficient evaluations of the gradient and Gramian of f are required [16]. Below, we give the derivation of the gradient $\nabla f = \mathrm{vec}\left(\left[\frac{\partial f}{\partial \mathbf{A}}\ \frac{\partial f}{\partial \mathbf{B}}\ \frac{\partial f}{\partial \mathbf{C}}\right]\right)$. The expression for the Gramian is identical to the batch algorithm, as shown in [5].

We derive the terms $\frac{\partial f}{\partial \mathbf{A}}$ and $\frac{\partial f}{\partial \mathbf{C}}$ of the gradient. The term $\frac{\partial f}{\partial \mathbf{B}}$ is obtained analogously to $\frac{\partial f}{\partial \mathbf{A}}$. Only the first, second and fourth terms of equation (2) are non-constant, so we compute their derivatives with respect to $\mathbf{A}$ and $\mathbf{C}$, using the expressions from the previous subsection. For $\frac{\partial f}{\partial \mathbf{A}}$, we find

$$\frac{\partial}{\partial \mathbf{A}}\frac{1}{2}||\,[\![\mathbf{A},\mathbf{B},\mathbf{C}]\!]\,||_F^2 = \mathbf{A}[(\mathbf{B}^{\mathrm{T}}\mathbf{B}) * (\mathbf{C}^{\mathrm{T}}\mathbf{C})]$$
$$-\frac{\partial}{\partial \mathbf{A}}\left\langle [\![\mathbf{A},\mathbf{B},\overline{\mathbf{C}}]\!], [\![\mathbf{X},\mathbf{Y},\mathbf{Z}]\!]\right\rangle = -\mathbf{X}[(\mathbf{Y}^{\mathrm{T}}\mathbf{B}) * (\mathbf{Z}^{\mathrm{T}}\overline{\mathbf{C}})]$$
$$-\frac{\partial}{\partial \mathbf{A}}\left\langle [\![\mathbf{A},\mathbf{B},\underline{\mathbf{c}}]\!], \mathbf{M}\right\rangle = -\mathbf{M}\mathbf{B}\mathrm{diag}(\underline{\mathbf{c}}),$$

leading to

$$\frac{\partial f}{\partial \mathbf{A}} = \mathbf{A}[(\mathbf{B}^{\mathrm{T}}\mathbf{B})*(\mathbf{C}^{\mathrm{T}}\mathbf{C})] - \mathbf{X}[(\mathbf{Y}^{\mathrm{T}}\mathbf{B})*(\mathbf{Z}^{\mathrm{T}}\overline{\mathbf{C}})] - \mathbf{M}\mathbf{B}\mathrm{diag}(\underline{\mathbf{c}}).$$

Analogously, we have

$$\begin{aligned}\frac{\partial f}{\partial \mathbf{C}} &= \mathbf{C}[(\mathbf{A}^{\mathrm{T}}\mathbf{A}) * (\mathbf{B}^{\mathrm{T}}\mathbf{B})] + \begin{bmatrix} -\frac{\partial f}{\partial \overline{\mathbf{C}}}\langle [\![\mathbf{A},\mathbf{B},\overline{\mathbf{C}}]\!], [\![\mathbf{X},\mathbf{Y},\mathbf{Z}]\!]\rangle \\ -\frac{\partial f}{\partial \underline{\mathbf{c}}}\langle [\![\mathbf{A},\mathbf{B},\underline{\mathbf{c}}]\!], \mathbf{M}\rangle \end{bmatrix} \\ &= \mathbf{C}[(\mathbf{A}^{\mathrm{T}}\mathbf{A}) * (\mathbf{B}^{\mathrm{T}}\mathbf{B})] + \begin{bmatrix} -\mathbf{Z}[(\mathbf{X}^{\mathrm{T}}\mathbf{A})*(\mathbf{Y}^{\mathrm{T}}\mathbf{B})] \\ -\mathrm{vec}(\mathbf{M})^{\mathrm{T}}(\mathbf{B}\odot\mathbf{A}) \end{bmatrix},\end{aligned}$$

as $\frac{\partial f}{\partial \mathbf{C}}$ can be partitioned into $\begin{bmatrix}\frac{\partial f}{\partial \overline{\mathbf{C}}} \\ \frac{\partial f}{\partial \underline{\mathbf{c}}}\end{bmatrix}$ and the second and fourth term of Equation (2) do not depend on $\underline{\mathbf{c}}$ or $\overline{\mathbf{C}}$, respectively.

In the Gauss–Newton (GN) method, the Hessian of f is approximated by its Gramian $\mathbf{J}^T\mathbf{J}$, where $\mathbf{J}$ is the Jacobian of f. Using a limited number of Conjugate Gradients (CG) iterations, the speed of the algorithm is increased and explicit evaluation of the Gramian is avoided, as CG only needs the Gramian-vector product $\mathbf{J}^{\mathrm{T}}\mathbf{J}\mathbf{p}$, which can be obtained efficiently by exploiting the block-structure of the Gramian. Following Sorber et al. [5], if we write $\mathbf{p} = \mathrm{vec}([\,\mathbf{P}_1\ \mathbf{P}_2\ \mathbf{P}_3\,])$, then $(\mathbf{J}^{\mathrm{T}}\mathbf{J}\mathbf{p})_{1,1}$, the contribution of the diagonal $1,1$-block of $\mathbf{J}^{\mathrm{T}}\mathbf{J}$ in $\mathbf{J}^{\mathrm{T}}\mathbf{J}\mathbf{p}$, can be computed as $\mathrm{vec}(\mathbf{P}_1[(\mathbf{B}^{\mathrm{T}}\mathbf{B}) * (\mathbf{C}^{\mathrm{T}}\mathbf{C})])$. The contribution of the off-diagonal $1,2$-block of $\mathbf{J}^T\mathbf{J}$, $(\mathbf{J}^{\mathrm{T}}\mathbf{J}\mathbf{p})_{1,2}$, can be computed as $\mathrm{vec}(\mathbf{A}[(\mathbf{C}^{\mathrm{T}}\mathbf{C})*(\mathbf{P}_2^{\mathrm{T}}\mathbf{B})]$. The contributions of the other diagonal and off-diagonal blocks can be obtained analogously. In practice, a block-Jacobi preconditioner [18] with diagonal blocks $[(\mathbf{B}^{\mathrm{T}}\mathbf{B}) * (\mathbf{C}^{\mathrm{T}}\mathbf{C})] \otimes \mathbb{I}_I$, $[(\mathbf{A}^{\mathrm{T}}\mathbf{A}) * (\mathbf{C}^{\mathrm{T}}\mathbf{C})] \otimes \mathbb{I}_J$ and $[(\mathbf{A}^{\mathrm{T}}\mathbf{A}) * (\mathbf{B}^{\mathrm{T}}\mathbf{B})] \otimes \mathbb{I}_K$, where $\mathbb{I}_n$ is the $n \times n$-identity matrix, is also applied to the system to improve the convergence speed.

C. Windowing and dynamic rank

Different weighting strategies can be followed to ensure that recent tensor slices influence the decomposition more than older ones. The use of exponential or truncated/rectangular windows are two popular windowing strategies [13], [19]. The first one uses a weighting matrix $\mathbf{L} = \mathrm{diag}([\lambda^k, \lambda^{k-1}, \ldots, \lambda, 1])$, so that every old slice is scaled down by a factor λ whenever a new slice arrives. The second one only considers the last M slices for the update and thus truncates the tensor by removing its outdated slices. Its weighting matrix looks as follows: $\mathbf{L} = \mathrm{diag}([0, \ldots, 0, 1, \ldots, 1])$, where $\mathbf{L}$ contains M ones. Both types can be combined to obtain a truncated exponential window with weighting matrix $\mathbf{L} = \mathrm{diag}([0, 0, \ldots, 0, \lambda^{M-1}, \lambda^{M-2}, \ldots, \lambda, 1])$.

Windowing can easily be incorporated into the updating algorithm, by modifying the objective function:

$$\min_{\mathbf{A},\mathbf{B},\mathbf{C}} ||\,[\![\mathbf{A},\mathbf{B},\mathbf{L}\mathbf{C}]\!] - \mathcal{T}^{(\mathrm{up})} \cdot_3 \mathbf{L}||_F^2.$$

The factor matrices $\mathbf{C}$ and $\mathbf{Z}$ are thus replaced by $\mathbf{L}\mathbf{C}$ and $\overline{\mathbf{L}}\mathbf{Z}$, respectively, where $\overline{\mathbf{L}}$ is the matrix $\mathbf{L}$ without its last row and column. If required, the factor matrix $\mathbf{C}$ can be recovered from $\mathbf{L}\mathbf{C}$ by left multiplication with $\mathbf{L}^\dagger$, where $\mathbf{L}^\dagger$ is obtained by inverting the non-zero entries of $\mathbf{L}$. Otherwise, $\lambda\mathbf{L}\mathbf{C}$ can directly be used for the initialization of the next updating step.

The updating method can easily be adapted to admit changes of the tensor rank, as in every update, the rank of the new CPD can be adjusted. To see this, note that the objective function (2) and the gradients do not change if $[\![\mathbf{X},\mathbf{Y},\mathbf{Z}]\!]$ has a different rank than $[\![\mathbf{A},\mathbf{B},\mathbf{C}]\!]$. Determining when the rank should change, is more difficult, however. See [20] for an extended discussion.

Algorithm 1: NLS updating for CPD

```
  Input : Old CPD $[\![\mathbf{X},\mathbf{Y},\mathbf{Z}]\!]_R$, new slice $\mathbf{M}$, windowing
          matrix $\mathbf{L}$, max number of GN iterations $P$, max
          number of CG iterations $Q$
  Output: Updated factor matrices $\mathbf{A}$, $\mathbf{B}$ and $\mathbf{C}$
1 Solve $(\mathbf{Y} \odot \mathbf{X})\mathbf{c}_{\mathrm{new}} = \mathrm{vec}(\mathbf{M})$ using Equation (1)
2 Decide on rank $R'$ of the updated CPD based on the
  error $||(\mathbf{Y} \odot \mathbf{X})\mathbf{c}_{\mathrm{new}} - \mathrm{vec}(\mathbf{M})||_F$ (default $R' = R$)
3 Concatenate $\mathbf{c}_{\mathrm{new}}$ to $\mathbf{Z}$ to obtain the initialization
  $\left[\!\!\left[\mathbf{X},\mathbf{Y},\mathbf{L}\begin{bmatrix}\mathbf{Z}\\ \mathbf{c}_{\mathrm{new}}^{\mathrm{T}}\end{bmatrix}\right]\!\!\right]_R$
4 Remove column or add random column to the
  initialization if $R' \neq R$
5 Solve the NLS-problem
  $\min_{\mathbf{A},\mathbf{B},\mathbf{C}} ||\,[\![\mathbf{A},\mathbf{B},\mathbf{L}\mathbf{C}]\!]_{R'} - \mathcal{T}^{(\mathrm{up})} \cdot_3 \mathbf{L}||_F^2$ with max $P$
  GN iterations and max $Q$ preconditioned CG iterations
  per GN iteration, using the efficient evaluations in
  Section III
6 Recover $\mathbf{C}$ from $\mathbf{C} = \mathbf{L}^\dagger(\mathbf{L}\mathbf{C})$ or store $\mathbf{L}\mathbf{C}$
7 Return the updated factor matrices $\mathbf{A}$, $\mathbf{B}$ and $\mathbf{C}$
```

D. Complexity analysis

A dogleg trust-region Gauss-Newton method is used to compute the CPD updates. This method performs a maximum of P iterations wherein the optimization step $\mathbf{p}_p$ is computed by solving $\mathbf{J}_p^{\mathrm{T}}\mathbf{J}_p\mathbf{p}_p = -\nabla f_p$, where $\mathbf{J}$ is the Jacobian of f. The latter system is solved by preconditioned CG with

Q iterations for which the complexity is dominated by the evaluation of $\mathbf{J}_p^{\mathrm{T}}\mathbf{J}_p\mathbf{p}_p$. Each update thus requires a maximum of P evaluations of f and ∇f and PQ evaluations of $\mathbf{J}_p^{\mathrm{T}}\mathbf{J}_p\mathbf{p}_p$ plus an additional number P' of evaluations of f for the trust-region method. P' is typically equal to P.

For the objective function, it can be noted that $\frac{1}{2}|| [\![\mathbf{X},\mathbf{Y},\mathbf{Z}]\!] ||_F^2$ and $\frac{1}{2}||\mathbf{M}||_F^2$ are constant and only have to be computed once. Assuming $R' = R$, the terms $\frac{1}{2}|| [\![\mathbf{A},\mathbf{B},\overline{\mathbf{C}}]\!] ||_F^2$ and $\big\langle [\![\mathbf{A},\mathbf{B},\overline{\mathbf{C}}]\!], [\![\mathbf{X},\mathbf{Y},\mathbf{Z}]\!]\big\rangle$ can be computed in $\mathcal{O}\big(R^2\max(I,J,M)\big)$ flops, with M the length of the window, using the simplification of equation (3). The last term $\langle [\![\mathbf{A},\mathbf{B},\underline{\mathbf{c}}]\!], \mathbf{M}\rangle$ can be computed in $\mathcal{O}\big(IJ\big)$ flops, totaling at a complexity of $\mathcal{O}\big(2R^2\max(I,J,M)+IJ\big)$ flops.

The gradient ∇f consists of three terms, $\frac{\partial f}{\partial \mathbf{A}}$, $\frac{\partial f}{\partial \mathbf{B}}$ and $\frac{\partial f}{\partial \mathbf{C}}$. They can all be computed in $\mathcal{O}\big(2R^2\max(I,J,M)+IJR\big)$ flops, totaling at $\mathcal{O}\big(6R^2\max(I,J,M)+3IJR\big)$ flops.

The Gramian-vector product $\mathbf{J}_p^{\mathrm{T}}\mathbf{J}_p\mathbf{p}_p$ requires $\mathcal{O}\big(3R^2\max(I,J,M)\big)$ flops per CG iteration [5] and thus $\mathcal{O}\big(3QR^2\max(I,J,M)\big)$ flops per GN iteration. Preconditioning adds three $R \times R$ matrix inversions and Q matrix-vector products per GN iteration, totaling $\mathcal{O}\big(3R^3+QR(I+J+M)\big)$ flops.

Summing these values for P iterations of the method and adding the P' evaluations of f for the trust-region method, leads to a total time complexity of $\mathcal{O}\big((8P+3QP+2P')R^2\max(I,J,M)+(3RP+P+P')IJ+3PR^3+PQR(I+J+M)\big)$, which for low rank tensors and a truncated window is dominated by the term $3RPIJ$. Note that this term only depends on the dimensions of the new slice $\mathbf{M}$ and not on the window length M.

The memory consumption of the proposed method is dominated by the storage of the old and new CPD, which is $\mathcal{O}\big(R(I+J+M)\big)$ and the new tensor slice, which is $\mathcal{O}\big(IJ\big)$. The gradients and Gramian-vector products that are used during the execution of the method both require $\mathcal{O}\big(R(I+J+M)\big)$ memory as well. In contrast, storing the full (windowed) tensor would require $\mathcal{O}\big(IJM\big)$ memory.

IV. Experiments

We compare the proposed method with batch algorithms for the CPD and with the PARAFAC-SDT and PARAFAC-RLST methods of Nion et al. [13]. All computations are done in Tensorlab [11]. The batch algorithms simply compute the CPD of the tensor formed by the slices in the window. These algorithms are a nonlinear least squares (NLS) algorithm, called `cpd_nls`, and an alternating least squares (ALS) algorithm, called `cpd_als`, both available in Tensorlab. In every step, they are initialized with the decomposition from the previous updating step, with the aforementioned least squares solution $\mathbf{c}_{\text{new}} = (\mathbf{Y}\odot\mathbf{X})^{\dagger}\text{vec}(\mathbf{M})$, corresponding to the new slice, concatenated to the third factor matrix. All optimization methods are limited to $P = 1$ iterations, while $Q = 5$ CG iterations are allowed for the linear systems that are solved during the algorithms. The experiments are performed on a computer with an Intel Core i7-6820HQ CPU at 2.70GHz and 16GB of RAM using MATLAB R2016b and Tensorlab 3.0.

Initially, a rank-R tensor of dimensions $1000\times1000\times100$ is generated by sampling two factor matrices from the standard normal distribution. The third factor matrix is sampled along a third-degree polynomial with coefficients from the standard normal distribution to obtain a model that varies slowly along its third mode. The different experiments have $R = 2, 4$ and 6, respectively. The tensor is perturbed by uniformly distributed noise over the interval $[-0.5, 0.5]$ with a signal-to-noise ratio (SNR) of $20\,\text{dB}$. First, a full CPD is computed for the first fifty mode-3 slices of the tensor. The other slices are then added one by one to the tensor, after which the decomposition is updated. A truncated exponential window of length $M = 30$ with forgetting factor $\lambda = 0.9$ is applied. Reported values are medians across ten runs.

In Table I, the median required CPU-time to compute an update using five different methods is shown. The five methods are the batch NLS and batch ALS algorithms using only slices from the truncated exponential window, i.e., using only the last M slices of the tensor, and PARAFAC-SDT, PARAFAC-RLST and the proposed updating method using the same window. For these large tensors, the updating method achieves about the same speed as PARAFAC-SDT and both are a factor 10 to 50 faster than the batch methods. For smaller tensors, the updating method is slower than PARAFAC-SDT.

In Table II, the median fitting errors over the fifty updating steps are shown. The error is here defined as the average error for all tensor entries, where the slices are weighted using the windowing matrix, i.e., the errors of older slices have a smaller weight in the mean than those of newer slices. The accuracy of the updating method is close to that of the batch methods, while PARAFAC-SDT and PARAFAC-RLST perform a lot worse, especially for larger values of R. As only one iteration is performed by the updating method during the experiments ($P = 1$ and $Q = 5$), higher accuracy can easily be traded for longer execution times by increasing P and/or Q. Results for $P = 5$ and $Q = 25$ are also included in Tables I and II. It can be seen that increasing the number of iterations does improve the results slightly. However, the execution time rises linearly with the number of performed iterations. In Figure 2, the errors of the different methods are plotted for the case $R = 6$ and $\text{SNR} = 50\,\text{dB}$. The updating method achieves an accuracy that is close to that of the batch methods. Increasing the number of iterations improves the results marginally. The error also remains relatively constant over the fifty updates, in contrast to PARAFAC-SDT, for which the errors accumulate.

Summarizing, the error of the updating method is slightly larger than the error of the batch methods, but this is compensated by its superior speed. As only the old CPD and the new slice have to be stored, the memory cost is lower compared to the batch methods, which have to store the last M tensor slices. For tensors with millions of entries or time-sensitive applications, this can make an important difference in the applicability of tensor methods. Although PARAFAC-SDT has the same speed as the updating method for large tensors, it consistently yielded a lower accuracy.

V. Conclusion

An NLS updating method was proposed for the CPD that exploits its structure to execute a fast NLS update whenever a new slice arrives. The batch NLS algorithm for the CPD is adapted so that it can be used in an updating context. By only using the previous decomposition and the new tensor slice when it arrives, the updating method becomes both time- and memory-efficient, while maintaining a good accuracy for the decomposition. Efficient expressions are derived for the computation of the objective function, gradient and Gramian. It is also shown that arbitrary windowing strategies and changes of the tensor rank can be handled straightforwardly. Finally, the performance of the method is demonstrated on a large-scale tensor. The algorithm is faster than batch ALS and NLS algorithms for the numerical experiments, while maintaining good accuracy, especially compared to PARAFAC-SDT and PARAFAC-RLST. As only the new slice and the old factor matrices are needed in the computation of the update, updating is very memory efficient, making it applicable for large-scale problems. A possible drawback of this memory-efficient approach is that small latent trends in the data may be ignored during multiple consecutive updates as these trends are dominated by the current model in every step. This could be mitigated by tracking some additional information, e.g., the previous (few) slice(s) or an extra rank-1 term.

Table I: Medians of the CPU-time (in ms) for a single update using the new updating method, NLS and ALS batch methods and PARAFAC-SDT and PARAFAC-RLST updating methods.

R	2	3	4	5	6
Update 1 $_{P=1,Q=5}$	60	81	104	140	169
NLS	2375	4464	2557	3563	5522
ALS	910	1222	1400	1401	2352
SDT	48	71	98	136	172
RLST	570	607	623	775	822
Update 2 $_{P=5,Q=25}$	166	217	296	398	495

Table II: Weighted mean errors for the new updating method, NLS and ALS batch algorithms and PARAFAC-SDT and PARAFAC-RLST. Medians over the fifty updating steps.

R	2	4	6
Update 1 $_{P=1,Q=5}$	$1.22 \cdot 10^{-2}$	$1.20 \cdot 10^{-2}$	$1.09 \cdot 10^{-2}$
NLS	$1.11 \cdot 10^{-2}$	$8.84 \cdot 10^{-3}$	$9.98 \cdot 10^{-3}$
ALS	$1.11 \cdot 10^{-2}$	$8.84 \cdot 10^{-3}$	$9.98 \cdot 10^{-3}$
SDT	$2.47 \cdot 10^{-2}$	$4.38 \cdot 10^{-2}$	$6.16 \cdot 10^{-2}$
RLST	$2.68 \cdot 10^{-2}$	$2.63 \cdot 10^{-1}$	$8.07 \cdot 10^{-1}$
Update 2 $_{P=5,Q=25}$	$1.16 \cdot 10^{-2}$	$1.18 \cdot 10^{-2}$	$1.06 \cdot 10^{-2}$

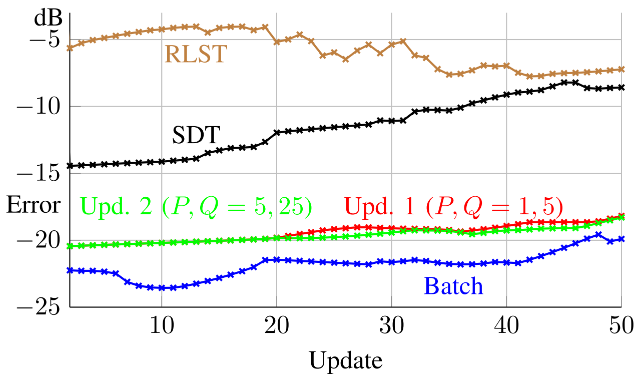

Figure 2. Weighted mean errors of the new updating method, ALS and NLS batch methods and PARAFAC-SDT and PARAFAC-RLST updating methods for $R = 6$ and SNR $= 50$ dB on a decibel scale.

References

[1] T. G. Kolda and B. W. Bader, "Tensor decompositions and applications," *SIAM Review*, vol. 51, no. 3, pp. 455–500, 2009.

[2] N. D. Sidiropoulos, L. De Lathauwer, X. Fu, K. Huang, E. E. Papalexakis, and C. Faloutsos, "Tensor decomposition for signal processing and machine learning," *IEEE Transactions on Signal Processing*, vol. 65, no. 13, pp. 3551–3582, 2017.

[3] A. Cichocki, C. Mandic, A. Phan, C. Caiafa, G. Zhou, Q. Zhao, and L. De Lathauwer, "Tensor decompositions for signal processing applications. From two-way to multiway component analysis," *IEEE Signal Processing Magazine*, vol. 32, pp. 145–163, 2015.

[4] A. H. Phan and A. Cichocki, "PARAFAC algorithms for large-scale problems," *Neurocomputing*, vol. 74, no. 11, pp. 1970–1984, 2011.

[5] L. Sorber, M. Van Barel, and L. De Lathauwer, "Optimization-based algorithms for tensor decompositions: Canonical polyadic decomposition, decomposition in rank-$(L_r, L_r, 1)$ terms, and a new generalization," *SIAM Journal on Optimization*, vol. 23, no. 2, pp. 695–720, 2013.

[6] I. Domanov and L. De Lathauwer, "Canonical polyadic decomposition of third-order tensors: Reduction to generalized eigenvalue decomposition," *SIAM Journal on Matrix Analysis and Applications*, vol. 35, no. 2, pp. 636–660, 2014.

[7] N. Vervliet, O. Debals, L. Sorber, and L. De Lathauwer, "Breaking the curse of dimensionality using decompositions of incomplete tensors: Tensor-based scientific computing in big data analysis," *IEEE Signal Processing Magazine*, vol. 31, no. 5, pp. 71–79, 2014.

[8] N. Vervliet and L. De Lathauwer, "A randomized block sampling approach to canonical polyadic decomposition of large-scale tensors," *IEEE Journal of Selected Topics in Signal Processing*, vol. 10, no. 2, pp. 284–295, 2016.

[9] E. Papalexakis, C. Faloutsos, and N. D. Sidiropoulos, "Parcube: Sparse parallelizable tensor decompositions," *Machine Learning and Knowledge Discovery in Databases*, pp. 521–536, 2012.

[10] L. Sorber, M. Van Barel, and L. De Lathauwer, "Structured data fusion," *IEEE Journal of Selected Topics in Signal Processing*, vol. 9, no. 4, pp. 586–600, 2015.

[11] N. Vervliet, O. Debals, L. Sorber, M. Van Barel, and L. De Lathauwer. Tensorlab 3.0. Available online, March 2016. URL: http://www.tensorlab.net.

[12] M. Mardani, G. Mateos, and G. B. Giannakis, "Subspace learning and imputation for streaming big data matrices and tensors," *IEEE Transactions on Signal Processing*, vol. 63, no. 10, pp. 2663–2677, 2015.

[13] D. Nion and N. D. Sidiropoulos, "Adaptive algorithms to track the parafac decomposition of a third-order tensor," *IEEE Transactions on Signal Processing*, vol. 57, no. 6, pp. 2299–2310, 2009.

[14] J. Sun, D. Tao, S. Papadimitriou, P. S. Yu, and C. Faloutsos, "Incremental tensor analysis: Theory and applications," *ACM Transactions on Knowledge Discovery from Data*, vol. 2, no. 3, pp. 11:1–11:37, October 2008.

[15] N. Vervliet, O. Debals, and L. De Lathauwer, "Tensorlab 3.0 — numerical optimization strategies for large-scale constrained and coupled matrix/tensor factorization," in *Conference Record of the 50th Asilomar Conference on Signals, Systems and Computers (ASILOMAR 2016)*, November 2016.

[16] ——, "Exploiting efficient data representations in tensor decompositions," *Technical Report 16-174, ESAT-STADIUS, KU Leuven, Belgium*, 2016.

[17] N. Vannieuwenhoven, K. Meerbergen, and R. Vandebril, "Computing the gradient in optimization algorithms for the CP decomposition in constant memory through tensor blocking," *SIAM Journal on Scientific Computing*, vol. 37, no. 3, pp. C415–C438, 2015.

[18] M. Benzi, "Preconditioning techniques for large linear systems: a survey," *Journal of Computational Physics*, vol. 182, no. 2, pp. 418–477, 2002.

[19] M. Moonen and B. De Moor, *SVD and Signal Processing, III: Algorithms, Architectures and Applications*. Elsevier, 1995.

[20] Q. Zhao, L. Zhang, and A. Cichocki, "Bayesian CP factorization of incomplete tensors with automatic rank determination," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 37, no. 9, pp. 1751–1763, 2015.