CPD UPDATING USING LOW-RANK WEIGHTS

Michiel Vandecappelle[⋆†]*, Martijn Bousse*[⋆]*, Nico Vervliet*[⋆]*, and Lieven De Lathauwer*[⋆†]

[⋆]Department of Electrical Engineering (ESAT), KU Leuven, Kasteelpark Arenberg 10, 3001 Leuven, Belgium
[†]Group Science, Engineering and Technology, KU Leuven Kulak, E. Sabbelaan 53, 8500 Kortrijk, Belgium
Email: {Michiel.Vandecappelle, Martijn.Bousse, Nico.Vervliet, Lieven.DeLathauwer}@kuleuven.be

ABSTRACT

Tensor updating methods enable tensor decompositions to adapt quickly when new data is added to the tensor. At present, updating methods for the canonical polyadic decomposition (CPD) give every tensor entry the same weight. In practice, however, data quality or relative importance might differ between tensor entries, which warrants the use of more general weighting schemes. In this paper, an NLS updating method is developed for the CPD that uses a weighted least squares (WLS) approach with a low-rank weight tensor. This weight tensor itself can also be updated to allow dynamic weighting schemes. By exploiting the CPD structure of both the data and weight tensors, the algorithm obtains better accuracy than the unweighted updating methods, while being more time- and memory efficient than batch WLS methods.

Index Terms— tensors, updating, weighted least squares.

1. INTRODUCTION

The popularity of tensor methods has been rising steadily in the past years and they have been applied in a wide variety of fields such as machine learning and signal processing [1, 2]. Tensors are higher-order extensions of vectors (first-order) and matrices (second-order). Tensor decompositions allow the compact storage of large tensors, enabling both compression and analysis of big, higher-order datasets. Apart from the classical alternating least squares (ALS) algorithms, several algebraic and all-at-once optimization-based algorithms for the computation of tensor decompositions have been developed: See, for example, [3–9] and references therein.

Signals are often perturbed by noise. If prior information about the noise is available, this information can be leveraged in the computation of tensor decompositions such as the canonical polyadic decomposition (CPD) to improve the quality of the model. Different tensor entries can be weighted differently in the least squares (LS) cost function. The weights may take into account the quality of the data of the corresponding tensor entries. As an example, consider an array of sensors with varying signal-to-noise ratios (SNR). In this case, it makes sense to weight entries according to the quality of the sensor they originate from, giving more weight to data from the better sensors. Although weighted LS (WLS) algorithms exist that admit general weight tensors [10, 11], it can be beneficial to use a low-rank weight tensor. First, low-rank structure can be exploited to make the decomposition less demanding, both in storage and computation, compared to using a full weight tensor [12]. Also, approximating the weight tensor is usually good enough, as a high accuracy of the weights is generally unnecessary or even unobtainable. Note that the standard CPD methods can be seen as WLS methods with a specific rank-1 weight tensor (consisting of only ones).

In practice, if new data is added to a tensor at regular time-intervals, its storage can quickly become problematic. In contrast, the CPD of this tensor needs only a fraction of the memory that the full tensor requires, while the important features of the data are maintained. Efficient updating methods for the CPD have been developed [13–15], but by combining CPD updating with a low-rank WLS method into a weighted CPD updating method, we can get the best of two worlds. Returning to the sensor array example, it would be useful to leverage the prior information about the sensors in the CPD updating process by incorporating a low-rank weight tensor. Even stronger, one can use information about the current state of the sensors to update the low-rank weight tensor itself, e.g., some types of sensors might have warm-up times or they may become influenced by external conditions that affect their performance. Therefore, we develop a weighted nonlinear LS (NLS) CPD updating algorithm that exploits the low-rank structure of both the weight and data tensors, as described in [16,17]. This makes the algorithm computation- and memory-efficient, properties that are primordial if the method is to be applied in an updating context. We use Tensorlab [18], a toolbox for tensor computations in MATLAB, for the implementation.

We fix notation, basic definitions and useful identities in the remainder of this section. Our method is derived in Section 2. We illustrate the algorithm on a practical example in Section 3.

1.1. Notation, definitions and identities

Scalars, vectors, matrices and tensors are denoted by lowercase (a), bold lowercase ($\mathbf{a}$), bold uppercase letters ($\mathbf{A}$), and letters in calligraphic script ($\mathcal{T}$), respectively. The kth frontal slice $\mathbf{T}_{::k}$ of a tensor is a matrix obtained by fixing the third index of the tensor. A mode-n unfolding of a tensor $\mathcal{T}$ is a matrix $\mathbf{T}_{(n)}$ with the mode-n vectors as its columns following the ordering in [19]. The vectorization operator is denoted by $\text{vec}(\cdot)$ and $[\mathbf{a}^{\text{T}}; \mathbf{b}^{\text{T}}]$ refers to row-wise concatenation. Kronecker, Hadamard, column- and row-wise Khatri–Rao products of matrices are written as $\otimes$, $*$, $\odot$, and $\odot^{\text{T}}$, respectively. The Moore–Penrose pseudoinverse of a matrix $\mathbf{A}$ is denoted by $\mathbf{A}^{\dagger}$. A tensor has rank 1 if it is the outer product of three non-zero vectors. The tensor rank is the minimal number of rank-1 tensors needed to write the tensor as their linear combination. The derivations will focus on the case of third-order real tensors. The CPD of a tensor

Funding: Michiel Vandecappelle is supported by an SB 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.

decomposes a rank-R tensor $\mathcal{T} \in \mathbb{R}^{I\times J\times K}$ as a linear combination of R rank-1 terms: $\mathcal{T} = \sum_{r=1}^{R} \mathbf{a}_r \otimes \mathbf{b}_r \otimes \mathbf{c}_r = [\![\mathbf{A}, \mathbf{B}, \mathbf{C}]\!]_R$, with $\mathbf{A} = [\mathbf{a}_1 \dots \mathbf{a}_R] \in \mathbb{R}^{I\times R}$, and similarly for $\mathbf{B}$ and $\mathbf{C}$. The mode-3 unfolding is given by $\mathbf{T}_{(3)} = \mathbf{C}(\mathbf{B} \odot \mathbf{A})^{\mathrm{T}}$. Useful identities are:

$$[\![\mathbf{A}, \mathbf{B}, \mathbf{C}]\!] * [\![\mathbf{D}, \mathbf{E}, \mathbf{F}]\!] = [\![\mathbf{A} \odot^{\mathrm{T}} \mathbf{D}, \mathbf{B} \odot^{\mathrm{T}} \mathbf{E}, \mathbf{C} \odot^{\mathrm{T}} \mathbf{F}]\!] \ , \qquad (1)$$

$$\langle [\![\mathbf{A}, \mathbf{B}, \mathbf{C}]\!], [\![\mathbf{D}, \mathbf{E}, \mathbf{F}]\!] \rangle = \mathbf{1}^{\mathrm{T}}[(\mathbf{D}^{\mathrm{T}}\mathbf{A}) * (\mathbf{E}^{\mathrm{T}}\mathbf{B}) * (\mathbf{F}^{\mathrm{T}}\mathbf{C})]\mathbf{1}. \qquad (2)$$

2. NLS CPD UPDATING USING LOW-RANK WEIGHTS

Assume that one has computed the rank-R CPD $[\![\mathbf{X}, \mathbf{Y}, \overline{\mathbf{Z}}]\!]_R$ of a third-order tensor $\overline{\mathcal{T}} \in \mathbb{R}^{I\times J\times K}$ using a WLS approach with low-rank weight tensor $\overline{\mathcal{W}} = [\![\mathbf{N}, \mathbf{P}, \overline{\mathbf{Q}}]\!]_L$. Then, let frontal slices $\mathbf{M}$ and $\mathbf{V}$ be added to both $\overline{\mathcal{T}}$ and $\overline{\mathcal{W}}$ in the third mode, as shown in Figure 1, obtaining the tensors $\mathcal{T}$ and $\mathcal{W}$, respectively. First, we want to update the CPD of the weight tensor to $[\![\mathbf{N}, \mathbf{P}, \mathbf{Q}]\!]_L$, where a new row $\underline{\mathbf{q}}$ is appended to $\overline{\mathbf{Q}}$ to include its new slice: $\mathbf{Q} = [\,\overline{\mathbf{Q}}\,;\,\underline{\mathbf{q}}\,]$. This can be done using the CPD updating method described in [13]. Next, we would like to use this updated weight tensor to update the CPD of $\overline{\mathcal{T}}$ to a CPD $[\![\mathbf{A}, \mathbf{B}, \mathbf{C}]\!]_{R^*}$ of $\mathcal{T}$ using a WLS approach with weight tensor $\mathcal{W}$, where R^* can be different from R.

The algorithm builds upon a dogleg trust-region Gauss-Newton (GN) algorithm, but the expressions are valid for more general NLS and quasi-Newton (QN) algorithms. The GN algorithm solves the following minimization problem:

$$\min_{\mathbf{s}_k} \frac{1}{2} \|\mathrm{vec}(\mathcal{F}_k) + \mathbf{J}_k\mathbf{s}_k\|_F^2 \quad \text{such that} \quad \|\mathbf{s}_k\| \leq \Delta_k,$$

where $\mathcal{F}_k = \mathcal{W} * ([\![\mathbf{A}, \mathbf{B}, \mathbf{C}]\!] - \mathcal{T})$ is the residual, and $\mathbf{s}_k$, $\mathbf{J}_k$ and Δ_k are the step, Jacobian and trust radius in iteration k, respectively. To increase efficiency, preconditioned conjugate gradient (PCG) iterations are used to solve the linear system $\mathbf{J}_k^{\mathrm{T}}\mathbf{J}_k\mathbf{s}_k = -\nabla f_k$ for the step $\mathbf{s}_k$, where ∇ is the gradient operator. Efficient expressions for the objective function, gradient and Gramian-vector products can be derived for the GN algorithm with PCG by exploiting the CPD structure of $\mathcal{W}$ and $\mathcal{T}$. We use a similar approach as that in [13] to obtain these expressions. They remain valid when the rank of the new CPD differs from that of the old CPD, so that the algorithm enables rank-changes between updates. The use of a block-Jacobi preconditioner and an initialization strategy are also discussed. The section ends with an analysis of the complexity of the algorithm. The full algorithm is given in Algorithm 1.

Algorithm 1: WLS CPD updating

Input : CPD $[\![\mathbf{X}, \mathbf{Y}, \overline{\mathbf{Z}}]\!]_R$ of old tensor $\overline{\mathcal{T}}$, new slice $\mathbf{M}$, CPD $[\![\mathbf{N}, \mathbf{P}, \overline{\mathbf{Q}}]\!]_L$ of old weighting tensor $\overline{\mathcal{W}}$, new weighting slice $\mathbf{V}$, number of GN iterations $\mathrm{it}_{\mathrm{GN}}$, number of CG iterations $\mathrm{it}_{\mathrm{CG}}$, new rank R^*.

Output: CPD $[\![\mathbf{A}, \mathbf{B}, \mathbf{C}]\!]_{R^*}$ of $\mathcal{T}$, CPD of $\mathcal{W}$.

1. Compute CPD of $\mathcal{W}$ using (unweighted) CPD updating.
2. Initialize $\mathbf{A}, \mathbf{B}, \mathbf{C}$ using (3).
3. Solve the NLS-problem $\min_{\mathbf{A},\mathbf{B},\mathbf{C}} \left\|\mathcal{W} * \left([\![\mathbf{A}, \mathbf{B}, \mathbf{C}]\!]_{R^*} - \mathcal{T}\right)\right\|_F^2$ with $\mathrm{it}_{\mathrm{GN}}$ GN iterations and $\mathrm{it}_{\mathrm{CG}}$ preconditioned CG iterations per GN iteration, using the efficient evaluations in Section 2.
4. Return the updated factor matrices $\mathbf{A}$, $\mathbf{B}$ and $\mathbf{C}$.

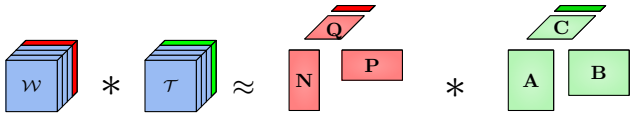


Fig. 1. WLS updating using low-rank weights enables efficient CPD updates by exploiting the low-rank structure of *both* the weight and data tensor. Left: the weight ($\mathcal{W}$) and the data ($\mathcal{T}$) tensor are extended with an extra slice (red and green, respectively) in the third mode. Right: first, the CPD of the weight tensor is updated by adding a new vector (red) to the factor matrix in the third mode and modifying the existing factor matrices (pink) to obtain the CPD $[\![\mathbf{N}, \mathbf{P}, \mathbf{Q}]\!]$. This updated low-rank weight tensor is then used to compute the updated CPD model $[\![\mathbf{A}, \mathbf{B}, \mathbf{C}]\!]$ of the data in a similar way, this time using a WLS approach (dark and light green).

2.1. Objective function

The WLS objective function for the computation of the CPD $[\![\mathbf{A}, \mathbf{B}, \mathbf{C}]\!]$ of the updated low-rank tensor $\mathcal{T}$ using the updated weight tensor $\mathcal{W}$ is as follows:

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

By splitting $\mathbf{C}$ as $[\,\overline{\mathbf{c}}\,;\,\underline{\mathbf{c}}\,]$, with $\underline{\mathbf{c}}$ the last row of $\mathbf{C}$, and similarly for the factor matrix $\mathbf{Q}$ of $\mathcal{W}$, f can be expanded as

$$f = \frac{1}{2} \left\| [\![\mathbf{N}, \mathbf{P}, \overline{\mathbf{Q}}]\!] * \left([\![\mathbf{A}, \mathbf{B}, \overline{\mathbf{C}}]\!] - \overline{\mathcal{T}}\right)\right\|_F^2 + \frac{1}{2} \left\| [\![\mathbf{N}, \mathbf{P}, \underline{\mathbf{q}}]\!] * ([\![\mathbf{A}, \mathbf{B}, \underline{\mathbf{c}}]\!] - \mathbf{M})\right\|_F^2 .$$

To limit memory usage, the tensor $\overline{\mathcal{T}}$ is never stored during the updating process. Its CPD approximation $[\![\mathbf{X}, \mathbf{Y}, \overline{\mathbf{Z}}]\!]$ is used instead. Thus, using (1), the objective function becomes

$$f \approx \frac{1}{2} \left\| \left[\!\left[\mathbf{A}', \mathbf{B}', \overline{\mathbf{C}}'\right]\!\right] - [\![\mathbf{X}', \mathbf{Y}', \overline{\mathbf{Z}}']\!] \right\|_F^2 + \frac{1}{2} \left\| [\![\mathbf{A}', \mathbf{B}', \underline{\mathbf{c}}']\!] - [\![\mathbf{N}, \mathbf{P}, \underline{\mathbf{q}}]\!] * \mathbf{M}\right\|_F^2 ,$$

where $\mathbf{A}' = \mathbf{N} \odot^{\mathrm{T}} \mathbf{A}$, $\mathbf{B}' = \mathbf{P} \odot^{\mathrm{T}} \mathbf{B}$, $\overline{\mathbf{C}}' = \overline{\mathbf{Q}} \odot^{\mathrm{T}} \overline{\mathbf{C}}$, $\underline{\mathbf{c}}' = \underline{\mathbf{q}} \odot^{\mathrm{T}} \underline{\mathbf{c}}$ etc. We can expand f to obtain

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

Here, we used that $\frac{1}{2} \left\| \left[\!\left[\mathbf{A}', \mathbf{B}', \overline{\mathbf{C}}'\right]\!\right] \right\|_F^2 + \frac{1}{2} \|[\![\mathbf{A}', \mathbf{B}', \underline{\mathbf{c}}']\!]\|_F^2 = \frac{1}{2} \|[\![\mathbf{A}', \mathbf{B}', \mathbf{C}']\!]\|_F^2$ and defined the weighted new slice $\mathbf{M}'$ as $[\![\mathbf{N}, \mathbf{P}, \underline{\mathbf{q}}]\!] * \mathbf{M}$. The CPD structure of f can be exploited to obtain expressions that admit efficient computations. We use (2) to simplify the first two terms to: $\mathbf{1}^{\mathrm{T}}[(\mathbf{A}'^{\mathrm{T}}\mathbf{A}') * (\mathbf{B}'^{\mathrm{T}}\mathbf{B}') * (\mathbf{C}'^{\mathrm{T}}\mathbf{C}')]\mathbf{1}$ and $\mathbf{1}^{\mathrm{T}}[(\mathbf{X}'^{\mathrm{T}}\mathbf{X}') * (\mathbf{Y}'^{\mathrm{T}}\mathbf{Y}') * (\overline{\mathbf{Z}}'^{\mathrm{T}}\overline{\mathbf{Z}}')]\mathbf{1}$, and the third term to $\mathbf{1}^{\mathrm{T}}[(\mathbf{X}'^{\mathrm{T}}\mathbf{A}') * (\mathbf{Y}'^{\mathrm{T}}\mathbf{B}') * (\overline{\mathbf{Z}}'^{\mathrm{T}}\overline{\mathbf{C}}')]\mathbf{1}$. The last two terms are simply inner products of matrices.

2.2. Gradient

The gradient $\nabla f = \left[\text{vec}\left(\frac{\partial f}{\partial \mathbf{A}}\right); \text{vec}\left(\frac{\partial f}{\partial \mathbf{B}}\right); \text{vec}\left(\frac{\partial f}{\partial \mathbf{C}}\right)\right]$, which we denote by $[\mathbf{g}_A; \mathbf{g}_B; \mathbf{g}_C]$ has the following terms:

$$\mathbf{g}_A = \left(\mathbf{M}^A\right)^{\mathrm{T}} \text{vec}\Big[\mathbf{A}'[(\mathbf{B}'^{\mathrm{T}}\mathbf{B}') * (\mathbf{C}'^{\mathrm{T}}\mathbf{C}')] - \mathbf{X}'[(\mathbf{Y}'^{\mathrm{T}}\mathbf{B}') * (\overline{\mathbf{Z}}'^{\mathrm{T}}\overline{\mathbf{C}})] - \mathbf{M}'\mathbf{B}'\text{diag}(\underline{\mathbf{c}}')\Big] ,$$

$$\mathbf{g}_B = \left(\mathbf{M}^B\right)^{\mathrm{T}} \text{vec}\Big[\mathbf{B}'[(\mathbf{A}'^{\mathrm{T}}\mathbf{A}') * (\mathbf{C}'^{\mathrm{T}}\mathbf{C}')] - \mathbf{Y}'[(\mathbf{X}'^{\mathrm{T}}\mathbf{A}') * (\overline{\mathbf{Z}}'^{\mathrm{T}}\overline{\mathbf{C}}')] - \mathbf{M}'^{\mathrm{T}}\mathbf{A}'\text{diag}(\underline{\mathbf{c}}')\Big] ,$$

$$\mathbf{g}_C = \left(\mathbf{M}^C\right)^{\mathrm{T}} \text{vec}\Big[\mathbf{C}'[(\mathbf{A}'^{\mathrm{T}}\mathbf{A}') * (\mathbf{B}'^{\mathrm{T}}\mathbf{B}')] + \begin{bmatrix} -\overline{\mathbf{Z}}'[(\mathbf{X}'^{\mathrm{T}}\mathbf{A}') * (\mathbf{Y}'^{\mathrm{T}}\mathbf{B}')] \\ -\text{vec}(\mathbf{M}')^{\mathrm{T}}(\mathbf{B}' \odot \mathbf{A}') \end{bmatrix}\Big] .$$

where $\mathbf{M}^A = [\,\mathbf{I}_R \otimes \text{diag}(\mathbf{n}_1);\, \ldots\, ; \mathbf{I}_R \otimes \text{diag}(\mathbf{n}_L)\,]$, with $\mathbf{I}_n$ the $(n \times n)$-identity matrix. The matrices $\mathbf{M}^B$ and $\mathbf{M}^C$ are defined analogously using $\mathbf{P}$ and $\mathbf{Q}$. As before, we split $\mathcal{T}'$ into its old part $\overline{\mathcal{T}'} = \overline{\mathcal{W}} * \overline{\mathcal{T}}$ and its new slice $\mathbf{M}'$ and replace $\overline{\mathcal{T}'}$ by its CPD $[\![\mathbf{X}', \mathbf{Y}', \overline{\mathbf{Z}'}]\!]$. By exploiting the CPD structure in this way, we obtain sub-gradients which can be computed efficiently.

2.3. Gramian-vector product and preconditioner

In every CG-iteration, we need to efficiently compute a Gramian-vector product $\mathbf{J}^{\mathrm{T}}\mathbf{J}\boldsymbol{\alpha}$. As the Gramian does not depend on $\mathcal{T}$, we can use similar expressions as those in [12] for the Gramian-vector products and preconditioner. We apply a block-Jacobi preconditioner [20] consisting of the diagonal blocks $\mathbf{P}^A$, $\mathbf{P}^B$ and $\mathbf{P}^C$, where

$$\mathbf{P}^A = \left(\mathbf{M}^A\right)^{\dagger} \left([(\mathbf{B}'^{\mathrm{T}}\mathbf{B}') * (\mathbf{C}'^{\mathrm{T}}\mathbf{C}')]^{-1} \otimes \mathbf{I}_I\right) \left(\mathbf{M}^A\right)^{\dagger\mathrm{T}} ,$$

and $\mathbf{P}^B$ and $\mathbf{P}^C$ are defined analogously. The inverses of the small $(RL \times RL)$-matrices are inexpensive. We can exploit the structure of the problem to avoid the computation and storage of the pseudo-inverses. To evaluate $\mathbf{P}^A\text{vec}(\mathbf{\Gamma}_1)$, where $\mathbf{\Gamma}_1$ has the same dimensions as $\mathbf{A}$, we normalize the rows of $\mathbf{N}$ to obtain $\hat{\mathbf{N}}$ and then compute $(\hat{\mathbf{N}} \odot^{\mathrm{T}} \mathbf{\Gamma}_1)[(\mathbf{B}'^{\mathrm{T}}\mathbf{B}') * (\mathbf{C}'^{\mathrm{T}}\mathbf{C}')]^{-1}$. Next, we tensorize this expression into an $(I \times R \times L)$-tensor $\hat{\mathcal{F}}$ and perform element-wise multiplications of $\hat{\mathbf{N}}$ with every lateral slice of $\hat{\mathcal{F}}$. Finally, we compute the mode-3 sum of this tensor to obtain $\mathbf{P}^A\text{vec}(\mathbf{\Gamma}_1)$. The products $\mathbf{P}^B\text{vec}(\mathbf{\Gamma}_2)$, and $\mathbf{P}^C\text{vec}(\mathbf{\Gamma}_3)$ can be computed analogously, where the $\mathbf{\Gamma}_1$ and $\mathbf{\Gamma}_2$ have the same dimensions as $\mathbf{B}$ and $\mathbf{C}$, respectively.

2.4. Initialization and windowing

Assuming the model does not change too abruptly, the previous CPD $[\![\mathbf{X}, \mathbf{Y}, \overline{\mathbf{Z}}]\!]$ can be used to initialize the algorithm, after extending $\overline{\mathbf{Z}}$ with a new row $\mathbf{c}_{\text{new}}^{\mathrm{T}}$: thus $\mathbf{A} = \mathbf{X}$, $\mathbf{B} = \mathbf{Y}$ and $\mathbf{C} = [\,\overline{\mathbf{z}}\,;\, \mathbf{c}_{\text{new}}^{\mathrm{T}}\,]$. The vector $\mathbf{c}_{\text{new}}$ can be obtained from the LS solution of the system

$$\min_{\mathbf{c}} \frac{1}{2} \left\| \mathbf{V} * \left([\![\mathbf{A}, \mathbf{B}, \mathbf{c}^{\mathrm{T}}]\!] - \mathbf{M} \right) \right\|_F^2 ,$$

where $\mathbf{A}$ and $\mathbf{B}$ are fixed. We can rewrite this as

$$\min_{\mathbf{c}} \frac{1}{2} \left\| (\mathbf{B}' \odot \mathbf{A}')(\mathbf{c} \otimes \underline{\mathbf{q}}^{\mathrm{T}}) - \text{vec}(\mathbf{M}') \right\|_F^2 .$$

Table 1: Per-iteration complexity of the WLS CPD updating algorithm for an $(I \times I \times I)$-tensor.

	Calls/it$_{\text{GN}}$	Complexity
Weighting of factor matrices	1	$\mathcal{O}(3RLI)$
Objective function	1+it$_{\text{Trust Reg.}}$	$\mathcal{O}(2(RL)^2 I + I^2)$
Gradient	1	$\mathcal{O}(6(RL)^2 I + 3RLI^2)$
Gramian-vector product	it$_{\text{CG}}$	$\mathcal{O}(3(RL)^2 I + 3(RL)^3)$

From the LS solution and the chain rule, we obtain after simplification:

$$\mathbf{c} = \text{unvec}\left(\text{vec}(\mathbf{M}')^{\mathrm{T}}\hat{\mathbf{Q}}\hat{\mathbf{W}}^{-1}\right)^{\mathrm{T}} \left(\mathbf{q}^{\mathrm{T}}/\|\mathbf{q}\|_2\right) , \qquad (3)$$

where $\hat{\mathbf{Q}} = (\mathbf{B}' \odot \mathbf{A}')$, $\hat{\mathbf{W}} = \hat{\mathbf{Q}}^{\mathrm{T}}\hat{\mathbf{Q}} = \left((\mathbf{A}'^{\mathrm{T}}\mathbf{A}') * (\mathbf{B}'^{\mathrm{T}}\mathbf{B}')\right)$, and unvec$(\cdot)$ reorders the $(RL \times 1)$-matrix into an $(R \times L)$-matrix. The pseudoinverse of a vector is cheap to compute and the Khatri-Rao structure of $\hat{\mathbf{Q}}$ can be exploited when computing vec$(\mathbf{M}')^{\mathrm{T}}\hat{\mathbf{Q}}$ [21].

To make the method perform in non-stationary environments, past slices can gradually be forgotten using a sliding window. This can be handled by simply removing the first row of $\overline{\mathbf{Z}}$ before the next updating step. An exponential window can be applied by multiplying the rows of $\mathbf{Q}$ with the appropriate scaling terms.

2.5. Complexity analysis

The per-iteration complexity of the algorithm is listed in Table 1 for a tensor $\mathcal{T} \in \mathbb{R}^{I \times J \times K}$, with $I = J = K$. The expressions are similar to those in [13], but in our case, the factor matrices have RL columns instead of R, the gradients require an extra multiplication and the evaluations of the preconditioned matrix-vector products require $3(RL)^2 \max(I, J, K) + 3(RL)^3$ flops. The weight tensor is updated using the NLS CPD updating method from [13], so the computation cost is dominated by that of the WLS updating step.

The method requires the storage of the new data and weighting slices ($\mathcal{O}(IJ)$), factor matrices, gradients and Gramian-vector products (all $\mathcal{O}(RL(I + J + K))$). Keeping the $\mathbf{M}^A$-type matrices in memory is not required. In contrast, only storing the full tensor would already require $\mathcal{O}(IJK)$ memory.

3. DIRECTION OF ARRIVAL ESTIMATION (DOA)

Direction-of-arrival (DOA) estimation problems arise frequently in the field of array processing [22]. It has been demonstrated that incorporating prior knowledge about the sensor accuracies by use of a low-rank weight tensor can improve the performance of DOA-methods [12]. We show that our proposed method can achieve a higher accuracy than standard LS updating when sensor conditions are non-constant, e.g., a sensor breaks down unexpectedly or the environment temperature changes, while being more efficient than batch WLS. The experiment considers the case of line-of-sight signals that impinge upon a uniform rectangular array (URA). The CPD can be applied for DOA-retrieval in this case [23–26].

Our URA has $M \times M$ sensors, each acquiring K samples from R different moving sources in the far field. The omnidirectional sensors are evenly spaced with inter-sensor spacing Δ. The azimuths $\overline{\mathbf{Z}} \in \mathbb{C}^{M \times R}$ and elevations $\mathbf{L} \in \mathbb{C}^{M \times R}$ of the R sources at each time step lead to an observed tensor $\mathcal{T} \in \mathbb{C}^{M \times M \times K}$, perturbed by noise, for which the kth frontal slice admits a low rank approximation $\mathcal{T} \approx [\![\mathbf{A}^{(k)}, \mathbf{E}^{(k)}, \mathbf{s}^{(k)}]\!]$. The matrix $\mathbf{A}^{(k)} \in \mathbb{C}^{M \times R}$ has entries $a_{mr}^{(k)} = \exp((m-1)2\pi/\lambda \sin(z_{rk}\pi/180)\Delta\mathrm{i})$ and $\mathbf{E}^{(k)} \in$

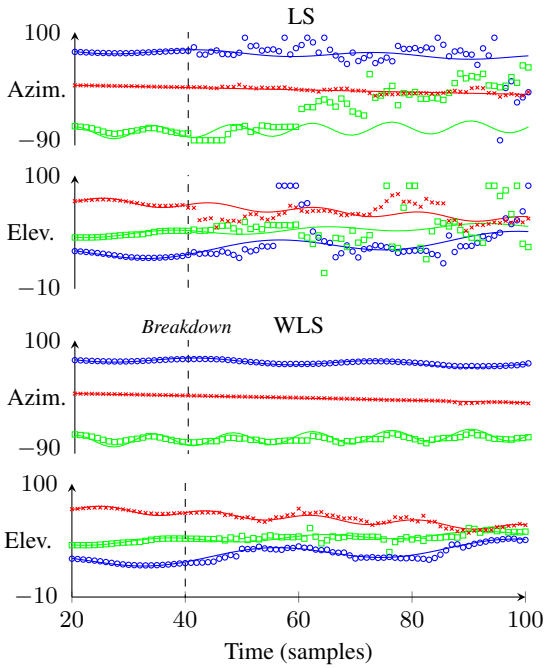


Fig. 2. A low-rank weight tensor can improve DOA estimation by giving lower importance to values obtained by broken sensors. Three sources are tracked during 80 samples and their DOAs (full line) and estimated DOAs (marks) are displayed. Tracking clearly fails after the sensor breakdown at $t = 40$ samples for the LS updating method (top two figures), while the WLS updating method (bottom two figures) continues to obtain good estimates despite the breakdown.

Table 2: The median absolute errors of the azimuths and elevations are much smaller if a low-rank weight tensor is applied.

	LS			WLS		
Azimuth	10.788	1.228	16.436	0.652	0.340	0.976
Elevation	3.898	2.311	3.548	0.297	0.270	0.250

Table 3: The updated WLS-method requires less CPU-time (s) than its batch counterpart.

Tensor dimensions	WLS Updating	Batch WLS
$400 \times 400 \times 400$	10.5	14.6
$500 \times 500 \times 500$	20.6	29.1
$600 \times 600 \times 600$	34.7	49.1
$700 \times 700 \times 700$	55.3	110
$800 \times 800 \times 800$	81.9	574

$\mathbb{C}^{M\times R}$ has entries $e_{mr}^{(k)} = \exp((m-1)2\pi/\lambda \sin(l_{rk}\pi/180)\Delta \mathrm{i})$, with i the imaginary unit. The vector $\mathbf{s}^{(k)} \in \mathbb{C}^{1\times R}$ contains the sources and λ is the wavelength. With a rank-R CPD of a few frontal slices of $\mathcal{T}$, the positions of the sources can be recovered.

We choose a URA with $M^2 = 400$ sensors and determine the DOA of $R = 3$ sources with oscillating trajectories. $K = 100$ samples are collected and in each time step, the CPD of the observed tensor is updated to incorporate the new data. We assume that the sensors obtain samples with an SNR of 20 dB. We further assume that sensors 26 to 185 break down after 40 samples, which results in their sample SNR dropping to -14 dB. We choose a weight tensor $\mathcal{W}$, where all w_{ijk} are set to 1, except for the entries where $26 \leq 10(i-1)+j \leq 185$ & $k > 40$. These are set to 0.02 to reflect the relative difference in SNR between the sensors after the breakdown. These weights were obtained by a heuristic approach, but optimal weights can be obtained from the Fisher information. Note that $\mathcal{W}$ has rank 3, but the parts before and after the breakdown are rank-1 and rank-2, respectively, so we start with a rank-1 $\mathcal{W}$ and update it to a rank-3 (at slice 40) and finally a rank-2 (at slice 46) CPD.

We compare LS updating methods for the weighted and non-weighted case. The updating methods start from a rank-3 CPD of the first 20 slices of $\mathcal{T}$ and update the CPD after every new sample slice is added to the tensor, using a sliding window of 6 slices, to estimate the mean DOAs during the last 6 time steps. The updating methods are limited to $\mathrm{it}_{\mathrm{GN}} = \mathrm{it}_{\mathrm{CG}} = 5$. The memory requirements of the updating methods are much smaller than those of their batch counterparts, as only one slice has to be stored alongside the old CPD. In Table 2, we report the median across 100 trials of the median absolute estimation errors during the updating process. It is clear that the accuracy of the DOA estimation improves by applying the low-rank weight tensor $\mathcal{W}$ to the problem, as the WLS updating method outperforms the LS updating method significantly. In Figure 2, one trial of the experiment is shown. One can again see that it is beneficial to apply a low-rank weight tensor to the CPD computation. Unlike the LS updating method, which fails after sensor breakdown, the estimates of the WLS updating method remain good.

For the relatively small third-order tensors from the DOA experiment, time gains are rather limited. In Table 3, we report timing results for large-scale tensors. The listed timings (in seconds) are for a one-slice update in the third mode of a rank-5 CPD, weighted with a rank-2 weight tensor. All methods are limited to $\mathrm{it}_{\mathrm{GN}} = \mathrm{it}_{\mathrm{CG}} = 5$. The CPDs are computed with an Intel Core i7-6820HQ CPU at 2.70GHz and 16GB of RAM using MATLAB R2016b and Tensorlab 3.0. It can be seen that the updating method outperforms its batch counterpart for sufficiently large tensors, with memory requirements that are negligible compared to those of the batch method.

4. CONCLUSION AND FURTHER WORK

A new updating method for CPD updating with low-rank weights is proposed. We exploit the CPD structure of both the weight and data tensor to efficiently compute a new CPD when new slices are added to the data tensor. Expressions are derived for the objective function, gradients and gradient-vector products of the NLS algorithm, along with an analysis of their complexity. We show that WLS CPD updating with low-rank weights outperforms the standard LS algorithm for DOA-estimation. Its accuracy is close to that of the batch WLS method, while memory requirements are much smaller.

The low-rank weight tensor can be updated while new samples are added to the tensor. Aside from using prior information, as we did in the DOA example, one could also modify the weight tensor in a data-driven way. This could, for instance, be done by either increasing or decreasing the weight of tensor entries that are badly explained by the model. Further, as the updating algorithm can handle rank changes, automatic rank-detection would be very useful to adapt the rank of the CPD of both the weight tensor and the data tensor to new data during the updating process. In this article, we focus on the case of third-order tensors, but extensions to higher-order tensors are possible. For these tensors, the efficiency gains in memory and computation time become even more significant.

5. REFERENCES

[1] 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.

[2] A. Cichocki, C. Mandic, A.H. 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.

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

[4] 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.

[5] 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.

[6] 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.

[7] 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.

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

[9] 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.

[10] P. Paatero, "A weighted non-negative least squares algorithm for three-way PARAFAC factor analysis," *Chemometrics and Intelligent Laboratory Systems*, vol. 38, no. 2, pp. 223–242, 1997.

[11] G. Hollander, P. Dreesen, M. Ishteva, and J. Schoukens, "Approximate decoupling of multivariate polynomials using weighted tensor decomposition," *Numerical Linear Algebra with Applications*, p. e2135, 2017.

[12] M. Boussé and L. De Lathauwer, "Nonlinear least squares algorithm for canonical polyadic decomposition using low-rank weights," in *IEEE 7th International Workshop on Computational Advances in Multi-Sensor Adaptive Processing (CAMSAP17)*, December 2017, pp. 39–43.

[13] M. Vandecappelle, N. Vervliet, and L. De Lathauwer, "Nonlinear least squares updating of the canonical polyadic decomposition," in *Proceedings of the 2017 25th European Signal Processing Conference (EUSIPCO2017)*, August 2017, pp. 693–697.

[14] 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.

[15] 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.

[16] 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.

[17] N. Vervliet, O. Debals, and L. De Lathauwer, "Exploiting efficient representations in tensor decompositions," *Technical Report 16-174, ESAT-STADIUS, KU Leuven, Belgium*, 2016.

[18] 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`.

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

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

[21] 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.

[22] H. Krim and M. Viberg, "Two decades of array signal processing research: the parametric approach," *IEEE signal processing magazine*, vol. 13, no. 4, pp. 67–94, 1996.

[23] N. D. Sidiropoulos, R. Bro, and G. B. Giannakis, "Parallel factor analysis in sensor array processing," *IEEE Transactions on Signal Processing*, vol. 48, no. 8, pp. 2377–2388, 2000.

[24] M. Li, Z. Li, and A. V. Vasilakos, "A survey on topology control in wireless sensor networks: Taxonomy, comparative study, and open issues," *Proceedings of the IEEE*, vol. 101, no. 12, pp. 2538–2557, 2013.

[25] R. Roy and T. Kailath, "ESPRIT-estimation of signal parameters via rotational invariance techniques," *IEEE Transactions on Acoustics, Speech, and Signal Processing*, vol. 37, no. 7, pp. 984–995, 1989.

[26] N. D. Sidiropoulos, "Generalizing Carathéodory's uniqueness of harmonic parameterization to N dimensions," *IEEE Transactions on Information Theory*, vol. 47, no. 4, pp. 1687–1690, 2001.