Low Multilinear Rank Updating

Michiel Vandecappelle*†, and Lieven De Lathauwer*†

***Abstract*—A low multilinear rank approximation (LMLRA) of a tensor is often used to compress a large tensor into a much more compact form, while still maintaining most of the information in the tensor. This can be achieved by only storing the principal subspaces in every mode and dropping the other singular vectors. These subspaces are then combined using a core tensor of (much) smaller dimensions than the original tensor. When the tensor is non-static, for example when new tensor slices are regularly added in one mode, one can expect that the LMLRA of the tensor changes only slightly and an approximation of the new tensor can be derived from the previous one. In this paper, a method is derived to track the different subspaces of a tensor by generalizing the rank-adaptive SURV method of Zhou et al. for tracking matrix subspaces to higher orders. Using a sequential truncation approach, this leads to efficient and accurate updates of the LMLRA.**

I. Introduction

Tensors, which extend vectors and matrices to higher order, have been steadily gaining in popularity and have been used extensively in applications in signal processing and machine learning [1], [2]. They provide a natural and convenient way to represent higher-order data without destroying its multilinear structure. As tensors can easily become impractically large, a lot of work has focused on the efficient computation of tensor decompositions. Like their matrix counterparts, these decompositions provide a way to extract meaningful components from tensors or to store large tensors compactly, while retaining the most important features of the data. Some well-known decompositions are the Canonical Polyadic Decomposition (CPD) [3], the Tensor-Train decomposition (TTD) [4] and the Low Multilinear Rank Approximation (LMLRA) [5], of which the Multilinear Singular Value Decomposition (MLSVD) [6] is a special case. It decomposes an Nth-order tensor into N orthogonal factor matrices, which are linked by a core tensor, see Figure 1. Even though the Eckart–Young theorem cannot be extended to the MLSVD, truncating this decomposition still leads to an approximation of the tensor which is very close to optimal for the given dimensions. See [7] and references therein for a discussion of this approximation accuracy. Moreover, the numerical properties of the MLSVD computation benefit greatly from the orthogonality of the factors.

Funding: Michiel Vandecappelle is supported by an SB Grant from the Research Foundation–Vlaanderen (FWO). Research furthermore supported by: (1) Flemish Government: This work was supported by the Fonds de la Recherche Scientifique–FNRS and the Fonds Wetenschappelijk Onderzoek–Vlaanderen under EOS Project no 30468160 (SeLMA) and under the "Onderzoeksprogramma Artificiële Intelligentie (AI) Vlaanderen" programme; (2) KU Leuven Internal Funds C16/15/059.

*KU Leuven, Dept. of Electrical Engineering ESAT/STADIUS, Kasteelpark Arenberg 10, bus 2446, B-3001 Leuven, Belgium

† Group Science, Engineering and Technology, KU Leuven – Kulak, E. Sabbelaan 53, 8500 Kortrijk, Belgium

(Michiel.Vandecappelle, Lieven.DeLathauwer)@kuleuven.be.

In real-life applications, tensors are often non-static. New data can become available over time and should be appended to the current tensor or old tensor values become outdated and are replaced by newer measurements. When time is limited, it can be prohibitively expensive to recompute a tensor decomposition when only a small subset of the tensor entries are modified. When the full tensor is not stored due to its size, this is impossible altogether. This shows the need for methods that can include the new information into the existing tensor decomposition while keeping execution time and memory requirements as low as possible. The truncated MLSVD (TMLSVD) is a particularly interesting decomposition to track, as it is a natural higher-order extension of the truncated SVD and is extensively used in tensor applications. The TMLSVD can offer a large compression ratio, while still giving a good approximation of the tensor. Updating this decomposition is also attractive from a computational point of view, as a major part of the update can be achieved using matrix subspace tracking methods, which have been extensively studied. See for example [8] for an overview of existing methods. We focus on the case where new slices are added to the tensor in a certain mode, see Figure 2. To better handle sudden changes in the signal subspaces, a sliding window is applied to downdate older slices. It is also desirable to be able to adapt the dimensions of the subspaces automatically. Many fast matrix subspace updating methods, such as PAST/PASTd [9] and FAPI [10] do not allow this. Instead, we will update a rank-revealing URV decomposition for every non-evolving mode, as this approach is particularly suited for sliding window adaptive rank updates.

Tensor subspace tracking when slices are added has been proposed before. In [11], [12], all subspaces of a tensor are tracked simultaneously. These methods neglect the multilinear structure of the problem by not truncating the new tensor slice after each mode is processed. They also require the storage of large intermediate matrices, making them unsuitable for large scale applications. The method from [13] was designed to estimate the subspaces of a tensor that differs only slightly from a tensor for which the subspaces are known. As such, it can be used for streaming applications. In [14], a general framework is proposed to track the different subspaces of a tensor. Only the principal subspaces in every mode are stored between updates, as well as a matricized version of the core tensor. As this framework assumes that the updates of the subspaces happen in parallel, no sequential truncating strategies are employed when moving from one subspace to the next, although these can significantly speed up the algorithm in non-parallel settings.

In this article, we will propose a method that allows to

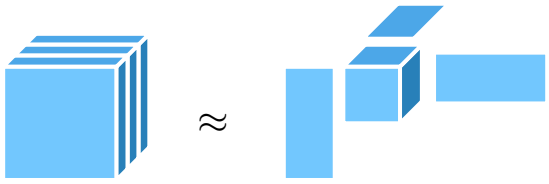


Figure 1. TMLSVD of a third-order tensor. The tensor is decomposed into three orthogonal factor matrices, linked by a small core tensor.

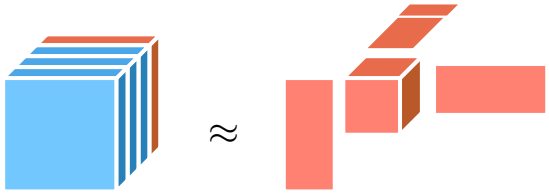


Figure 2. Update of the MLSVD of a third-order tensor. The three factor matrices holding the subspaces are changed slightly and the third factor matrix is extended in the evolving direction. The core tensor changes as well.

efficiently track a TMLSVD of a tensor along a sliding window. The dimension of the subspace of every mode is adjusted automatically whenever this is required. The algorithm is significantly faster than batch methods for large tensor dimensions, yet its memory requirements are much lower. In Section II, the algorithm is laid out and its complexity is evaluated. Setting good threshold values for the method is discussed in Section III. Finally, Section IV shows some experiments on synthetic data. The algorithm is laid out hereafter for a third-order tensor that is updated in the third mode, but evidently, any mode can be chosen as the updating mode. Extending the method to higher orders is straightforward, as this only increases the number of non-evolving modes.

A. Notation and terminology

Scalars, vectors and matrices are denoted by lowercase (a), bold lowercase ($\mathbf{a}$) and bold uppercase letters ($\mathbf{A}$), respectively, while tensors are written in calligraphic script ($\mathcal{A}$). The Kronecker product of matrices is denoted by $\otimes$. The notation $\mathbf{U}_{j\otimes i}$ is a shorthand for $\mathbf{U}_j \otimes \mathbf{U}_i$. When using subscripts to specify certain entries of a tensor $\mathcal{T}$, a colon refers to all entries of a specific mode, e.g. $\mathbf{t}_{1:2}$ is a vector holding all entries of the third-order tensor $\mathcal{T}$ which have 1 and 2 as their first and third index, respectively. Vectorization of the tensor $\mathcal{X} \in \mathbb{R}^{I\times J\times K}$ maps $x_{i_1 i_2 i_3}$ to $\text{vec}(\mathcal{X})_q$, with $q = i_1 + (i_2-1)I + (i_3-1)IJ$. The inverse is denoted by $\text{unvec}(\mathbf{x})$. A mode-n fiber of a tensor is a vector obtained by fixing all indices but the nth. The mode-n unfolding $\mathbf{T}_{(n)}$ of a tensor $\mathcal{T}$ is constructed by stacking all its mode-n fibers as columns of a matrix. For example: $t_{i_1 i_2 i_3}$ is mapped to $\left(\mathbf{T}_{(2)}\right)_{i_2,q}$, with $q = i_1 + (i_3-1)I$. The mode-n tensor-matrix product of a tensor $\mathcal{T} \in \mathbb{R}^{I_1\times I_2\times I_3}$ and a matrix $\mathbf{X} \in \mathbb{R}^{J\times I_n}$ is written as $\mathcal{T} \cdot_n \mathbf{X}$. For $n = 1$, 2 and 3, $\mathcal{T} \cdot_n \mathbf{X} \in \mathbb{R}^{J\times I_2\times I_3}$, $\mathbb{R}^{I_1\times J\times I_3}$ and $\mathbb{R}^{I_1\times I_2\times J}$, respectively. In unfolded form, computing the mode-n product is the same as computing $\mathbf{X}\mathbf{T}_{(n)}$.

A low multilinear rank approximation (LMLRA) of a tensor $\mathcal{T} \in \mathbb{R}^{I\times J\times K}$ is defined as $\mathcal{T} \approx \mathcal{S} \cdot_1 \mathbf{U}_1 \cdot_2 \mathbf{U}_2 \cdot_3 \mathbf{U}_3$, where $\mathcal{S} \in \mathbb{R}^{R_1\times R_2\times R_3}$, $\mathbf{U}_1 \in \mathbb{R}^{I\times R_1}$, $\mathbf{U}_2 \in \mathbb{R}^{J\times R_2}$, $\mathbf{U}_3 \in \mathbb{R}^{K\times R_3}$. The vector (R_1, R_2, R_3) with $R_1 \leq I$, $R_2 \leq J$, $R_3 \leq K$ holds the number of (linearly independent) columns that are used in the factor matrices $\mathbf{U}_i$ of the approximation for every mode. If the approximation is exact and the $R^{(i)}$ are minimal, then this vector is called the multilinear rank of the tensor. By restraining the $\mathbf{U}^{(i)}$ to be orthogonal and $\mathcal{S}$ to be all-orthogonal, a truncated MLSVD is obtained [6]. The factor matrices of the TMLSVD can be obtained by computing the principal column spaces of all mode-n unfoldings of the tensor. The core tensor $\mathcal{S}$ can then be found by forming $\mathcal{S} = \mathcal{T} \cdot_1 \mathbf{U}_1{}^{\mathrm{T}} \cdot_2 \mathbf{U}_2{}^{\mathrm{T}} \cdot_3 \mathbf{U}_3{}^{\mathrm{T}}$.

II. LMLRA Updating

Assume that one has a matrix $\mathbf{X} \in \mathbb{R}^{I\times J}$, where every column $\mathbf{x}_{:j}$ represents a sample obtained at time j. Using a low rank matrix approximation method, one can then find matrices $\mathbf{P} \in \mathbb{R}^{I\times R}$ and $\mathbf{Q} \in \mathbb{R}^{J\times R}$ such that $\mathbf{X} \approx \mathbf{P}\mathbf{Q}^T$. Every time sample $\mathbf{x}_{:j}$ can then be represented as $\mathbf{P}\mathbf{q}_{j:}^{\mathrm{T}}$, as shown in Figure 3 (top). The other way around, computing $\mathbf{P}^{\mathrm{T}}\mathbf{x}_{:j}$ gives a low dimensional representation of the time sample $\mathbf{x}_{:j}$. When new time samples are added as new columns to the matrix $\mathbf{X}$, the matrix $\mathbf{P}$ has to be updated so that it represents the principal column space of the new $\mathbf{X}$. Using this updated $\mathbf{P}$, we can again find low rank approximations of any time sample in $\mathbf{X}$.

If the time sample is not a vector, but a matrix or even a tensor, the matrix tracking problem above can be extended to a tensor tracking problem. By keeping the tensor structure intact, one can often obtain more meaningful low rank approximations than when matrix methods are used. Let $\mathcal{T} \in \mathbb{R}^{I\times J\times K}$ be a tensor where the frontal slice $\mathbf{t}_{::k}$ represents the sample obtained at time k. A low multilinear rank approximation of $\mathcal{T}$ can now be obtained by computing a TMLSVD of $\mathcal{T}$, where the subspaces of the first two modes are truncated so that they only hold the principal mode-1 and mode-2 subspaces, respectively. This leads to the following decomposition: $\mathcal{T} = \mathcal{S} \cdot_1 \mathbf{U}_1 \cdot_2 \mathbf{U}_2 \cdot_3 \mathbf{U}_3$, or, unfolded in the third mode, $\mathbf{T}_{(3)}^{\mathrm{T}} = \mathbf{U}_{2\otimes 1}\mathbf{S}_{(3)}^{\mathrm{T}}\mathbf{U}_3^{\mathrm{T}}$, as shown in Figure 3 (bottom). The same interpretation can be given as in the matrix case: computing $\mathbf{U}_{2\otimes 1}^{\mathrm{T}}\text{vec}(\mathbf{T}_{::k})$ yields a low dimensional representation of the time sample $\mathbf{T}_{::k}$. When new time samples are added to the tensor, the mode-1 and mode-2 principal subspaces have to be updated in a stable and efficient way to include these new samples. If old time samples are to be forgotten, the subspaces should be adjusted accordingly as well.

When a tensor $\mathcal{T} \in \mathbb{R}^{I\times J\times K}$ is tracked with a sliding window of length w, new slices are added and removed in a certain mode, for example mode 3. This mode will be referred to as the evolving mode. All other modes are called non-evolving modes. Only after adding the first w slices, do we start to remove the oldest slice. From then on, the tracked tensor is always of dimensions $(I \times J \times w)$. When no slices are removed, the window is said to be infinite and the tracked tensor at update n has dimensions $(I \times J \times n)$. For both

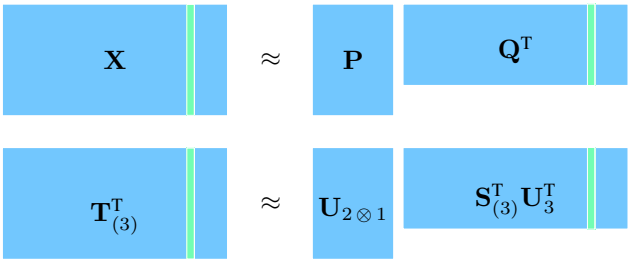


Figure 3. Top: Low rank matrix approximation of a matrix $\mathbf{X}$ as the product of two matrices $\mathbf{P}$ and $\mathbf{Q}^{\mathrm{T}}$. One time-sample is highlighted in $\mathbf{X}$ and in the sample matrix $\mathbf{Q}^{\mathrm{T}}$. Bottom: Higher-order extension of the factorization above. The samples are now frontal slices of a tensor $\mathcal{T}$ and the low multilinear rank approximation is now done in the first two modes with a TMLSVD, leading to a factorization $\mathbf{T}_{(3)}^{\mathrm{T}} = \mathbf{U}_{2\otimes 1}\mathbf{S}_{(3)}^{\mathrm{T}}\mathbf{U}_3^{\mathrm{T}}$. As for the matrix case, one time sample is highlighted in $\mathbf{T}_{(3)}^{\mathrm{T}}$ and in the sample matrix $\mathbf{S}_{(3)}^{\mathrm{T}}\mathbf{U}_3^{\mathrm{T}}$.

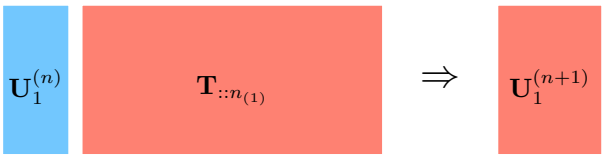


Figure 4. Updating of the TMLSVD for the first mode. The new columns of the mode-1 unfolding are included in the subspace $\mathbf{U}_1^{(n)}$.

cases, we will denote the tensor dimensions as $(I \times J \times N)$. Superscripts of objects refer to the index of the update in which they have been computed, so $\mathbf{X}^{(n)}$ refers to a matrix that was computed in the nth updating step. To perform an update (and/or downdate) of the TMLSVD, the subspaces (the principal column spaces of the unfoldings of $\mathcal{T}$), in the non-evolving modes have to be updated, as shown in Section II-A. The storage and computational complexities of the algorithm are compared to those of the batch method in Section II-B.

A. Multilinear subspace tracking

For mode 1, the goal is to track the column space of the mode-1 unfolding $\mathbf{T}_{(1)}$ of $\mathcal{T}$. Let us assume that this column space has dimension $R_1^{(n)}$ after n slices have been added in the third mode, with $1 \leq R_1^{(n)} \leq I$. Whenever a new slice $\mathbf{T}_{::n+1} \in \mathbb{R}^{I\times J\times 1}$ is added to the tensor, this corresponds to adding all the mode-1 fibers of the slice as columns to $\mathbf{T}_{(1)}^{(n)}$, see Figure 4. Thus, the first mode subspace should take every mode-1 fiber of the new slice $\mathbf{T}_{::n+1}$ into account. Likewise, when an old slice is removed from the tensor, the contributions of all mode-1 fibers of that old slice should be removed.

As only the subspace of $\mathbf{T}_{(1)}$ has to be tracked, and not the singular values, any subspace-revealing decomposition can be updated instead of the SVD. The URV decomposition is used below, where $\mathbf{U}$ and $\mathbf{V}$ are orthogonal and $\mathbf{R}$ is upper or lower triangular. The URV decomposition can be updated in a stable and efficient way using for instance the Signed URV (SURV) method of Zhou et al. [15]. In essence, a decomposition $\mathbf{Q}^{(n)}\mathbf{R}^{(n)}{\mathbf{\Phi}^{(n)}}^{\mathrm{T}} = \left[\mathbf{N}_1^{(n)}|\mathbf{S}_1^{(n)}\right]$ is tracked, where $\mathbf{S}_1^{(n)} \in \mathbb{R}^{I\times R_1^{(n)}}$ holds an orthonormal basis for the column space of $\mathbf{T}_{(1)}^{(n)}$ and $\mathbf{N}_1^{(n)} \in \mathbb{R}^{I\times(I-R_1^{(n)})}$ holds an orthonormal basis for the orthogonal complement, which could be used in future updates. Note that $\mathbf{S}_1^{(n)}$ is on the right of $\mathbf{N}_1^{(n)}$ to follow the convention from [15]. The (possibly) very long matrix $\mathbf{\Phi}^{(n)} \in \mathbb{R}^{Jw\times R_1^{(n)}}$, the size of which depends on the window length w, is never stored during the updating process, making the algorithm particularly memory-efficient. This algorithm applies both hyperbolic rotations and Givens row and column rotations to obtain the upper-triangular structure of $\mathbf{R}^{(n)}$, while being particularly careful that only stable hyperbolic rotations are used. The dimension of the subspace is automatically adjusted if necessary. This is done by only allowing subspace vectors that contribute to the subspace by more than a predefined threshold value $\gamma^{(1)}$ to be in $\mathbf{S}_1^{(n+1)} \in \mathbb{R}^{I\times R_1^{(n+1)}}$, while keeping the others in $\mathbf{N}_1^{(n+1)} \in \mathbb{R}^{I\times(I-R_1^{(n+1)})}$. The mode-1 factor matrix $\mathbf{U}_1^{(n+1)}$ of the new TMLSVD is simply equal to $\mathbf{S}_1^{(n+1)}$.

The new mode-1 factor matrix $\mathbf{U}_1^{(n+1)}$ can be used to compress the new slice $\mathbf{T}_{::n+1} \in \mathbb{R}^{I\times J\times 1}$ to its principal subspace in the first mode. As a result, one obtains $\tilde{\mathbf{T}}_{::n+1} = \mathbf{T}_{::n+1} \cdot_1 {\mathbf{U}_1^{(n+1)}}^{\mathrm{T}} \in \mathbb{R}^{R_1^{(n+1)}\times J\times 1}$ which is a compressed instance of $\mathbf{T}_{::n+1}$. This compressed slice can then be used for the following non-evolving modes, making these modes considerably less costly to update. The batch equivalent of this sequentially truncated method is described in [16]. The same approach can be used for the slice that is to be downdated, but using the old factor matrix $\mathbf{U}_1^{(n)}$ instead of the newly computed $\mathbf{U}_1^{(n+1)}$. It is beneficial to order the updates of the non-evolving modes from the one with the smallest dimension up to the one with the largest dimension to maximize the compression in each mode, as explained in Section II-B below.

B. Complexity

An updating method should have a lower time and memory complexity than its corresponding batch algorithm, as computation time and storage space are typically limited in tracking applications. Below, both are discussed.

a) Memory complexity: The tensor that is to be decomposed has IJN entries. For large values of I, J and N, storing and accessing this tensor in local memory can become too costly. As a result, batch methods to compute the MLSVD become infeasible. In essence, the proposed updating method also requires all slices within the sliding window to be stored, as these slices have to be downdated in later iterations. However, only the most recent and the oldest slice in the window are required for the current TMLSVD update and downdate. As such, the updating method requires much less data to be quickly accessible in local memory. The other slices can be pulled from long term memory when they are to be downdated. For the comparison, we will only consider the locally stored slices which have $2IJ$ entries in total. For a non-evolving mode, say mode 1, only the matrices $\mathbf{Q}^{(n)} \in \mathbb{R}^{I\times I}$ and $\mathbf{R}^{(n)} \in \mathbb{R}^{I\times I}$ have to be stored. In total, the storage requirements are $\mathcal{O}(2(I^2+J^2+IJ)$, which is generally much smaller than the $\mathcal{O}(IJN)$ of the full tensor.

b) Computational complexity: For the first mode, the complexity of one subspace update is similar to that of a single QR-update ($\mathcal{O}(I^2)$ flops). Of course, every new slice adds J columns to $\mathbf{T}_{(1)}^{(n)}$, while removing one slice corresponds to removing J columns from $\mathbf{T}_{(1)}^{(n)}$, so the total complexity of one slice update or downdate for the first mode is $\mathcal{O}(I^2J)$. When the subspaces are relatively stable, one can often get away with sampling only a subset of these columns from the new/old slice because of the low rank assumption on the second mode. If q columns are sampled, the complexity of one slice update or downdate is thus $\mathcal{O}(qI^2)$ flops. The second subspace has a complexity of $\mathcal{O}(J^2)$ flops per column update. Because of the sequential truncation after the computation of the first subspace, however, only $R_1^{(n+1)}$ columns have to be updated instead of J, bringing the complexity for the second subspace to $\mathcal{O}(J^2R_1^{(n+1)})$ flops. Recomputing the mode-1 and mode-2 subspaces whenever a sample is added or removed, would require $\mathcal{O}(I^2JN + J^2R_1^{(n+1)}N)$ flops. This is clearly higher than the operation count of the updating method.

To determine the ordering of the modes that offers the largest speedup, one can compare the operation counts of both possible orderings. Updating the second mode after the first one is faster if $I^2J + J^2R_1^{(n+1)} < J^2I + I^2R_2^{(n+1)}$. Defining the dimension reductions S_i for both modes as $S_1 = I - R_1^{(n+1)}$ and $S_2 = J - R_2^{(n+1)}$, the equation can be rewritten as $I^2J + J^2(I - S_1) < J^2I + I^2(J - S_2)$, or $J^2S_1 > I^2S_2$. When the values S_i lie close to each other, one should thus update the mode with the smallest dimension first.

III. Choosing the threshold values $\gamma^{(i)}$

The SURV decomposition adjusts the dimension of the subspace in the ith mode automatically through the value of the threshold $\gamma^{(i)}$. Whenever a singular value drops below this threshold, its corresponding vector is removed from the subspace and the dimension of the subspace is decreased. Depending on the expected noise level of the tensor, the value of $\gamma^{(i)}$ has to be chosen accordingly: the higher the noise, the larger the singular value threshold should be set. Zhou et al. propose to set the threshold equal to the estimated size of the largest singular value of a 'noise only' matrix, multiplied by a scaling factor $\alpha > 1$ to avoid false positives. When a singular value exceeds this threshold, its corresponding vector is considered to be part of the subspace. This leads to a threshold value of $\gamma^{(i)} = \alpha\sigma_n(\sqrt{m} + \sqrt{n})$, wherein σ_n is an estimate of the standard deviation of the noise and m and n are the dimensions of $\mathbf{T}_{(i)}$ after applying the sliding window. For example, for the first mode, $m = I$ and $n = wJ$.

IV. Experiments

First, a low multilinear rank tensor $\mathcal{T} \in \mathbb{R}^{50\times50\times800}$ is generated, where the first mode has rank 3 and the second mode has rank 4. The entries of the factor matrices and core tensor are drawn from the uniform distribution on $[-1, 1]$. In the third mode, the tensor is multiplied with a sinusoidal signal, i.e. the kth slice is multiplied by $\sin(\frac{k-1}{399}\pi)$. Then Gaussian distributed noise is added to the tensor with an SNR of 50dB and a TMLSVD of the tensor is tracked using a sliding window of length 200. In Figure 5, the results are shown. The subspaces and their dimensions are accurately estimated during the full updating process. Compared to the batch method, the updating method is faster after enough slices have been added to the tensor. The difference between both grows even more if the dimensions are chosen larger than in this experiment.

In a second experiment, two low multilinear rank tensors are appended in the third mode, with each tensor of dimensions $(50 \times 50 \times 800)$. The first tensor has rank 3 in the first mode and rank 4 in the second mode, while the second tensor has rank 8 in the first mode and rank 7 in the second. The entries of the factor matrices and core tensor are drawn from the uniform distribution on $[-1, 1]$ and Gaussian distributed noise with an SNR of 40dB is added to the tensor. The subspaces are updated with a window length of 200. In Figure 6, the true and estimated rank is shown for the first and second mode, along with the angles between the true and estimated subspaces. The computation times for the updating and batch methods are also displayed. The correct dimension is tracked for both subspaces during most of the updating process. When the sliding window moves into the second tensor and thus covers two tensors with low multilinear rank partially, the mode-n subspace dimension is estimated as approximately the sum of the mode-n rank of the first and the second tensor. The angles between the true and estimated subspaces are larger after the tensor switch, as it takes a while before the old slices are removed from the subspaces. After the window moves entirely into the second tensor, the error drops back to the same level as before the tensor switch. The computation time for the updating method is significantly lower than for the batch method in this experiment.

V. Conclusion

An updating method was proposed to track the TMLSVD of a tensor as new slices are added and removed in a certain mode. The algorithm tracks the subspaces while automatically detecting their dimension in a stable and efficient way, while requiring much less storage capacity than the tensor itself. The algorithm is particularly fast if the subspaces do not change much across updates, but even abrupt changes can be tracked.

In future work, tracking of a low dimensional subspace for the evolving mode will be included in the method, as well as an efficient way to update the core tensor. This can allow one to track a full TMLSVD of a tensor with a preset maximal error between updates, while still being efficient with respect to both storage and computation time.

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.

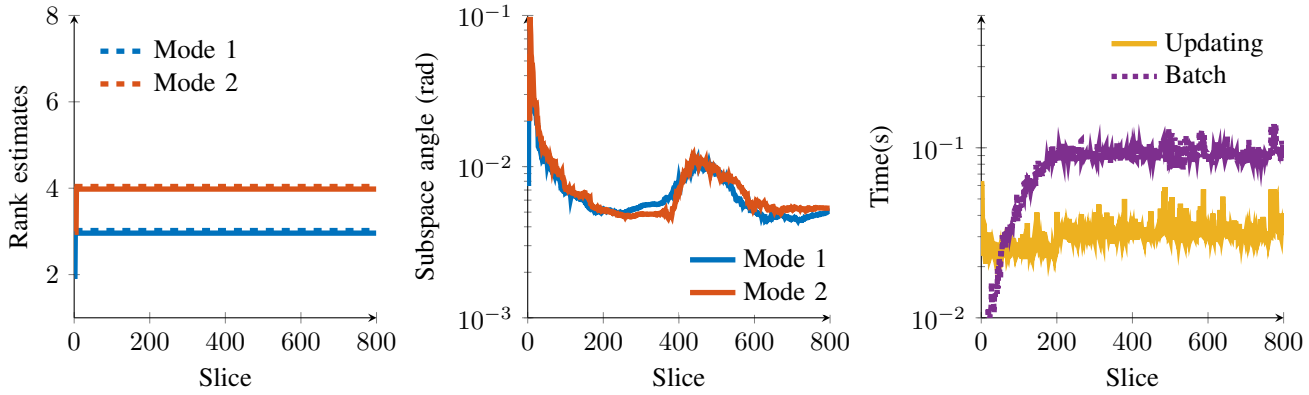


Figure 5. Tracking of the MLSVD of a $(50 \times 50 \times 800)$-tensor with low multilinear rank following a sinusoidal pattern in the third mode. Gaussian distributed noise is added with an SNR of 50dB. The window length is 200. True (dashed lines) and estimated (full lines) mode-n rank. Center: Angle between the true and estimated subspace. Right: Computation time for each update compared to recomputation of the full TMLSVD. The updating method outperforms the batch algorithm with respect to computation time, while obtaining good estimates for the subspaces and correctly estimating their dimensions.

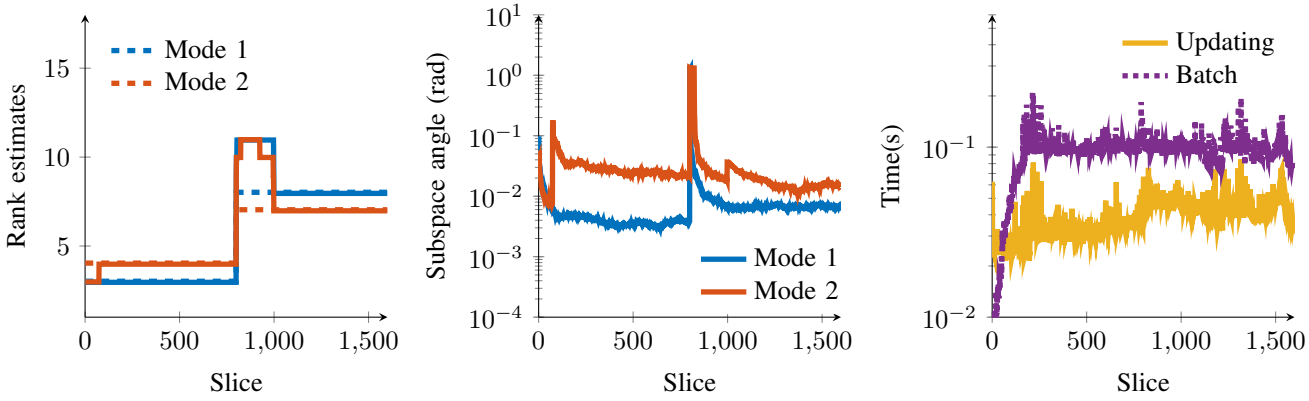


Figure 6. Tracking of the MLSVD of two $(50 \times 50 \times 800)$-tensors with low multilinear rank, appended in the third mode. Gaussian distributed noise is added with an SNR of 40dB. The window length is 200. Left: True (dashed lines) and estimated (full lines) mode-n rank. Center: Angle between the true and estimated subspace. Right: Computation time for each update compared to recomputation of the full TMLSVD. The updating method tracks the subspaces accurately, even when abrupt changes in the subspaces occur, while being much faster than repeatedly applying the batch method.

[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] F. L. Hitchcock, "The expression of a tensor or a polyadic as a sum of products," *Journal of Mathematical Physics*, vol. 6, no. 1–4, pp. 164–189, 1927.

[4] I. V. Oseledets, "Tensor-train decomposition," *SIAM Journal on Scientific Computing*, vol. 33, no. 05, pp. 2295–2317, 2011.

[5] L. R. Tucker, "Some mathematical notes on three-mode factor analysis," *Psychometrika*, vol. 31, no. 3, pp. 279–311, 1966.

[6] L. De Lathauwer, B. De Moor, and J. Vandewalle, "A multilinear singular value decomposition," *SIAM Journal on Matrix Analysis and Applications*, vol. 21, no. 4, pp. 1253–1278, 2000.

[7] L. De Lathauwer, B. De Moor, and J. Vandewalle, "On the best rank-1 and rank-$(R_1, R_2, \ldots, R_N)$ approximation of higher-order tensors," *SIAM Journal on Matrix Analysis and Applications*, vol. 21, pp. 1324–1342, 03 2000.

[8] J. P. Delmas, *Subspace Tracking for Signal Processing*. John Wiley & Sons, Ltd, 2010, ch. 4, pp. 211–270.

[9] B. Yang, "An extension of the PASTd algorithm to both rank and subspace tracking," *IEEE Signal Processing Letters*, vol. 2, no. 9, pp. 179–182, Sep. 1995.

[10] R. Badeau, B. David, and G. Richard, "Fast approximated power iteration subspace tracking," *IEEE Transactions on Signal Processing*, vol. 53, no. 8, pp. 2931–2941, Aug 2005.

[11] W. Hu, X. Li, X. Zhang, X. Shi, S. Maybank, and Z. Zhang, "Incremental tensor subspace learning and its applications to foreground segmentation and tracking," *International Journal of Computer Vision*, vol. 91, no. 3, pp. 303–327, Feb 2011.

[12] X. Ma, D. Schonfeld, and A. Khokhar, "Dynamic updating and downdating matrix SVD and tensor HOSVD for adaptive indexing and retrieval of motion trajectories," in *2009 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP 2009)*, Taipei, Taiwan, April 2009, pp. 1129–1132.

[13] 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, Oct 2008.

[14] Y. Cheng, F. Roemer, O. Khatib, and M. Haardt, "Tensor subspace tracking via Kronecker structured projections (TeTraKron) for time-varying multidimensional harmonic retrieval," *EURASIP Journal on Advances in Signal Processing*, vol. 2014:123, pp. 1–14, 11 2014.

[15] M. Zhou and A. van der Veen, "Stable subspace tracking algorithm based on a Signed URV decomposition," *IEEE Transactions on Signal Processing*, vol. 60, no. 6, pp. 3036–3051, June 2012.

[16] N. Vannieuwenhoven, R. Vandebril, and K. Meerbergen, "A new truncation strategy for the higher-order singular value decomposition," *SIAM Journal on Scientific Computing*, vol. 34, no. 2, pp. A1027–A1052, 2012.