Rank-one Tensor Approximation with Beta-divergence Cost Functions

Michiel Vandecappelle*†, Nico Vervliet*, and Lieven De Lathauwer*†

***Abstract*—β-divergence cost functions generalize three popular cost functions for low-rank tensor approximation by interpolating between them: the least-squares (LS) distance, the Kullback—Leibler (KL) divergence and the Itakura—Saito (IS) divergence. For certain types of data and specific noise distributions, beta-divergence cost functions can lead to more meaningful low-rank approximations than those obtained with the LS cost function. Unfortunately, much of the low-rank structure that is heavily exploited in existing second-order LS methods, is no longer exploitable when moving to general β-divergences. In this paper, we show that, unlike in the general rank-R case, rank-1 structure can still be exploited. We therefore propose an efficient method that uses second-order information to compute nonnegative rank-1 approximations of tensors for general β-divergence cost functions.**

***Index Terms*—tensors, β-divergences, low-rank, CPD, BSS**

I. Introduction

With the recent explosion of interest in data and data analytics, tensors have found their way into many applications in machine learning and signal processing [1], [2]. These higher-order generalizations of vectors and matrices represent higher-order datasets while preserving their higher-order structure. Unfortunately, as the number of entries of an Nth-order tensor with dimensions $I \times I \times \ldots \times I$ grows as I^N, its size quickly becomes problematic. By working with a low-rank approximation of the tensor instead of the full tensor, the required storage space and computation time can often be reduced dramatically, while still maintaining all the important features of the data in the tensor. A rank-R approximation of a tensor consists of R rank-1 terms, which themselves are tensors that can be written as an outer product of N nonzero vectors. To obtain a rank-R approximation, several algebraic and optimization-based methods have been developed [3]–[7].

By using β-divergences d_β as cost functions, the standard least-squares cost function can be generalized. These divergences, a special class of Bregman divergences [8], are defined for $\beta \in \mathbb{R}$ and interpolate continuously between the Itakura–Saito (IS) divergence ($\beta = 0$), Kullback–Leibler (KL) divergence ($\beta = 1$) and the least-squares distance ($\beta = 2$). When used as a cost function, values of $\beta < 2$ penalize errors on small entries more heavily compared to the least-squares distance, while the converse is true for values of $\beta > 2$. β-divergences are thus particularly useful when the data consists of entries of different magnitudes. As such, audio data is a prime example where β-divergence cost functions (with $\beta < 2$) outperform the least-squares cost function as these manage to capture the low intensity components of the signals better; see, for example, [9] and references therein for a profound discussion of the use of β-divergences in nonnegative matrix factorization (NMF) of audio spectra. Notably, an NMF with the KL- or IS-divergence corresponds to maximum likelihood estimation under the assumption of Poisson distributed data and data perturbed by multiplicative Gamma noise, respectively [10], while the least-squares cost function assumes additive independent and identically distributed (i.i.d.) Gaussian noise. The β-divergence $d_\beta(x,y)$ is defined as follows:

$$d_\beta(x,y) = \begin{cases} \frac{x^\beta + (\beta-1)y^\beta - \beta\left(xy^{(\beta-1)}\right)}{\beta(\beta-1)} & \beta \in \mathbb{R} \setminus \{0,1\} \\ x \ln\left(\frac{x}{y}\right) - x + y & \beta = 1 \\ \frac{x}{y} - \ln\left(\frac{x}{y}\right) - 1 & \beta = 0 \end{cases} \tag{1}$$

Where the least-squares distance considers only the absolute difference between a target value x and its estimate y, other β-divergences also take the size of x and y themselves into account. If $\beta = 0$, the divergence is scale invariant, meaning that $d_0(x, 2x)$ is the same for any value of x. If $\beta > 0$, $d_\beta(x, 2x)$ is larger for large values of x, while for $\beta < 0$, $d_\beta(x, 2x)$ is larger for small values of x.

A. Notation

We denote scalars, vectors, matrices and tensors by lowercase (a), bold lowercase ($\mathbf{a}$), bold uppercase ($\mathbf{A}$), and calligraphic letters ($\mathcal{A}$), respectively. For simplicity, we will only consider third-order tensors. Powers ($\mathbf{X}^{\bullet a}$), divisions and logarithms of matrices are entry-wise throughout the text. The Kronecker, column-wise Khatri–Rao, and Hadamard products of matrices and the outer product of vectors are denoted by $\otimes$, $\odot$, $*$ and $\circledcirc$ respectively, while the mode-n tensor contraction (i.e., tensor-vector product) of a tensor $\mathcal{T}$ and a vector $\mathbf{x}$ is written as $\mathcal{T} \cdot_n \mathbf{x}$. The transpose of a matrix $\mathbf{X}$ is denoted by $\mathbf{X}^{\mathrm{T}}$ and $\mathrm{diag}(\mathbf{x})$ forms a square matrix that has $\mathbf{x}$ as its

Funding: Michiel Vandecappelle is supported by an SB Grant from the Research Foundation–Vlaanderen (FWO). Nico Vervliet is supported by Internal Funds KU Leuven (PDM/18/146). 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); (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

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

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

diagonal. When using subscripts to specify certain entries of a tensor $\mathcal{T}$, a colon (:) refers to all entries of a specific mode. The 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 $q = i_1 + (i_2-1)I + (i_3-1)IJ$. Its 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 $(K\times K)$-identity matrix is written as $\mathbf{I}_K$ and $\mathbf{1}$ is a column vector of appropriate dimensions, consisting of only ones. For a rank-R approximation $\mathcal{T} \approx \sum_{r=1}^{R} \mathbf{a}_r \otimes \mathbf{b}_r \otimes \mathbf{c}_r$, the column vectors $\mathbf{a}_r$, $\mathbf{b}_r$ and $\mathbf{c}_r$ can be collected into factor matrices $\mathbf{A}$, $\mathbf{B}$ and $\mathbf{C}$. The rank-R approximation is then written as $\mathcal{T} \approx [\![\mathbf{A},\mathbf{B},\mathbf{C}]\!]$.

II. β-DIVERGENCE RANK-R APPROXIMATION

For tensor data, β-divergence cost functions have been used to compute nonnegative rank-R approximations of tensors. The proposed algorithms [11]–[14], however, are all either first-order methods, are alternating schemes, only consider KL- or IS-divergences, or are combinations of these. In contrast, we propose a method that uses second-order information to achieve near second-order convergence, is all-at-once and applicable for any value of β. For a tensor $\mathcal{T} \in \mathbb{R}^{I\times J\times K}$ and its low-rank approximation $\mathcal{M} = [\![\mathbf{A},\mathbf{B},\mathbf{C}]\!]$, the β-divergence cost function (1) leads to the following optimization problem: $\min_{\mathbf{A},\mathbf{B},\mathbf{C}} f(\mathbf{x})$, with $\mathbf{x} = [\text{vec}(\mathbf{A}); \text{vec}(\mathbf{B}); \text{vec}(\mathbf{C})]$ and

$$f(\mathbf{x}) = \frac{1}{\beta(\beta-1)} \sum_{n=1}^{N} t_n^\beta + (\beta-1)m_n^\beta - \beta\left(t_n m_n^{(\beta-1)}\right), \quad (2)$$

where $N = IJK$ and n runs over all tensor entries. Using (1), the cost functions for the degenerate special cases $\beta = 1$ (KL divergence) and $\beta = 0$ (IS divergence) are:

$$f(\mathbf{x}) = \sum_{n=1}^{N} t_n \ln\left(\frac{t_n}{m_n}\right) - t_n + m_n, \quad \text{and} \quad (3)$$

$$f(\mathbf{x}) = \sum_{n=1}^{N} \frac{t_n}{m_n} - \ln\left(\frac{t_n}{m_n}\right) - 1, \quad (4)$$

respectively. Note that the case $\beta = 2$ corresponds to the least-squares cost function. β-divergences are only defined for nonnegative values, so we constrain $\mathbf{A}$, $\mathbf{B}$ and $\mathbf{C}$ to be nonnegative matrices. For any value of β, the gradient $\mathbf{g} = [\mathbf{g}_\mathbf{A}; \mathbf{g}_\mathbf{B}; \mathbf{g}_\mathbf{C}]$ is:

$$\begin{aligned} \mathbf{g}_\mathbf{A} &= \text{vec}\left(\mathbf{R}_{(1)}(\mathbf{C}\odot\mathbf{B})\right) \quad (5)\\ \mathbf{g}_\mathbf{B} &= \text{vec}\left(\mathbf{R}_{(2)}(\mathbf{C}\odot\mathbf{A})\right)\\ \mathbf{g}_\mathbf{C} &= \text{vec}\left(\mathbf{R}_{(3)}(\mathbf{B}\odot\mathbf{A})\right), \end{aligned}$$

with $\mathcal{R} = \frac{\mathcal{M}-\mathcal{T}}{\mathcal{M}^{\bullet(2-\beta)}}$. Note that $\mathbf{g}$ can equivalently be written as $\mathbf{g} = \mathbf{J}^\text{T}\text{vec}(\mathcal{R})$, with $\mathbf{J}$ the Jacobian of the low-rank model $\mathcal{M}$ w.r.t. $\mathbf{A}$, $\mathbf{B}$, $\mathbf{C}$. Only $\mathcal{R}$ depends on the specific cost function that is chosen. For $\beta = 2$, $\mathcal{R}$ is simply the residual $\mathcal{M}-\mathcal{T}$. For a second-order method, (a good approximation of) the Hessian is required. In [7], an approximation of the form $\mathbf{J}^\text{T}\mathbf{Z}\mathbf{J}$ is proposed. This approximation is further generalized to arbitrary entry-wise cost functions in [15], using similar ideas as in [16]. The diagonal matrix $\mathbf{Z}$ has $\text{vec}(\mathcal{D})$ as its diagonal, where $\mathcal{D}$ is the entry-wise second-order derivative of the β-divergence cost function w.r.t. to $\mathcal{M}$:

$$\mathcal{D} = \frac{(\beta-1)\mathcal{M} - (\beta-2)\mathcal{T}}{\mathcal{M}^{\bullet(3-\beta)}} = \frac{(\beta-2)(\mathcal{M}-\mathcal{T}) + \mathcal{M}}{\mathcal{M}^{\bullet(3-\beta)}}. \quad (6)$$

Note that for $\beta \notin [1,2]$, $\mathcal{D}$ can have negative values when the low-rank approximation $\mathcal{M}$ is not close enough to the tensor $\mathcal{T}$. Consequently, the Hessian approximation is not guaranteed to be positive semidefinite (PSD), i.e., to have only eigenvalues ≥ 0. Positive semidefiniteness of the Hessian approximation guarantees that the local quadratic approximation of the cost function has a minimum. Without this property, one can end up in a saddle point instead. If $\mathbf{J}^\text{T}\mathbf{Z}\mathbf{J}$ is not PSD, one needs to use an optimization algorithm that can handle indefinite Hessian approximations. An alternative is to first find a good initialization, e.g., the solution for $\beta = 2$, before moving to the β-divergence algorithm. See [15] for an extended discussion. For $\beta = 2$, the matrix $\mathbf{Z}$ is equal to the identity matrix and one recovers the Gauss-Newton approximation $\mathbf{J}^\text{T}\mathbf{J}$ of the Hessian, which is the Gramian of the Jacobian.

Due to the shape of β-divergence cost functions, simply projecting the unconstrained Gauss–Newton point to the nonnegative orthant generally leads to poor results. In the kth iteration of the algorithm, the current model is $\mathbf{x}_k$ and a nonlinear projected gradient method is used to find the next step $\mathbf{p}$ such that $\mathbf{x}_k + \mathbf{p} = \mathbf{x}_{k+1}$. The cost function $f(\mathbf{x})$ is approximated by its second-order Taylor expansion $m(\mathbf{x})$. To assure convergence of the algorithm, we first find the Cauchy point $\mathbf{p}_C$ of $m(\mathbf{x})$ within an l_∞-norm trust region that incorporates the nonnegativity constraints on $\mathbf{x}$. Starting from this point, we then approximately solve the quadratic program $\min_\mathbf{p} m(\mathbf{x}+\mathbf{p})$ subject to $\mathbf{x}+\mathbf{p} \geq 0$ within the trust region and use $\mathbf{p}$ as the next step of the algorithm.

III. RANK-1 APPROXIMATIONS

The low-rank structure of the model can be exploited when $\beta = 2$. For example: products of the form $\mathbf{M}_{(1)}(\mathbf{C}\odot\mathbf{B})$ can be efficiently computed as $\mathbf{A}\left((\mathbf{B}^\text{T}\mathbf{B}) * (\mathbf{C}^\text{T}\mathbf{C})\right)$. Unfortunately, this structure is no longer exploitable in the computation of the objective function, gradient and Hessian approximation for other values of β. The model has to be expanded to the full tensor before computing products, sums or powers of its entries $m_n = \sum_{r=1}^{R} a_{ir}b_{jr}c_{kr}$, which negatively influences both time and storage complexity of the method. Fortunately, for $R = 1$, we have $\mathcal{M} = [\![\mathbf{a},\mathbf{b},\mathbf{c}]\!]$ and thus:

$$m_n^\beta = (a_i b_j c_k)^\beta = a_i^\beta b_j^\beta c_k^\beta, \quad (7)$$

which allows us to exploit the multilinear structure of the model and simplifies the computations significantly compared to the rank-R case. Dedicated, efficient expressions for the objective function, gradient and Hessian approximation are derived in this section. In various applications, e.g., [17], [18], one only needs a rank-1 approximation of a tensor in order to extract its most important component. Using similar ideas as in [19]–[21], the method could also be used to extract more than one nonnegative rank-1 term from a tensor.

A. Cost function

Writing $s_{\mathbf{x}}^{y} = \sum_{q=1}^{Q} x_q^y$ for a vector $\mathbf{x} \in \mathbb{R}^Q$, we can rewrite $f(\mathbf{x})$ from (2) to obtain the optimization problem $\min_{\mathbf{a},\mathbf{b},\mathbf{c}} f(\mathbf{x})$ with

$$f = \frac{1}{\beta(\beta-1)} \left[\left(\sum_{n=1}^{N} t_n^{\beta} \right) + (\beta-1) s_{\mathbf{a}}^{\beta} s_{\mathbf{b}}^{\beta} s_{\mathbf{c}}^{\beta} - \beta \mathcal{T} \cdot_1 \mathbf{a}^{\bullet(\beta-1)} \cdot_2 \mathbf{b}^{\bullet(\beta-1)} \cdot_3 \mathbf{c}^{\bullet(\beta-1)} \right]. \quad (8)$$

The first term $\left(\sum_{n=1}^{N} t_n^{\beta}\right)$ does not involve the model parameters and can be precomputed. Although adding this term does not change the optimal $\mathbf{a}$, $\mathbf{b}$ and $\mathbf{c}$, it gives us a frame of reference to assess the quality of the model, as it assures that $f = 0$ when $m_n = t_n, \forall n \in \{1, \ldots, N\}$. The second term follows from (7). It allows us to only compute the (non-integer) power of the entries of the three vectors $\mathbf{a}$, $\mathbf{b}$ and $\mathbf{c}$ instead of the power of all tensor entries m_n, which offers a considerable efficiency gain: $\mathcal{O}(I+J+K)$ flops and storage compared to $\mathcal{O}(IJK)$. In the last term, we again use (7) to avoid forming the full tensor $\mathcal{M}$ and computing (non-integer) powers of all its entries. For the KL-divergence, i.e., $\beta = 1$, the cost function (3) becomes:

$$\begin{aligned} f &= \left(\sum_{n=1}^{N} t_n \ln(t_n) - t_n \right) - \left(\sum_{n=1}^{N} t_n \ln(m_n) \right) + s_{\mathbf{a}}^{1} s_{\mathbf{b}}^{1} s_{\mathbf{c}}^{1} \\ &= \left(\sum_{n=1}^{N} t_n \ln(t_n) - t_n \right) - \mathbf{1}^{\mathrm{T}} \begin{bmatrix} \mathbf{T}_{(1)}^{\mathrm{T}} \ln(\mathbf{a}) \\ \mathbf{T}_{(2)}^{\mathrm{T}} \ln(\mathbf{b}) \\ \mathbf{T}_{(3)}^{\mathrm{T}} \ln(\mathbf{c}) \end{bmatrix} + s_{\mathbf{a}}^{1} s_{\mathbf{b}}^{1} s_{\mathbf{c}}^{1}. \end{aligned}$$

The first term can again be precomputed. For the second term, because of the property $\ln(abc) = \ln(a) + \ln(b) + \ln(c)$, we only need to compute logarithms of the three vectors $\mathbf{a}$, $\mathbf{b}$ and $\mathbf{c}$ instead of logarithms of all entries of $\mathcal{M}$. For the last term, the sum over all tensor entries is converted to three vector sums, as before. For the IS-divergence, i.e., $\beta = 0$, the cost function (4) can be rewritten as follows:

$$f = -N - \left(\sum_{n=1}^{N} \ln(t_n) \right) + \mathcal{T} \cdot_1 \frac{1}{\mathbf{a}} \cdot_2 \frac{1}{\mathbf{b}} \cdot_3 \frac{1}{\mathbf{c}} + \mathbf{1}^{\mathrm{T}} \begin{bmatrix} JK \ln(\mathbf{a}) \\ IK \ln(\mathbf{b}) \\ IJ \ln(\mathbf{c}) \end{bmatrix}.$$

The same ideas can be used as in the previous two cases.

B. Gradient

The gradient $[\mathbf{g_a}; \mathbf{g_b}; \mathbf{g_c}]$ can be simplified in a similar manner as the cost function. For general β, $\mathbf{g_a}$ from (5) can be written as

$$\begin{aligned} \mathbf{g_a} &= \sum_{j=1}^{J} \sum_{k=1}^{K} \frac{\mathbf{m}_{:jk} - \mathbf{t}_{:jk}}{\mathbf{m}_{:jk}^{\bullet(2-\beta)}} b_j c_k \\ &= s_{\mathbf{b}}^{\beta} s_{\mathbf{c}}^{\beta} \mathbf{a}^{\bullet(\beta-1)} - \mathbf{a}^{\bullet(\beta-2)} * \left[\mathcal{T} \cdot_2 \mathbf{b}^{\bullet(\beta-1)} \cdot_3 \mathbf{c}^{\bullet(\beta-1)} \right]. \quad (9) \end{aligned}$$

Analogously, we find $\mathbf{g_b}$ and $\mathbf{g_c}$. Note that we never need to explicitly form the $(I \times J \times K)$-tensor $\left(\frac{\mathcal{M}-\mathcal{T}}{\mathcal{M}^{\bullet(2-\beta)}}\right)$ and thus also avoid computing (non-integer) powers of all entries of $\mathcal{M}$. For $\beta \notin \{0,1\}$, all factors of the first term have already been obtained in (8), so only a scalar-vector multiplication is required. For the second term, both factors of the Hadamard product are new, so we need to compute an entry-wise vector power and a contraction. Both factors are reused in the computation of the Hessian approximation, however.

C. Hessian approximation

The full Hessian approximation $\mathbf{J}^{\mathrm{T}}\mathbf{Z}\mathbf{J}$ consists of 3×3 blocks $\mathbf{G}^{(p,q)}$, with $p,q \in \{1,2,3\}$. Using $\mathcal{D}$ from (6), the diagonal and off-diagonal blocks are of the form

$$\mathbf{G}^{(1,1)} = \mathrm{diag}\left(\mathcal{D} \cdot_2 \mathbf{b}^{\bullet 2} \cdot_3 \mathbf{c}^{\bullet 2} \right) \quad \text{and}$$
$$\mathbf{G}^{(1,2)} = \left[\mathbf{a}\mathbf{b}^{\mathrm{T}} \right] * \left(\mathcal{D} \cdot_3 \mathbf{c}^{\bullet 2} \right),$$

respectively [15]. Due to symmetry, only the blocks with $p \le q$ need to be computed. To compute these blocks efficiently, we exploit the rank-1 structure to avoid forming the tensor $\mathcal{D}$ and taking (non-integer) powers of all entries of a tensor. The diagonal blocks can be computed using the same strategies as for the gradient. As mentioned above, we precompute and reuse factors from the gradient when possible. Note that for rank-1 approximations, the diagonal blocks are diagonal matrices. Letting $\mathbf{G}_{\mathrm{diag}}^{(1,1)}$ be the diagonal of $\mathbf{G}^{(1,1)}$, we find:

$$\begin{aligned} \mathbf{G}_{\mathrm{diag}}^{(1,1)} &= \sum_{j=1}^{J} \sum_{k=1}^{K} \frac{(\beta-1)\mathbf{m}_{:jk} - (\beta-2)\mathbf{t}_{:jk}}{\mathbf{m}_{:jk}^{\bullet(3-\beta)}} b_j^2 c_k^2 \\ &= (\beta-1) s_{\mathbf{b}}^{\beta} s_{\mathbf{c}}^{\beta} \mathbf{a}^{\bullet(\beta-2)} \\ &\quad - (\beta-2)\, \mathbf{a}^{\bullet(\beta-3)} * \left[\mathcal{T} \cdot_2 \mathbf{b}^{\bullet(\beta-1)} \cdot_3 \mathbf{c}^{\bullet(\beta-1)} \right]. \end{aligned}$$

The matrices $\mathbf{G}^{(2,2)}$ and $\mathbf{G}^{(3,3)}$ can be found similarly. For the off-diagonal blocks, we have

$$\begin{aligned} \mathbf{G}^{(1,2)} &= \left[\mathbf{a}\mathbf{b}^{\mathrm{T}} \right] * \sum_{k=1}^{K} \frac{(\beta-1)\mathbf{M}_{::k} - (\beta-2)\mathbf{T}_{::k}}{\mathbf{M}_{::k}^{\bullet(3-\beta)}} c_k^2 \\ &= \left[\mathbf{a}^{\bullet(\beta-2)} \left(\mathbf{b}^{\bullet(\beta-2)} \right)^{\mathrm{T}} \right] * \left[(\beta-1) s_{\mathbf{c}}^{\beta} \mathbf{a}\mathbf{b}^{\mathrm{T}} - (\beta-2) \left(\mathcal{T} \cdot_3 \mathbf{c}^{\bullet(\beta-1)} \right) \right]. \end{aligned}$$

Analogously, one can find $\mathbf{G}^{(1,3)}$ and $\mathbf{G}^{(2,3)}$. Again, results from the gradient (9) are reused in the first term, leaving only a scaled vector outer product to compute. The second term still has to be computed. Note, however, that the factor $(\mathcal{T} \cdot_3 \mathbf{c}^{\bullet(\beta-1)}$ can be obtained as an intermediate result of computing $\mathcal{T} \cdot_2 \mathbf{b}^{\bullet(\beta-1)} \cdot_3 \mathbf{c}^{\bullet(\beta-1)}$ in the gradient.

For large-scale problems, one can use the (precondioned) conjugate gradient Steihaug [22] (for convex cost functions) or minres [23] (for nonconvex cost functions) method to approximately solve $\min_{\mathbf{p}} m(\mathbf{x}_k + \mathbf{p})$ with $\mathbf{x}_k + \mathbf{p} \ge 0$. Using this approach, we only need to compute products of the form

$[\mathbf{y_a};\mathbf{y_b};\mathbf{y_c}]=\mathbf{J}^{\mathrm{T}}\mathbf{Z}\mathbf{J}[\mathbf{x_a};\mathbf{x_b};\mathbf{x_c}]$, where $\mathbf{y_a}$ and $\mathbf{x_a}$ have the same size as $\mathbf{a}$, etc. Using the previously derived expressions, we can compute such products $\mathbf{y_a}=\mathbf{G}^{(1,1)}\mathbf{x_a}+\mathbf{G}^{(1,2)}\mathbf{x_b}+\mathbf{G}^{(1,3)}\mathbf{x_c}$, without having to form the matrix $\mathbf{J}^{\mathrm{T}}\mathbf{Z}\mathbf{J}$, in the following way:

$$\begin{aligned}\mathbf{G}^{(1,1)}\mathbf{x_a} &= \left[\mathbf{x_a} * \mathbf{a}^{\bullet(\beta-3)}\right] * \Big[(\beta-1)s_{\mathbf{b}}^{\beta}s_{\mathbf{c}}^{\beta}\mathbf{a} \\ &\qquad -(\beta-2)\,\mathcal{T}\cdot_2\mathbf{b}^{\bullet(\beta-1)}\cdot_3\mathbf{c}^{\bullet(\beta-1)}\Big],\\ \mathbf{G}^{(1,2)}\mathbf{x_b} &= (\beta-1)s_{\mathbf{c}}^{\beta}\mathbf{a}^{\bullet(\beta-1)}\left[\mathbf{x_b}^{\mathrm{T}}\mathbf{b}^{\bullet(\beta-1)}\right] \\ &\qquad -(\beta-2)\,\mathbf{a}^{\bullet(\beta-2)} * \left[\mathcal{T}\cdot_2\mathbf{b}_{\mathbf{x_b}}\cdot_3\mathbf{c}^{\bullet(\beta-1)}\right],\\ \mathbf{G}^{(1,3)}\mathbf{x_c} &= (\beta-1)s_{\mathbf{b}}^{\beta}\mathbf{a}^{\bullet(\beta-1)}\left[\mathbf{x_c}^{\mathrm{T}}\mathbf{c}^{\bullet(\beta-1)}\right] \\ &\qquad -(\beta-2)\,\mathbf{a}^{\bullet(\beta-2)} * \left[\mathcal{T}\cdot_2\mathbf{b}^{\bullet(\beta-1)}\cdot_3\mathbf{c}_{\mathbf{x_c}}\right],\end{aligned}$$

where $\mathbf{b}_{\mathbf{x_b}}=\mathbf{x_b}*\mathbf{b}^{\bullet(\beta-2)}$ and $\mathbf{c}_{\mathbf{x_c}}=\mathbf{x_c}*\mathbf{c}^{\bullet(\beta-2)}$. Note that the first terms of $\mathbf{G}^{(1,2)}$ and $\mathbf{G}^{(1,3)}$ both have the factor $(\beta-1)\mathbf{a}^{\bullet(\beta-1)}$, so their sum can be computed as

$$(\beta-1)\mathbf{a}^{\bullet(\beta-1)}\left(s_{\mathbf{c}}^{\beta}\mathbf{x_b}^{\mathrm{T}}\mathbf{b}^{\bullet(\beta-1)}+s_{\mathbf{b}}^{\beta}\mathbf{x_c}^{\mathrm{T}}\mathbf{c}^{\bullet(\beta-1)}\right).$$

The sum of their second terms can also be rewritten as

$$(2-\beta)\,\mathbf{a}^{\bullet(\beta-2)} * \left[\mathcal{T}\cdot_2\mathbf{b}^{\bullet(\beta-1)}\cdot_3\mathbf{c}_{\mathbf{x_c}}+\mathcal{T}\cdot_2\mathbf{b}_{\mathbf{x_b}}\cdot_3\mathbf{c}^{\bullet(\beta-1)}\right].$$

Computing the product $\mathbf{J}^{\mathrm{T}}\mathbf{Z}\mathbf{J}[\mathbf{x_a};\mathbf{x_b};\mathbf{x_c}]$ only requires storing vectors of length I, except for the intermediate steps in the computation of the tensor contractions. Note that even this can be avoided by computing the tensor contractions slice by slice, i.e., computing only one entry at the time. When $\mathcal{T}$ is sparse or structured (e.g., Hankel, Löwner, low multilinear rank), this structure can also be exploited during the computations [24], leading to lower computation times and storage requirements. Similar expressions can be derived for $\mathbf{y_b}$ and $\mathbf{y_c}$. As $\mathbf{J}^{\mathrm{T}}\mathbf{Z}\mathbf{J}$ is diagonally dominant and its diagonal blocks are diagonal matrices, a Jacobi preconditioner can be expected to perform well. The preconditioner $\mathbf{M}^{-1}$ can be obtained cheaply by computing $\mathbf{G}^{(1,1)}_{\text{diag}}$, $\mathbf{G}^{(2,2)}_{\text{diag}}$ and $\mathbf{G}^{(3,3)}_{\text{diag}}$:

$$\mathbf{M}^{-1}=\operatorname{diag}\left(\left[\frac{1}{\mathbf{G}^{(1,1)}_{\text{diag}}};\frac{1}{\mathbf{G}^{(2,2)}_{\text{diag}}};\frac{1}{\mathbf{G}^{(3,3)}_{\text{diag}}}\right]\right).$$

Compared to the expressions for general R, those for the rank-1 case offer a significant efficiency gain. On the one hand, more of the multilinear structure can be exploited. On the other hand, more intermediate results from the objective function and gradient can be reused in the gradient and Hessian approximation, respectively.

IV. Experiments

Two experiments are performed on synthetic data, using a computer with an Intel Core i7-6820HQ CPU at 2.70GHz and 16GB of RAM, MATLAB R2016b and Tensorlab 3.0 [25].

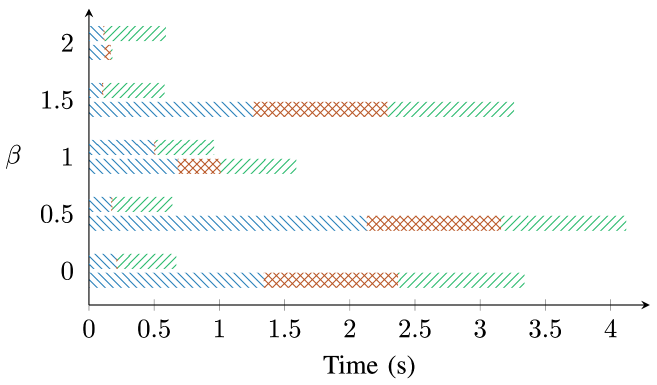

Figure 1. Time spent computing cost function (▨), gradient (▨) and Hessian approximation (▨) during 10 iterations of the NLS method with (top bars) and without (bottom bars) rank-1 exploitation. The rank-1 method offers a large speedup, except for the case $\beta=2$ (LS cost function).

Experiment 1: We analyze the decrease in computation time that the exploitation of the rank-1 structure brings by computing rank-1 decompositions of 10 random rank-1 tensors $\mathcal{T}\in\mathbb{R}_{+}^{10\times10\times10\times10\times10\times10}$. The factors are sampled uniformly from $[0,1]$. We initialize with random factors with entries sampled uniformly from $[0,1]$ and perform 10 iterations of the general rank-R and the specialized rank-1 algorithms. In Figure 1, we see the time needed for the computation of the cost function, gradient and Hessian approximation for different values of β, averaged over the 10 tensors. One can see that the rank-1 algorithm requires about 2 to 6 times less computation time compared to the general β-divergence algorithm. The main improvements lie in the computation of the gradient, due to the fact that almost all factors can be reused from the objective function. Except for $\beta=1$ the computation of the cost function also becomes much more efficient, partly because the full tensor $\mathcal{M}$ never has to be formed explicitly. Of course, in this noiseless case, the optimal factor matrices are the same for any value of β. Thus in practice, one should use the least-squares algorithm instead, as it is still faster than the rank-1 β-divergence method.

Experiment 2: We compare the number of successful rank-1 approximations for three different methods: the proposed second-order rank-1 NLS method, the first-order generalized canonical polyadic decomposition (GCPD) method [11], and a first-order method using multiplicative updates (MU) [12]. A relative step size of $\epsilon=2.22\cdot10^{-16}$ is used as a stopping criterion for the NLS and MU methods and a relative cost function improvement of ϵ is used as as stopping criterion for the GCPD method. The methods were additionally limited to 100 iterations, but this upper bound is only reached by the GCPD method for $\beta<1$. We perturb 20 rank-1 tensors $\mathcal{T}\in\mathbb{R}_{+}^{50\times50\times50}$ with factor matrices sampled uniformly from $[0,1]$ by multiplicative Gamma noise with an SNR of 40dB. These tensors are approximated by a rank-1 tensor. We call an approximation successful if the relative error on the obtained factor matrices lies below 10^{-3}. In Table I, we can see that the rank-1 NLS method and the MU algorithm find the solution in all cases, while the GCPD

Table I
FRACTION OF SUCCESSFUL APPROXIMATIONS, COMPUTATION TIME AND MEMORY REQUIREMENTS FOR THREE RANK-1 β-DIVERGENCE APPROXIMATION METHODS. THE NLS ALGORITHM IS SLIGHTLY FASTER THAN THE MU ALGORITHM AND REQUIRES LESS STORAGE SPACE. THE GCPD METHOD GENERALLY PERFORMS BADLY FOR $\beta < 1$.

	Method $\setminus$ β	-0.5	0	0.5	1	1.5	2.5
Success rate	Rank-1 NLS	1	1	1	1	1	1
	MU	1	1	1	1	1	1
	GCPD	0	0.05	0.65	1	1	1
Time (s)	Rank-1 NLS	1.62	1.25	0.47	0.37	0.33	0.25
	MU	5.26	4.85	1.66	0.11	0.36	0.49
	GCPD	7.53	1.73	3.90	0.73	1.04	1.15
Memory	Rank-1 NLS	$\mathcal{O}(I)$ + tensor contractions					
	MU	$\mathcal{O}(I^3)$					
	GCPD	$\mathcal{O}(I^3)$					

method fails for values of $\beta < 1$, where convexity of d_β is not guaranteed. The NLS algorithm is slightly faster than the MU algorithm. Further, for an $(I \times I \times I)$-tensor $\mathcal{T}$, the MU and GCPD algorithms require the formation of a full (dense) $(I \times I \times I)$-approximation tensor in every iteration, even when $\mathcal{T}$ is sparse. In contrast, the rank-1 NLS method only needs to store some vectors of length I and compute contractions with the original tensor $\mathcal{T}$.

V. CONCLUSION

As rank-1 approximations are often used in practice, we propose a dedicated algorithm to compute a rank-1 β-divergence approximation to a tensor. The algorithm uses a good approximation of the Hessian and thus offers fast local convergence, while it avoids the construction of large intermediate tensors and can exploit available structure in the tensor. In future work, we will look at other classes of cost functions and possible extensions to higher ranks.

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] E. Acar, D. M. Dunlavy, and T. G. Kolda, "A scalable optimization approach for fitting canonical tensor decompositions," *Journal of Chemometrics*, vol. 25, no. 2, pp. 67–86, 2011.

[7] N. Vervliet and L. De Lathauwer, "Numerical optimization based algorithms for data fusion," in *Data Fusion Methodology and Applications*, M. Cocchi, Ed. Elsevier, 2019, vol. 31, pp. 1–41.

[8] R. Hennequin, B. David, and R. Badeau, "Beta-divergence as a subclass of Bregman divergence," *IEEE Signal Processing Letters*, vol. 18, no. 2, pp. 83–86, 2011.

[9] C. Févotte and J. Idier, "Algorithms for nonnegative matrix factorization with the β-divergence," *Neural Computation*, vol. 23, no. 9, pp. 2421–2456, 2011.

[10] C. Févotte, N. Bertin, and J.-L. Durrieu, "Nonnegative matrix factorization with the Itakura–Saito divergence: With application to music analysis," *Neural computation*, vol. 21, no. 3, pp. 793–830, 2009.

[11] D. Hong, T. G. Kolda, and J. A. Duersch, "Generalized canonical polyadic tensor decomposition," *arXiv preprint arXiv:1808.07452*, 2018.

[12] A. Cichocki and A.-H. Phan, "Fast local algorithms for large scale nonnegative matrix and tensor factorizations," *IEICE transactions on fundamentals of electronics, communications and computer sciences*, vol. 92, no. 3, pp. 708–721, 2009.

[13] E. C. Chi and T. G. Kolda, "On tensors, sparsity, and nonnegative factorizations," *SIAM Journal on Matrix Analysis and Applications*, vol. 33, no. 4, pp. 1272–1299, 2012.

[14] S. Hansen, T. Plantenga, and T. G. Kolda, "Newton-based optimization for Kullback–Leibler nonnegative tensor factorizations," *Optimization Methods and Software*, vol. 30, no. 5, pp. 1002–1029, 2015.

[15] M. Vandecappelle, N. Vervliet, and L. De Lathauwer, "Canonical polyadic decomposition with general cost functions using generalized Gauss–Newton," 2019, Internal Report 19-05, ESAT-STADIUS, KU Leuven (Leuven, Belgium).

[16] N. Schraudolph, "Fast curvature matrix-vector products for second-order gradient descent," *Neural computation*, vol. 14, pp. 1723–38, 08 2002.

[17] X. Shi, H. Ling, J. Xing, and W. Hu, "Multi-target tracking by rank-1 tensor approximation," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR 2013)*, 2013, pp. 2387–2394.

[18] Y. Yang, Y. Feng, X. Huang, and J. Suykens, "Rank-1 tensor properties with applications to a class of tensor optimization problems," *SIAM Journal on Optimization*, vol. 26, no. 1, pp. 171–196, 2016.

[19] M. Kim and P. Smaragdis, "Efficient model selection for speech enhancement using a deflation method for nonnegative matrix factorization," in *2014 IEEE Global Conference on Signal and Information Processing (GlobalSIP 2014)*. IEEE, 2014, pp. 537–541.

[20] N. Gillis and F. Glineur, "Using underapproximations for sparse nonnegative matrix factorization," *Pattern recognition*, vol. 43, no. 4, pp. 1676–1687, 2010.

[21] A. Shashua and T. Hazan, "Non-negative tensor factorization with applications to statistics and computer vision," in *Proceedings of the 22nd International Conference on Machine Learning (ICML 2005)*, August 2005, pp. 792–799.

[22] T. Steihaug, "The conjugate gradient method and trust regions in large scale optimization," *SIAM Journal on Numerical Analysis*, vol. 20, no. 3, pp. 626–637, 1983.

[23] C. Paige and M. Saunders, "Solution of sparse indefinite systems of linear equations," *SIAM Journal on Numerical Analysis*, vol. 12, no. 4, pp. 617–629, 1975.

[24] N. Vervliet, O. Debals, and L. De Lathauwer, "Exploiting efficient representations in large-scale tensor decompositions," 2016, Internal Report 16-174, ESAT-STADIUS, KU Leuven (Leuven, Belgium), Accepted for publication in SIAM Journal on Scientific Computing.

[25] 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.