Thesis Proposal: Scalable and Practical Tensor Decompositions using Randomization

Casey Battaglino[1]
Advisor: Dr. Richard Vuduc

August 16, 2018

[1]Georgia Institute of Technology Computational Science and Engineering.

Contents

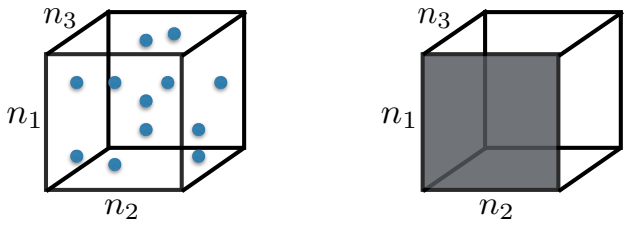


Figure 1: A visualization of an order 3 ($d = 3$) sparse and dense tensor (left and right, respectively.

1 Introduction

An emerging area of research in computational science considers efficiently computing on data sets that are inherently multi-way– that is, they can be represented by higher-order *tensors* [2, 36]. Tensor *decompositions* are a powerful tool for the analysis of multi-way data, with many applications including including chemometrics [32], quantum chemistry [35], neuroscience [14], healthcare [63], and deep learning [42]. Tensor decompositions generally attempt to express an input tensor in a form that is lower-dimensional or of a more desirable structure. The resulting decomposition may be valuable for its interpretability (e.g., for factor analysis), or as a compressed format that alleviates the curse of dimensionality. Two of the most popular decompositions are visualized in fig. 2, and the storage costs of the decompositions mentioned in this proposal are shown in table 1, where the original tensor has dimension n_k in mode k and the decomposition has rank r_k in mode k. Typically, $r_k << n_k$.

Representation	**Storage Cost**
Original Tensor	$\prod_k n_k = \mathcal{O}(n^d)$
CP	$\sum_k r_k n_k = \mathcal{O}(nr)$
Tucker	$\sum_k r_k n_k + \prod_k r_k = \mathcal{O}(r^d)$
Tensor Train	$n_1 r_1 + n_d r_d + \sum_{k=2}^{d-1} r_{k-1} r_k n_k = \mathcal{O}(nr^2)$

Table 1: Storage costs of tensor decompositions in this proposal.

As we can see from this table, a full representation of tensor data can scale arbitrarily large (exponential in its order, if each dimension is of a similar size). It is thus desirable to develop decomposition algorithms that scale in kind. We propose to utilize *randomized* methods towards this end. Recent developments in randomized numerical linear algebra utilize techniques such as random projection and sampling to perform common operations such as low-rank decomposition and least squares much faster than traditional methods, with the drawback that error bounds must often be restated in probabilistic terms. A central motivation in this proposal is the belief that the dimensionality of tensors lends itself *particularly* well to these randomized methods.

Towards this goal we propose to develop efficient, high-performance implementations of randomized algorithms for leading tensor decompositions. We demonstrate in our prior work that the CANDECOMP/PARAFAC (CP) decomposition can be performed with randomized least squares, and that the

tensor structure actually produces favorable conditions for the algorithm– in fact, the randomized algorithm can extract more robust features than the leading deterministic algorithm at much lower cost (section 4).

In addition we target scaling up the Tucker decomposition in distributed memory using methods drawn from the randomized SVD (section 5), and propose that the tensor structure also suits both this method and another computation, the Tensor Train decomposition (section 6).

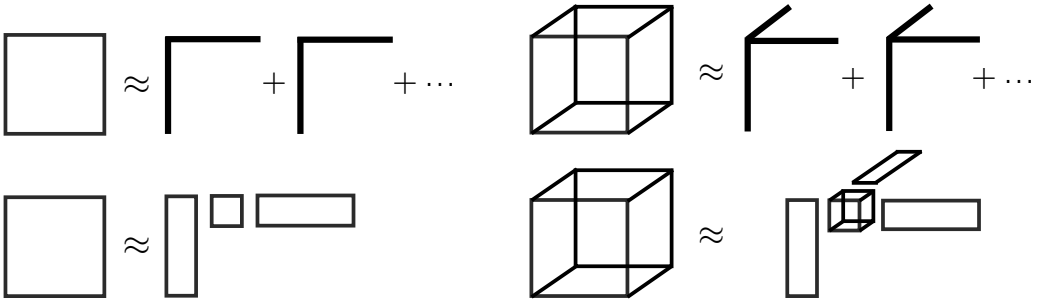


Figure 2: Left: A low-rank matrix decomposition, visualized as a sum of rank-one matrices (top) or as a product of smaller matrices (bottom). Right: The CP decomposition, which is a sum of rank-one tensors (top), and the Tucker decomposition, which is a product of a smaller tensor with d factor matrices in each mode (bottom).

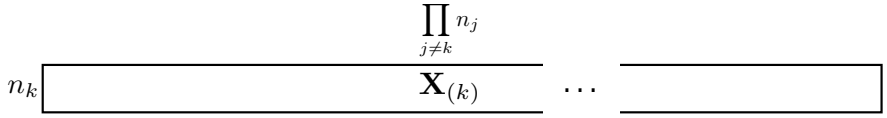


Figure 3: Intermediate computations in tensor decompositions often involve matrices where one dimension is much larger than the other. This is one example of a feature that is easily exploitable by randomized methods.

1.1 Proposal

We propose a line of research to apply sketching techniques to tensor decompositions, and to leverage them in such a way that allows for high-performance algorithms that outperform the state of the art. A key aspect

within each decomposition is to show how sketching algorithms can leverage the specific higher-order structure of tensors.

1. Candecomp/Parafac (CP) Decomposition

    - The leading method for computing the CP decomposition is CP-ALS, which performs iterative least-squares updates in each mode.
    - In completed work [8], we use methods drawn from *randomized least squares* and apply them to this core computation.
    - We show that the intermediate tensor structure produces conditions favorable for sketching due to both the dimensionality and 'mixing' incurred by the Khatri Rao product.
    - We introduce a new termination condition for the CP decomposition that uses sampled entries from the original tensor, and probabilistically bound the quality of this condition using the Chernoff-Hoeffding inequality.
    - We demonstrate scalability on real and synthetic data sets.
    - We demonstrate the surprising result that this method often works *better* than CP-ALS, returning good solutions more robustly.
    - Deliverable: this method is published [8] and released as part of MATLAB Tensor Toolbox [7][1].

2. Tucker Decomposition / HOSVD

    - The leading method for computing the Tucker decomposition is the HOSVD, which performs mode-wise SVDs followed by tensor contractions.
    - We propose to apply ideas from the Randomized SVD to the HOSVD in distributed memory.
    - We propose that the tensor structure makes random projection highly efficient because it can be expressed as a series of small tensor contractions without intermediate communication.
    - We propose that the tensor structure uniquely allows for high-quality output because the efficiency of projections allows for significant *oversampling* in very little time.

[1] `http://gitlab.com/tensors/tensor_toolbox`

- We propose to demonstrate scalability on massive real and synthetic data sets on up to 1000 nodes.
- Deliverable: this method will be submitted for publication and included as part of TuckerMPI[2].

3. Tensor Train Decomposition

- The tensor train decomposition is a leading low-rank representation for very high-order tensors.
- We propose exploring how the structure of the tensor train computation can be exploited by randomized methods in a way comparable to the previous two methods (CP/Tucker).
- We propose demonstration of scalability on massive real and synthetic data sets.
- We propose developing a practical framework for the tensor train and quantized tensor train that allows for randomization.

2 Background and Definitions

In this section, we provide information on the necessary tensor properties and operations, and then introduce tensor decompositions and associated randomized methods.

2.1 Tensors

A tensor is an element in a tensor product of vector spaces. In data analysis, it suffices to think about a tensor as a multidimensional array. We represent a tensor as a Euler script capital letter, e.g., $\mathcal{X} \in \mathbb{R}^{n_1 \times \cdots \times n_d}$.

The number of *modes* (or dimensions) of a tensor is referred to as its *order*, denoted by d. The values n_k denote the dimensions of a tensor, and we let $n = \sqrt[d]{\prod_k n_k}$, so that $n^d = \prod_k n_k$. Thus, the term n^d to more intuitively expresses the number of elements in a tensor as exponential in its order. We also let $n_k^{\oslash} = n^d/n_k$ represent the product of all dimensions *except* n_k.

Let $\mathcal{I} = \{\mathbf{i} = (i_1, \ldots, i_d)\}$ be the set of indices of a tensor. We can thus express an individual element of a tensor $\mathcal{X}$ for any multiindex $\mathbf{i}$ as $x_{\mathbf{i}}$.

[2] `http://tensors.gitlab.io/TuckerMPI/`

The *mode-k fibers* of a tensor are higher-order analogues of matrix columns and rows. The mode-k *unfolding* or *matricization* of a tensor aligns the mode-k fibers as the columns of an $n_k \times n_k^{\oslash}$ matrix. Assuming 1-indexing, tensor entry $x_{\mathbf{i}}$ then maps to entry (i_k, j) of $\mathbf{X}_{(k)}$ via the relation:

$$j = 1 + \sum_{\substack{\ell=1 \\ \ell \neq k}}^{d} (i_\ell - 1) m_\ell, \qquad \text{where} \qquad m_\ell = \prod_{\substack{q=1 \\ q \neq \ell}}^{\ell-1} n_q. \tag{1}$$

The *norm* of a tensor is the square root of the sum of its squared entries, e.g., $\|\mathcal{X}\| = \|\mathbf{X}_{(k)}\|_F$ for any k. Given a decomposed representation $\mathcal{M}$ of a tensor, the normalized residual error can be written as:

$$\|\mathcal{X} - \mathcal{M}\| / \|\mathcal{X}\| \tag{2}$$

Given matrices $\mathbf{A} \in \mathbb{R}^{n_1 \times n_2}$ and $\mathbf{B} \in \mathbb{R}^{m_1 \times m_2}$, their *Kronecker product* is

$$\mathbf{A} \otimes \mathbf{B} = \begin{bmatrix} a_{11}\mathbf{B} & a_{12}\mathbf{B} & \cdots & a_{1n_2}\mathbf{B} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n_1 1}\mathbf{B} & a_{n_1 2}\mathbf{B} & \cdots & a_{n_1 n_2}\mathbf{B} \end{bmatrix} \in \mathbb{R}^{n_1 m_1 \times n_2 m_2}.$$

Assuming $n_2 = m_2$, their *Khatri-Rao product*, also known as the *matching columnwise Kronecker product*, is

$$\mathbf{A} \odot \mathbf{B} = \begin{bmatrix} \mathbf{a}_1 \otimes \mathbf{b}_1 & \mathbf{a}_2 \otimes \mathbf{b}_2 & \cdots & \mathbf{a}_J \otimes \mathbf{b}_J \in \mathbb{R}^{n_1 m_1 \times n_2}. \end{bmatrix}$$

Assuming $n_1 = m_1$ and $n_2 = m_2$, their *Hadamard product* is $\mathbf{A} \circledast \mathbf{B} \in \mathbb{R}^{n_1 \times n_2}$, the elementwise product of the matrices. Three useful identities involving the products just defined are:

$$(\mathbf{A} \odot \mathbf{B})^{\mathsf{T}} (\mathbf{A} \odot \mathbf{B}) = \mathbf{A}^{\mathsf{T}}\mathbf{A} \circledast \mathbf{B}^{\mathsf{T}}\mathbf{B}, \tag{3}$$

$$\mathbf{AB} \otimes \mathbf{CD} = (\mathbf{A} \otimes \mathbf{C})(\mathbf{B} \otimes \mathbf{D}), \quad \text{and} \tag{4}$$

$$\mathbf{AB} \odot \mathbf{CD} = (\mathbf{A} \otimes \mathbf{C})(\mathbf{B} \odot \mathbf{D}). \tag{5}$$

The *mode-k tensor-times-matrix product* (TTM) is a contraction between a matrix and a tensor in its kth mode.

$$\mathcal{Y} = \mathcal{X} \times_k \mathbf{A} \quad \Leftrightarrow \quad \mathbf{Y}_{(k)} = \mathbf{A}\mathbf{X}_{(k)}. \tag{6}$$

We can also write this element-wise as

$$y_{i_1 i_2 \cdots i_{k-1} j i_{k+1} \cdots i_d} = \sum_{i_k} x_{i_1 \cdots i_d} u_{j i_k} \tag{7}$$

We will use bracket notation to denote multiple products, e.g. $\mathcal{X} \times \{\mathbf{U}_k\}$ refers to $\mathcal{X}$ multiplied by $\mathbf{U}_k$ for every $k = 1, \ldots, d$. The result is invariant to which order the TTMs are performed in, providing the modes are unique. If a tensor can be written as a series of mode-k products, its mode-k matricization has a particular structure [36]:

$$\begin{aligned} \mathcal{Y} = \mathcal{X} \times \{\mathbf{U}_k\} \quad &\Leftrightarrow \\ \mathbf{Y}_{(k)} = \mathbf{U}_k \mathbf{X}_{(k)} (\mathbf{U}_d \otimes \cdots \otimes \mathbf{U}_{k+1} \otimes \mathbf{U}_{k-1} \otimes \cdots \otimes \mathbf{U}_1)^\mathsf{T}. \end{aligned} \tag{8}$$

Notation	Definition
$\mathbf{X}_{(k)}$	mode-k unfolding of $\mathcal{X}$
$\mathcal{X} \times_k \mathbf{U}_k$	Tensor-Times Matrix Multiplication (TTM)
$\mathbf{A} \otimes \mathbf{B}$	Kronecker Product
$\mathbf{A} \odot \mathbf{B}$	Khatri-Rao Product
$\mathbf{A} \circledast \mathbf{B}$	Hadamard Product
d	Number of modes (order) of a tensor
n_k	Size of mode k of tensor $\mathcal{X}$
r_k	Rank (core size) of mode k[3]
p_k	Number of processors along mode k
s_k	Sketch size of mode k
n, r, p, s	$\sqrt[d]{\prod n_k}$, $\sqrt[d]{\prod r_k}$, $\sqrt[p]{\prod p_k}$, $\sqrt[d]{\prod s_k}$
$n_k^{\oslash}, r_k^{\oslash}, p_k^{\oslash}, s_k^{\oslash}$	$n^d/n_k, r^d/r_k, p^d/p_k, s^d/s_k$

Table 2: Basic tensor notation used in this proposal.

2.2 CP Decomposition

The CP tensor decomposition aims to approximate an order-d tensor as a sum of r rank-one tensors [27, 11, 25, 36]:

$$\mathcal{X} \approx \tilde{\mathcal{X}} = \sum_{k=1}^{d} \mathbf{a}_k^{(1)} \circ \mathbf{a}_k^{(2)} \circ \cdots \circ \mathbf{a}_k^{(d)}, \tag{9}$$

where *factor vector* $\mathbf{a}_r^{(k)}$ has length n_k. Each rank-one tensor is called a *component*. The collection of all factor vectors for a given mode is called a *factor matrix*:

$$\mathbf{A}_k = \begin{bmatrix} \mathbf{a}_1^{(k)} & \mathbf{a}_2^{(k)} & \cdots & \mathbf{a}_r^{(k)} \end{bmatrix} \in \mathbb{R}^{n_k \times r}.$$

The mode-k matricization of $\tilde{\boldsymbol{\mathcal{X}}}$ can be written in terms the factor matrices as

$$\tilde{\mathbf{X}}_{(k)} = \mathbf{A}^{(k)} \mathbf{Z}^{(k)\mathsf{T}} \quad \text{where} \quad \mathbf{Z}^{(k)} = \mathbf{A}^{(d)} \odot \cdots \mathbf{A}^{(k+1)} \odot \mathbf{A}^{(k-1)} \odot \cdots \odot \mathbf{A}^{(1)}. \tag{10}$$

We may alternatively represent eq. (9) by normalizing all the factor vectors to unit length and expressing the product of the normalization factors as a scalar weight λ_r for each component:

$$\tilde{\boldsymbol{\mathcal{X}}} = \sum_{k=1}^{d} \lambda_k \, \mathbf{a}_k^{(1)} \circ \mathbf{a}_k^{(2)} \circ \cdots \circ \mathbf{a}_k^{(d)}. \tag{11}$$

CP-ALS

The standard method for fitting the CP model is alternating least squares (CP-ALS) [25, 36]. The method alternates among the modes, fixing every factor matrix but $\mathbf{A}_k$ and solving for it. From eq. (10), we see that we can find $\mathbf{A}_k$ by solving the linear least squares problem given by

$$\arg\min_{\mathbf{A}^{(k)}} \|\mathbf{X}_{(k)} - \mathbf{A}^{(k)} \mathbf{Z}^{(k)\mathsf{T}}\|_F. \tag{12}$$

In CP-ALS, we work with the normal equations for eq. (12):

$$\mathbf{X}_{(k)} \mathbf{Z}^{(k)} = \mathbf{A}^{(k)} (\mathbf{Z}^{(k)\mathsf{T}} \mathbf{Z}^{(k)}),$$

and solve for $\mathbf{A}^{(k)}$ for given $\mathbf{X}_{(k)}$ and $\mathbf{Z}^{(k)}$. By identity eq. (3), we have

$$\mathbf{Z}^{(k)\mathsf{T}} \mathbf{Z}^{(k)} = \mathbf{A}^{(d)\mathsf{T}} \mathbf{A}^{(d)} \circledast \cdots \circledast \mathbf{A}^{(k+1)\mathsf{T}} \mathbf{A}^{(k+1)} \circledast \mathbf{A}^{(k-1)\mathsf{T}} \mathbf{A}^{(k-1)} \circledast \cdots \circledast \mathbf{A}^{(1)\mathsf{T}} \mathbf{A}^{(1)}. \tag{13}$$

The CP-ALS algorithm [36] is presented in algorithm 1. Note the step where vector $\boldsymbol{\lambda}$ stores normalization values of each column so that the final approximation is as in eq. (11); this normalization helps alleviate issues due to scaling ambiguity.

The initialization of the factor matrices can impact the performance of the algorithm. There are many possible ways to do the the initialization. One way is to initialize is to set $\mathbf{A}_k$ to be the leading r left singular vectors of the mode-k unfolding, $\mathbf{X}_{(k)}$, and we call this HOSVD initialization, as it corresponds to the factor matrices in the rank-$(r \times \cdots \times r)$ HOSVD (see section 2.3). A less expensive but less effective initialization is to choose random factor matrices.

Finally, a termination criterion must be supplied. For instance, a simple way to check convergence is to see when the change in residual error (eq. (2)) becomes negligible.

Algorithm 1 CP-ALS

1: **function** $[\boldsymbol{\lambda}, \{\mathbf{A}^{(k)}\}] = \text{CP-ALS}(\mathcal{X}, r)$ ▷ $\mathcal{X} \in \mathbb{R}^{n_1 \times \cdots \times n_d}$
2: Initialize factor matrices $\mathbf{A}^{(2)}, \ldots, \mathbf{A}^{(d)}$
3: **repeat**
4: **for** $k = 1, \ldots, d$ **do**
5: $\mathbf{V} \leftarrow \mathbf{A}^{(d)\mathsf{T}}\mathbf{A}^{(d)} \circledast \cdots \circledast \mathbf{A}^{(k+1)\mathsf{T}}\mathbf{A}^{(k+1)} \circledast \mathbf{A}^{(k-1)\mathsf{T}}\mathbf{A}^{(k-1)} \circledast \cdots \circledast \mathbf{A}^{(1)\mathsf{T}}\mathbf{A}^{(1)}$
6: $\mathbf{Z}^{(k)} \leftarrow \mathbf{A}^{(d)} \odot \cdots \odot \mathbf{A}^{(k+1)} \odot \mathbf{A}^{(k-1)} \odot \cdots \odot \mathbf{A}^{(1)}$
7: $\mathbf{W} \leftarrow \mathbf{X}_{(k)}\mathbf{Z}^{(k)}$
8: Solve $\mathbf{A}^{(k)}\mathbf{V} = \mathbf{W}$ for $\mathbf{A}^{(k)}$
9: Normalize columns of $\mathbf{A}^{(k)}$ and update $\boldsymbol{\lambda}$
10: **end for**
11: **until** termination criteria met
12: **return** $\boldsymbol{\lambda}$, factor matrices $\{\mathbf{A}^{(k)}\}$
13: **end function**

Because of the elegant least squares structure in eq. (13), the bottleneck for CP-ALS is generally in performing lines 6-7, which we refer to as the matricized Khatri-Rao product (MTTKRP). High-performance implementations of CP-ALS generally target the MTTKRP [54, 33, 26].

2.3 The Tucker Decomposition

The Tucker decomposition [59] approximates a tensor with a core tensor contracted with matrices in each mode:

$$\mathcal{X} \approx \mathcal{M} = \mathcal{G} \times_1 \mathbf{U}_1 \times_2 \mathbf{U}_2 \cdots \times_d \mathbf{U}_d = \mathcal{G} \times \{\mathbf{U}_k\},$$

where $\mathcal{G}$ is a dense core of size $r_1 \times r_2 \times \cdots \times r_d$, and the factor matrices $\mathbf{U}_k$ have size $n_k \times r_k$ for $k = 1, \ldots, d$.

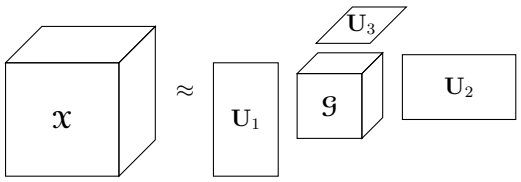


Figure 4: Tucker decomposition of 3rd-order tensor ($d = 3$).

The HOSVD [17] is a method for computing the Tucker decomposition that computes a series of SVDs of unfolded tensors to compute the orthogonal factor matrices $\mathbf{U}_k$ whose columns approximately span the columns of the unfolded tensor $\mathbf{X}_{(k)}$ for each mode k. We will refer to this computation as the Mode-wise Truncated Fiber Space Basis Computation (MTFSBC). Following this operation, the resulting factor matrices are applied to the input tensor to compress the original data into the core $\mathcal{G}$: $\mathcal{G} \leftarrow \mathcal{X} \times \{\mathbf{U}_k^\mathsf{T}\}$.

Just as we would compress a matrix by truncating its SVD, we can truncate the factors $\mathbf{U}_k$ so that $\mathcal{G}$ is a smaller core ($r_k < n_k$), a method we refer to as the *truncated* HOSVD (T-HOSVD), presented in algorithm 2. This is particularly effective for compression; as with n, we let $r = \sqrt[d]{\prod r_k}$, and so the core is exponentially smaller than $\mathcal{X}$, a factor of $(\frac{n}{r})^d$. The total compression ratio includes the factor matrices:

$$n^d / \left(r^d + \sum_{k=1}^{d} n_k r_k \right). \tag{14}$$

The HOSVD can be implemented by forming the $n_k \times n_k$ Gram matrix $\mathbf{S}_k = \mathbf{X}_{(k)}\mathbf{X}_{(k)}^\mathsf{T}$ and computing its eigendecomposition. The eigenvectors of $\mathbf{S}_k$ correspond to the left singular vectors of $\mathbf{X}_{(k)}$, and $\lambda_j(\mathbf{S}_k) = \sigma_j(\mathbf{X}_{(k)})^2$. This Gram computation is utilized by TuckerMPI, which performs a distributed Gram matrix computation followed by a local eigendecomposition [5]. We use the Gram variant because we assume that dimensions are reasonably-sized, e.g., $n_k \leq 10^4$ for all k.

A new variant of the T-HOSVD is the *sequentially* truncated HOSVD (ST-HOSVD) [60], which is implemented in TuckerMPI [5]. Whereas the T-HOSVD forms the core after performing the MTFSBC for all modes (in line 8), the ST-HOSVD performs a MTFSBC for a single mode followed by a core-compression step in that mode, such that the kth TTM occurs within the kth iteration of the for loop. The working size of the tensor thus reduces by

Algorithm 2 T-HOSVD ▷ Gram Variant

procedure T-HOSVD($\mathcal{X}$, $(r_1, \ldots, r_d)$)
  for $k = 1, \ldots, d$ **do**
    $\mathbf{S}_k \leftarrow \mathbf{X}_{(k)}\mathbf{X}_{(k)}^{\mathsf{T}}$ ▷ Gram
    $[\mathbf{U}, \boldsymbol{\lambda}] \leftarrow \text{eig}(\mathbf{S}_k)$ ▷ Eigensolve
    $\mathbf{U}_k \leftarrow$ leading r_k eigenvectors in $\mathbf{U}$
  end for
  $\mathcal{G} \leftarrow \{\mathcal{X} \times \mathbf{U}_k^{\mathsf{T}}\}$ ▷ TTM (Core Formation)
  $\mathcal{M} \leftarrow \{\mathcal{G}, \{\mathbf{U}_k\}\}$
  return $\mathcal{M}$
end procedure

a factor of n_k/r_k at each iteration, with an equivalent reduction in the cost of subsequent SVD steps.

2.4 Tensor Train Decomposition

The Tensor Train decomposition (TT) factors an order-d tensor into a set of d tensors. The kth tensor is denoted $\mathcal{G}^{(k)}$, where $\mathbf{G}^{(1)}$ and $\mathbf{G}^{(d)}$ are of order 2 (matrices), and all other tensors are of order 3. This representation has its roots in the MPS formulation in quantum physics [45].

The input tensor can then be reconstructed via the relation:

$$\mathcal{X} \approx \mathbf{G}^{(1)} \times_2^1 \mathcal{G}^{(2)} \times_3^1 \mathcal{G}^{(3)} \times_3^1 \cdots \times_3^1 \mathcal{G}^{(d-1)} \times_3^1 \mathbf{G}^{(d)} \tag{15}$$

where $\mathcal{X} \times_a^b \mathcal{Y}$ can be computed as a reordering ('tensorization') of $\mathbf{Y}_{(b)}\mathbf{X}_{(a)}$, or more generally as

$$\mathcal{X}(i_1, \ldots, i_d) \approx \sum_{k_1=1}^{r_1} \sum_{k_2=1}^{r_2} \cdots \sum_{k_{d-1}=1}^{r_{d-1}} \mathbf{G}^{(1)}(i_1, k_1)\mathcal{G}^{(2)}(k_1, i_2, k_2) \cdots \mathcal{G}^{(d-1)}(k_{d-2}, i_{d-1}, k_{d-1})\mathbf{G}^{(d)}(k_{d-1}, i_d)$$

where $\mathbf{G}^{(1)} \in \mathbb{R}^{r_1 \times n_1}$, $\mathcal{G}^{(k)} \in \mathbb{R}^{r_{k-1} \times n_k \times r_k}$ for $1 < k < d$ and $\mathbf{G}^{(d)} \in \mathbb{R}^{R_{d-1} \times n_d}$. The relationship between tensors can be better understood in terms of a tensor network diagram. In this visualization, each node represents a tensor, and each line represents a mode, with the associated mode size. Lines that connect tensors represent the common mode that can be contracted to reconstruct the input tensor: The most straightforward way to compute a tensor

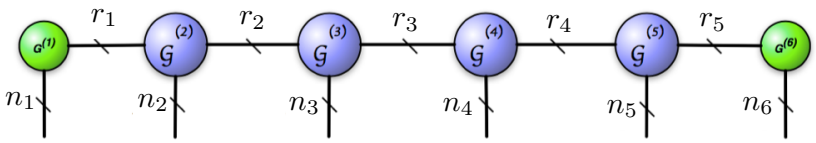


Figure 5: Tensor Train Decomposition

train decomposition is to flatten the tensor and compute a rank-revealing matrix decomposition of the result (as illustrated later in fig. 10). The left factor of this decomposition becomes a new node in the tensor train, while the remaining product is reshaped and used as an input for further iterations. There are also iterative methods that solve using a sequence of updates, including least squares [28, 23]. These generally operate by initializing random carriages $\mathcal{G}^{(k)}$ and then performing sweeps over each carriage in sequence, a process closely related to DMRG in quantum physics [65]. More scalable methods exist that look at only a portion of the data [44].

Tensor network representations such as the tensor train are particularly valuable in cases where the order of a tensor is so high that the dense core of the Tucker decomposition is not feasible to construct. Among many applications it has proven useful in deep learning [67, 42], supervised learning [56], and control systems [21]. It has also proven useful for the approximation of functions [22].

2.5 Randomized Least Squares

Sketching is a technique for solving linear algebra problems by constructing a smaller problem whose solution is a reasonable approximation to the original problem with high probability [66]. For instance, a large matrix may be formed by applying random sampling or random projections to form a smaller *sketch* matrix. We focus on the case where a regression problem $\min_{\mathbf{x}} \|\mathbf{Ax} - \mathbf{b}\|_2$ (with overdetermined $\mathbf{A} \in \mathbb{R}^{n \times d}$) is transformed using some random projection $\mathbf{M} \in \mathbb{R}^{S \times n}$, with $S \ll n$, such that an exact solution to $\min_{\mathbf{x}} \|\mathbf{MAx} - \mathbf{Mb}\|_2$ is an approximate solution to the original problem [51, 20, 6].

There are two leading sampling approaches for randomized least squares problems. One involves sampling from the coefficient matrix in a weighted manner, e.g. by computing (or estimating) *leverage scores* for each row and sampling based on their distribution. The other approach, which we

have implemented, is to *mix* the coefficient matrix with the intention of evenly distributing leverage scores across all rows in such a way that *uniform* sampling is effective.

Definition 2.1. Given $\mathbf{A} \in \mathbb{R}^{n \times d}, n > d$, the *leverage score* of row i of $\mathbf{A}$ is $l_i = ||\mathbf{U}(i,:)||_2^2$ for $i \in \{1, \ldots, n\}$ where $\mathbf{U}$ contains the d left singular vectors of $\mathbf{A}$.

Thus, the leverage score of a row corresponds in some sense to the importance of that row in constructing the column-space of the coefficient matrix.

In 2007, Drineas et al. [20] presented a relative-error least squares algorithm that gives a $1+\epsilon$ approximation. They first mix the coefficient matrix using a randomized Hadamard transform (discussed later), and then sample

$$\mathcal{O}(\max\{d \log(n) \log(d \log(n)), d \log(nd)/\epsilon\}) \tag{16}$$

rows of the resulting matrix before computing the solution using normal equations. The dependence of the sampling size on ϵ makes this algorithm fairly impractical for typical direct solvers. However, subsequent work by Rokhlin and Tygert [51] applied a related sketching strategy to the *preconditioning* of a Krylov-subspace method, establishing a relationship between sample size and condition number.

Avron et al. synthesized these concepts into a high-performance solver called Blendenpik [6]. They first apply a randomized Hadamard transform (or similar transform), compute a QR decomposition of the result, and use its $\mathbf{R}$ factor as a preconditioner for the standard LSQR solver. Additionally they show that the condition number of their system depends on the *maximal* leverage score of the matrix, referred to as *coherence*.

Definition 2.2 ([18, 10])**.** *Coherence* is the maximum leverage score of $\mathbf{A}$, i.e.,

$$\mu(\mathbf{A}) = \max_{i \in \{1,\ldots,n\}} l_i,$$

where l_i is the leverage score of the ith row of $\mathbf{A}$. It holds that $\frac{d}{n} \leq \mu(\mathbf{A}) \leq 1$.

Intuitively, if a row of a matrix $\mathbf{A}$ contains the only nonzero in a column then $\mu(\mathbf{A}) = 1$ and any row-sampling $\mathbf{SA}$ must include that row (which has leverage score 1) or it will be rank deficient. If coherence is close to 1, a uniform row-sampling is likely to be *nearly* rank-deficient, leading to

a poorly conditioned reduced-size least squares problem and an inaccurate approximate solution vector.

In [8] we show that the standard formulation of CP-ALS may increase incoherence, making uniform sampling effective in many situations. However, in order to guarantee incoherence (w.h.p.) regardless of input, it is necessary to preprocess with a mixing step.

This mixing strategy relates to a more general class of transformations that rely on quality guarantees provided by the Johnson-Lindenstrauss Lemma [31]. This lemma specifies a class of random projections that preserve the distances between all pairs of vectors with reasonable accuracy. The *fast* Johnson-Lindenstrauss transform (FJLT) is able to avoid explicit matrix multiplications by utilizing efficient algorithms such as the fast Fourier transform (FFT), discrete cosine transform (DCT), or Walsh-Hadamard Transform (WHT) [3]. These transforms can operate on a vector $\mathbf{x} \in \mathbb{R}^n$ in $\mathcal{O}(n \log_2 n)$ time. What these algorithms have in common is that they improve incoherence, mixing information across every element of a vector, while at the same time being orthogonal operations (i.e., a change of basis). The theoretical quality guarantees and theoretical computational costs are the same for all fast transforms.

The FJLT consists of three steps. First, each row of the coefficient matrix is sign-flipped with probability 1/2. This is equivalent to computing $\mathbf{DA}$ with diagonal matrix $\mathbf{D} \in \mathbb{R}^{n \times n}$, where each diagonal element is ± 1 with equal probability. Second, we apply the fast mixing operation $\mathcal{F}$. Third we uniformly sample S rows of the result with uniform probability. Thus, the entire operation can be written out as $\mathbf{S}\mathcal{F}\mathbf{DA}$, where $\mathbf{S}$ is a row-sampling operator (containing unit row vectors $\mathbf{e}_i$, for each sampled row i). The reasoning behind first applying $\mathbf{D}$ is that input data is often sparse in the frequency domain, and randomly flipping the signs of the coefficient matrix is an orthogonal operation that spreads out the frequency domain of the signal [3].

2.6 Randomized SVD

The MTFSBC requires finding the leading left singular vectors and singular values of the tensor matricization. Halko, Martinsson, and Tropp [24, Alg. 4.1] present a *randomized range finder* algorithm for estimating the left singular vectors of a matrix $\mathbf{X}$ of size $m \times n$ where n is very large. The goal is to find the leading r left singular values where it is assumed that the remaining singular values are small. The key idea is to multiply $\mathbf{X}$ on the

right by a sketching matrix $\mathbf{\Omega}$ of size $n \times q$ where $r < q \ll n$. The matrix $\mathbf{\Omega}$ is some appropriate random matrix; for instance, it could have i.i.d. entries drawn from a standard normal distribution. The result of the multiplication is a much smaller matrix of size $m \times q$ whose q leading left singular vectors approximately include the range of the r leading left singular vectors of $\mathbf{X}$. The value of $q - r$ is the *oversampling parameter*, and the is typically small, e.g., less than 20. We summarize the method in algorithm 3, and its expected error is analyzed in [24].

Algorithm 3 Randomized Range-Finder

1: **procedure** RRF($\mathbf{X}$, q) ▷ $\mathbf{X} \in \mathbb{R}^{m \times n}$
2: $\quad \mathbf{\Omega} \leftarrow$ suitable random matrix of size $n \times q$
3: $\quad \mathbf{Y} \leftarrow \mathbf{X\Omega}$
4: $\quad \mathbf{U} \leftarrow$ orthonormal basis of size $m \times q$ for $(\mathbf{Y})$
5: **end procedure**

For this method to be efficient, the desired rank should be small, i.e., $r \ll n$. If $m \ll n$, then we know that $r \leq m \ll n$, so this property is satisfied. Indeed, we will observe that the matrix unfoldings in the MTFSBC have exactly that property since we are working with matrices of roughly $n \times n^{d-1}$ in size.

Sketching methods of this nature have become popular within matrix decompositions [57], but have only recently seen use in tensor methods or distributed computation.

3 Literature Survey

A broad survey of tensor decompositions is provided by Kolda and Bader [36]. Additional papers outline specific tensor techniques in machine learning [52], quantum chemistry [35], latent variable models [4], and neuroscience [14]. In this proposal we focus on scalable implementations of three of the most popular tensor decompositions: the Candecomp/Parafac (CP) decomposition [27, 11, 25], the Tucker decomposition [59], and the Tensor Train decomposition [43].

Scalable CP Decompositions There are many methods that improve performance of the CP decomposition by finding efficient ways of computing

MTTKRP kernel within CP-ALS [54, 33, 26, 38]. Since we propose randomized methods that actually reduce flop count, we survey those methods in more detail:

Vervliet and Lauthauwer present a stochastic gradient descent (SGD) algorithm for CP that samples blocks from the original tensor to update corresponding blocks of the factor matrices [61]. This approach is similar in spirit to CPRAND, but takes an altogether different approach to the randomization. They use *contiguous* samples in the block updates and finer control over step sizes. They also update only a portion of each factor matrix in each iteration.

Another framework that draws from SGD is FlexiFaCT, which targets coupled tensor decompositions for parallel computation [9]

Cheng et al. have recently applied a leverage score-based sampling to the least-squares step of the sparse CP decomposition by showing how leverage scores of an unfolded tensor can be estimated by the leverage scores of the factor matrices [12]. This approach is similar to the way that we bound the coherence of Khatri-Rao products. Reynolds et al. also use randomization within CP-ALS, specifically for the case of rank reduction, where the input to the algorithm is already in CP format [50].

Wang et al. have applied sketching methods to *orthogonal* tensors with provable guarantees [64]. Song et al. show that this sketch can be computed without reading the entire tensor (in sublinear time) under certain conditions [55].

An alternative to sketching is to compress the tensor using lossy methods before computation. Zhou and Cichocki examine the effectiveness of performing a CP decomposition on a compressed representation of the data using the lossy Tucker decomposition to produce a smaller problem size [68]. ParCube [46] compresses the original tensor by directly sampling and performs a decomposition on the result. Another useful approach for high-order tensors is known as tensor reshaping, that can cast much of the computation in terms of three-way tensors [49].

Scalable Tucker Decompositions The current state of the art for the distributed, deterministic dense HOSVD is TuckerMPI, based on the work of Austin, Ballard, and Kolda [5], whereas Kaya and Ucar have optimized the Tucker decomposition in distributed memory for sparse tensors [34]. In shared memory, optimizations have targeted the core formation step [37], or

developing compressed data structures [53].

Zhou, Cichocki, and Xie have previously implemented an HOSVD based on the Randomized Ranger Finder algorithm [68]. They implement RandTucker, which performs a Randomized Range Finder on the *full* matricizations of the tensor, and RandTucker2i, which performs a second iteration after projecting with the first set of estimated factor matrices.

Vervliet, Debals, and De Lauthawer similarly implement a randomized Tucker in their serial MATLAB toolbox TensorLab3[62]. This allows the user to perform an arbitrary number of iterations that improve the final result. Navasca and Pompey [41] also present a serial HOSVD implementation that uses the randomized SVD on the matricizations in each mode, but it does not take any optimizations into account. Other randomized methods for computing the HOSVD have focused on sampling directly from the tensor [58, 19].

Our proposed work differs from earlier approaches [68, 62, 41] in that we target a distributed-memory implementation that performs sketching as a series of small TTMs (equivalent to a Kronecker product of sketching matrices). We also propose to target optimization of the communication involved in this sketching step.

Scalable Tensor Train Decompositions There is not much literature specifically on high-performance tensor train decompositions, so it is a valuable area for future work. Phan, et al. have compared various methods [48], and cross-approximation approaches allow for computation on only a subset of data [44], but in general the emphasis has been placed on algorithms that operate efficiently by using a tensor train representation as an intermediate representation [13].

Randomization has been applied to the Tensor Train decomposition in existing work [29], but the emphasis has not been on performance. There is also no existing distributed implementation of these methods. Finally, a related tensor network decomposition, the Hierarchical Tucker decomposition, has been successfully targeted with sampling methods [47].

4 Randomized CP Decomposition

The CANDECOMP/PARAFAC (CP) tensor decomposition is an important tool for data analysis in applications such as chemometrics [40], biogeochem-

istry [30], neuroscience [1, 16, 15], cyber traffic analysis [39], and many others. We consider the problem of accelerating the alternating least squares (CP-ALS) algorithm using randomization.

Because randomized methods have been used successfully for solving linear least squares problems [20, 6, 66], it is natural that they might prove beneficial to CP-ALS since its key kernel is the solution of a least squares problem. However, the CP-ALS least squares subproblem has a special structure that already greatly reduces its cost (see eq. (13)), so it is not obvious that sketching would be beneficial. Nevertheless, we find that our randomized algorithms significantly reduce the memory and computational overhead of the CP-ALS process for dense tensors to increase performance (fig. 7) and moreover positively impact algorithmic robustness (fig. 6). To the best of our knowledge, this is the first successful application of matrix sketching methods in the context of CP. Our contributions are as follows:

- The least squares coefficient matrix in the CP-ALS subproblem is a Khatri-Rao product of factor matrices. Our randomized algorithm prefers *incoherent* matrices. We prove that the coherence of the Khatri-Rao product is bounded above by the product of the coherence of its factors:

  Lemma 4.1. *Given* $\mathbf{A} \in \mathbb{R}^{I \times J}$ *and* $\mathbf{B} \in \mathbb{R}^{K \times L}$, $\mu(\mathbf{A} \odot \mathbf{B}) \leq \mu(\mathbf{A})\mu(\mathbf{B})$.

- We introduce the CPRAND algorithm that uses a randomized least squares solver for the subproblems in CP-ALS and never explicitly forms the full Khatri-Rao matrices used in the subproblems. We also introduce the complementary CPRAND-MIX algorithm that employs efficient *mixing* to promote incoherence and thereby improves the robustness of the method.

- We derive a novel, lightweight stopping condition that estimates the model fit error, and we prove its accuracy using Chernoff-Hoeffding bounds.

- We demonstrate the speed and robustness of our algorithms over a large number of synthetic tensors as well as real-world data sets. In comparison with CP-ALS, CPRAND is faster and much less sensitive to the starting point. (For instance, see fig. 7 for speed and fig. 6 for robustness).

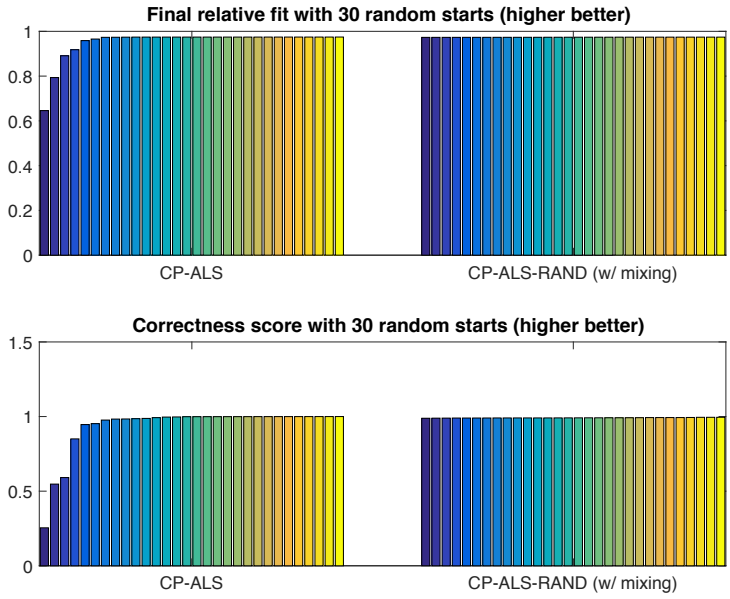
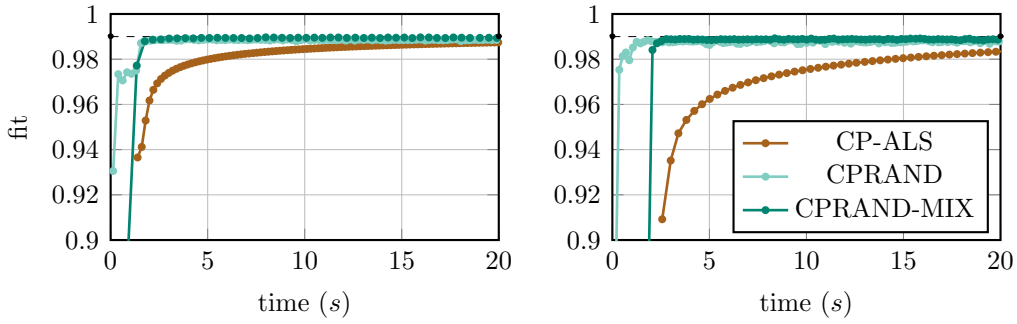


Figure 6: To demonstrate robustness, we show fit and score over 30 runs of a standard CP-ALS algorithm vs. our algorithm (CP-ALS-RAND with mixing) on a $5 \times 201 \times 61$ tensor drawn from chemometrics. Fit corresponds to the normalized residual error (eq. (2)) subtracted from 1. Score measures how well the recovered factors correspond to the best known actual factors.

We give an example of our methods' fast time to solution in fig. 7, comparing CPRAND and CPRAND-MIX with CP-ALS. For the CPRAND methods, we use 100 sampled rows for each least squares solve. The randomized methods converge much more quickly, in only a few iterations. The fit is not monotonically increasing for the randomized methods due to (small) variations in the solution to each randomized subproblem.

5 Randomized Tucker Decomposition

A standard method for computing the Tucker decomposition is the Higher-Order SVD (HOSVD) [17], a generalization of the singular value decomposition to tensors. As mentioned in section 2.3, a direct approach to the HOSVD involves computing a series of SVDs of unfolded tensors to compute the orthogonal factor matrices whose columns approximately span the



(a) Random $300 \times 300 \times 300$ tensor

(b) Random $80 \times 80 \times 80 \times 80$ tensor

Figure 7: Runtime comparison for fitting the CP tensor decomposition on random synthetic tensors generated to have rank 5, factor collinearity of 0.9, and 1% noise. We compare a single run of three methods using a target rank of 5. CPRAND and CPRAND-MIX use random initialization, 100 sampled rows for each least squares solve. CP-ALS uses HOSVD initialization. The marks indicate each iteration. The thin dashed black line represents a fit of 99%, which is the best we expect when the noise is 1%.

columns of the unfolded tensor (i.e., the mode-k fiber space). We will refer to this computation as the Mode-wise Truncated Fiber Space Basis Computation (MTFSBC). Following this operation, the resulting factor matrices are applied to the input tensor to compress the original data.

We consider a distributed-memory application for large dense tensors, as might arise in large-scale scientific simulations. For instance, we consider a five-way tensor representing three spatial dimensions, a time dimension, and a number of different variables. A user may wish to examine this data on a local machine with limited memory or to store many simulations in limited storage, and the HOSVD has been shown to be a highly scalable method for performing this compression. The current state of the art for distributed HOSVD is TuckerMPI[4], based on the work of Austin, Ballard, and Kolda [5]. As in the Gram approach of section 2.3, the MTFSBC kernel is performed with a distributed Gram matrix computation followed by a local eigensolve. The factors are then applied to the input tensor using a distributed tensor-matrix multiplication, forming the dense core tensor. The core tensor and factor matrices can be used to construct an approximation to the original

[4] `http://tensors.gitlab.io/TuckerMPI/`

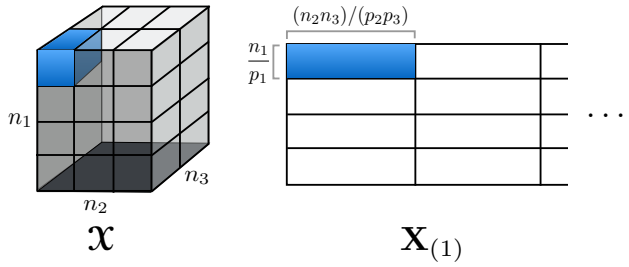


Figure 8: A tensor on a $4 \times 3 \times 2$ processor grid. The matricized subtensor shown in blue maps to a submatrix in the distributed matricized tensor, as illustrated on the right.

data or any subset thereof.

Distributed implementation We will assume the tensor is distributed according to a *Cartesian* distribution. Given a processor grid $p_1 \times \cdots \times p_d$, we define notation such that the total number of processors is $p^d = \prod_{k=1}^{d} p_k$. We use an overbar to denote a local object, so each process owns a tensor $\bar{\mathcal{X}}$ which is a subtensor of $\mathcal{X}$. Assuming for notational convenience that n_k evenly divides p_k, the size of each subtensor is $\frac{n_1}{p_1} \times \cdots \times \frac{n_d}{p_d}$, so process $(\ell_1, \ldots, \ell_d)$ owns the subtensor with range $\mathbf{i} = ((\ell_1 - 1)\frac{n_1}{p_1} : \ell_1 \frac{n_1}{p_1} - 1, \ldots)$. Furthermore, if we matricize a subtensor in mode k to produce $\bar{\mathbf{X}}_{(k)}$, the result is a submatrix in the full matricization $\mathbf{X}_{(k)}$. This is visualized in fig. 8, though the columns owned by a processor are not always contiguous as shown here. Collective communications in this model are then implemented either on block-rows or block-columns.

The column-communicator for process $(\ell_1, \ldots, \ell_d)$ in mode k is the set of p_k processes $(\ell_1, \ldots, \ell_{k-1}, *, \ell_{k+1}, \ldots, \ell_d)$, whereas the row-communicator is the set of $p_k^{\oslash}$ processes $(*, \ldots, *, \ell_k, *, \ldots, *)$. These communicators are visualized in fig. 9.

Proposal We propose efficiently performing the MTFSBC kernel in distributed memory by applying methods from randomized numerical linear algebra. We propose that a series of local multiplications followed by a small global communication can result in a *sketch* of the original tensor that fits in local memory. Finally, we propose to apply this to the efficient computation of a Tucker decomposition for the compression of scientific data. We

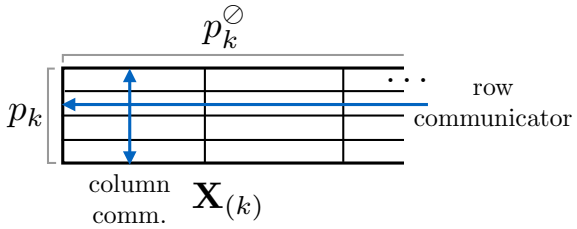


Figure 9: A process-centric view of fig. 8. The column-communicator in mode k is the set of p_k processes that own blocks in the same column of $\mathbf{X}_{(k)}$, while the row-communicator is the set of $p_k^{\oslash}$ processes that own blocks in the same row.

summarize our proposed contributions as follows:

- Implement and develop a cost analysis for an efficient sketch-based MTFSBC in distributed memory, using randomization to reduce computation and communication. This method would exploit the Kronecker structure of the sketching operation to reduce computation.
- Introduce a new parallel method for a sequence of tensor-times-matrix operations that trades extra computation for reduced communication and show how and when this can be used to further increase the efficiency of sketching.
- Apply our kernel to the Tucker tensor decomposition, benchmark our method on large-scale scientific data sets, and demonstrate scaling on synthetic data sets to improve upon the fastest known traditional implementation.
- Provide theory to show how the error introduced through randomization can be compensated for.

Sketching as a series of TTMs Consider the algorithm for estimating the range of a matrix in algorithm 3. Similarly, to estimate the range of an unfolded tensor we can compute $\mathbf{Y}_{(k)} \leftarrow \mathbf{X}_{(k)}\mathbf{\Omega}$, where $\mathbf{\Omega}$ is a $n_k^{\oslash} \times \ell$ random matrix. This approach has previously been taken for tensors [68, 62, 41]. The cost of this step would be comparable to a TTM, which could easily become a bottleneck. However, suppose instead that we multiply with a *Kronecker*

product of $d-1$ small random matrices of size $s_k \times n_k$:

$$\mathbf{Y}_{(k)} \leftarrow \mathbf{X}_{(k)}\Big(\mathbf{\Omega}_1 \otimes \cdots \otimes \mathbf{\Omega}_{k-1} \otimes \mathbf{\Omega}_{k+1} \otimes \cdots \otimes \mathbf{\Omega}_d\Big)^{\mathsf{T}} \tag{17}$$

Using eq. (8), we can see that this is equivalent to performing a series of TTMs with these small matrices:

$$\mathcal{Y} \leftarrow \mathcal{X} \times \left\{\mathbf{\Omega}_j^{\mathsf{T}}\right\}_{j \neq k}. \tag{18}$$

We propose to use this method because it uses far fewer flops than previous work, and the operation can be efficiently performed in distributed memory as a series of distributed TTMs.

Reducing communication of a TTM sequence We present a novel method for computing a TTM sequence in distributed memory which outperforms standard methods when the input matrices have a large aspect ratio (so that the output tensor is much smaller than the input tensor). This method significantly reduces communication in certain cases but may be more expensive in others. It also performs more computation than the standard approach, so we use a performance model to guide its usage.

Consider computing a sequence of TTMs $\mathcal{Y} \leftarrow \mathcal{X} \times_1 \mathbf{A} \times_2 \mathbf{B} \times_3 \mathbf{C}$. Previous implementations of the TTM [5] first perform the local multiply $\bar{\mathbf{A}}\bar{\mathbf{X}}_{(1)}$ on each process, forming an intermediate result of size $s_1 \times n_1^{\oslash}/p_1^{\oslash}$. This is followed by a Reduce-Scatter among the column communicator so that each process owns a local result of size $s_1/p_1 \times n_1^{\oslash}/p_1^{\oslash}$, and the process is repeated for the following modes $k = 2$ and $k = 3$. If the matrices $\mathbf{A}$, $\mathbf{B}$, and $\mathbf{C}$ have more columns than rows, then the tensor gets smaller at each step. While this standard method is work efficient, it involves communicating the intermediate tensors in each step, which are larger than the final output.

We propose that this communication can be *deferred* until later steps, while still maintaining the correct output. We will call this proposed deferred-communication approach *dc-TTM*, noting that element (i, j, k) of $\mathcal{Y}$ can be written element-wise using eq. (7) as

$$y_{ijk} = \sum_{\mu} \sum_{\nu} \sum_{\xi} x_{\mu\nu\xi} a_{\mu i} b_{\nu j} c_{\xi k}. \tag{19}$$

While this approach isn't work optimal, the reduction in communication should yield useful applications, particularly for low-flop operations such as

sketching. We propose to elaborate on this communication/computation tradeoff.

6 Randomized Tensor Train Decomposition

The exact 'quasi-optimal' scheme for building a tensor train is TT-SVD [43], where the first two steps for a 4-way tensor are illustrated in fig. 10. As shown, This simply involves a series of reshape operations (permutations) followed by skinny SVDs. Provided that reshape operations can be efficiently performed in distributed memory (or avoided altogether), we can implement this method efficiently in distributed memory (using a 'Gram' style SVD approach, as in algorithm 2). This would be the first know distribute implementation of a tensor train decomposition.

Furthermore, both common solution methods for computing a tensor train decomposition (SVD or optimization) may be able to efficiently leverage the previously discussed ideas from randomized SVD and randomized least squares, respectively. Thus we propose to explore the following questions:

- Can 'sweeping' optimization algorithms for the tensor train be written in a way that can leverage randomized least squares, e.g., as described in section 2.5?
- Can the SVD method for forming the tensor train decomposition be expressed in a way that can be efficiently performed in distributed memory? E.g., starting from the distributed tensor representation outlined in section 5.
- Can ideas from randomized low-rank decompositions be applied efficiently to this formulation in distributed memory? E.g., using methods from [57].

Proposal We summarize our proposed contributions as follows:

- Implement the first distributed-memory implementation of the tensor train decomposition.
- Analyze the communication and computation behavior of the 'reshape-SVD' method in distributed memory.

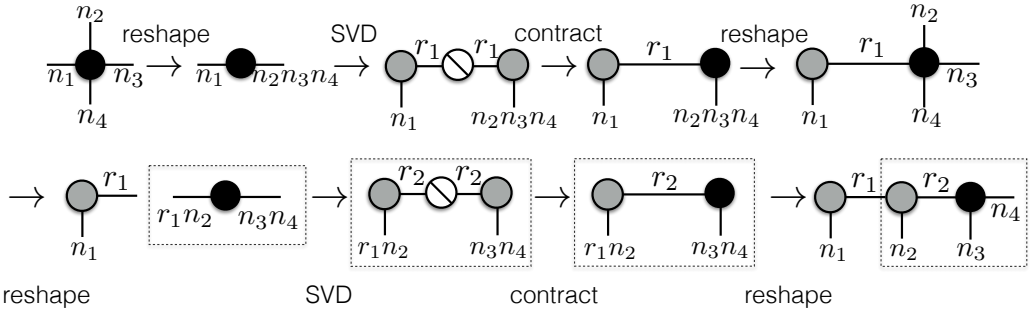


Figure 10: The creation of the first two 'carriages' of a tensor train decomposition for a 4-way tensor.

- Apply methods from our randomized Tucker decomposition to the tensor train in distributed memory.
- Benchmark both methods on a supercomputer and analyze/establish tradeoffs.

7 Summary

- Randomized CP Decomposition.
  - ☑ Introduce a randomized CP decomposition drawing from randomized least squares.
  - ☑ Introduce a 'mixed' method for the randomized CP decomposition
  - ☑ Establish how to perform sampling within CP-ALS without forming any large matrix products.
  - ☑ Benchmark; establish speed and robustness.
  - ☑ Introduce a faster, randomized stopping condition and establish theoretical bounds.
  - ☑ Deliverable: Include methods in MATLAB Tensor Toolbox.
- Randomized Tucker Decomposition.
  - ☑ Introduce a parallel method for sketch-based MTFSBC that uses a series of TTMs.

- ☑ Implement a sketch-based MTFSBC in distributed memory.
- ☑ Develop a cost analysis of this method.
- ☑ Introduce a new parallel method for a sequence of tensor-times-matrix operations that trades extra computation for reduced communication
- ☐ Analyze tradeoffs
- ☐ Apply our kernel to the Tucker tensor decomposition, benchmark our method on large-scale scientific data sets
- ☐ Demonstrate scaling on synthetic data sets.
- ☐ Provide theory to show how the error introduced through randomization can be compensated for.
- ☐ Deliverable: Include HOSVD-Sketch algorithm in TuckerMPI.

- Randomized Tensor Train Decomposition.
  - ☐ Implement the first distributed-memory implementation of the tensor train decomposition.
  - ☐ Analyze the communication and computation behavior of the 'reshape-SVD' method in distributed memory.
  - ☐ Analyze possibilities for randomized least squares in the 'sweeping' tensor train algorithm.
  - ☐ Apply methods from our randomized Tucker decomposition to the tensor train in distributed memory, leveraging the lop-sided dimensionality of the working tensor.
  - ☐ Benchmark both methods on a supercomputer and analyze/establish tradeoffs.

In all three proposed sections we emphasize the core of this thesis, which is that the dimensionality of tensors lends itself very well to randomized methods, which we propose to demonstrate in both shared and distributed memory.

Bibliography

[1] E. Acar, C. A. Bingol, H. Bingol, R. Bro, and B. Yener, *Multiway analysis of epilepsy tensors*, Bioinformatics, 23 (2007), pp. i10–i18, https://doi.org/10.1093/bioinformatics/btm210.

[2] E. Acar, R. J. Harrison, F. Olken, O. Alter, M. Helal, L. Omberg, B. Bader, A. Kennedy, H. Park, Z. Bai, D. Kim, R. Plemmons, G. Beylkin, T. Kolda, S. Ragnarsson, L. Delathauwer, J. Langou, S. P. Ponnapalli, I. Dhillon, L.-h. Lim, J. R. Ramanujam, C. Ding, M. Mahoney, J. Raynolds, L. Eldn, C. Martin, P. Regalia, P. Drineas, M. Mohlenkamp, C. Faloutsos, J. Morton, B. Savas, S. Friedland, L. Mullin, and C. Van Loan, *Future directions in tensor-based computation and modeling*, 2009.

[3] N. Ailon and B. Chazelle, *Approximate nearest neighbors and the fast Johnson-Lindenstrauss transform*, in Proceedings of the Thirty-eighth Annual ACM Symposium on Theory of Computing, STOC'06, ACM, 2006, pp. 557–563, https://doi.org/10.1145/1132516.1132597.

[4] A. Anandkumar, R. Ge, D. Hsu, S. M. Kakade, and M. Telgarsky, *Tensor decompositions for learning latent variable models*, J. Mach. Learn. Res., 15 (2014), pp. 2773–2832, http://dl.acm.org/citation.cfm?id=2627435.2697055.

[5] W. Austin, G. Ballard, and T. G. Kolda, *Parallel tensor compression for large-scale scientific data*, in IPDPS'16: Proceedings of the 30th IEEE International Parallel and Distributed Processing Symposium, May 2016, pp. 912–922, https://doi.org/10.1109/IPDPS.2016.67.

[6] H. Avron, P. Maymounkov, and S. Toledo, *Blendenpik: Supercharging LAPACK's least-squares solver*, SIAM J. Scientific Computing, 32 (2010), pp. 1217–1236, https://doi.org/10.1137/090767911.

[7] B. W. Bader, T. G. Kolda, et al., *Matlab tensor toolbox (Version 2.6)*. Available online, February 2015, http://www.sandia.gov/~tgkolda/TensorToolbox/.

[8] C. Battaglino, G. Ballard, and T. G. Kolda, *A practical randomized cp tensor decomposition*, SIAM Journal on Matrix Analysis and Applications, 39 (2017).

[9] A. Beutel, P. P. Talukdar, A. Kumar, C. Faloutsos, E. E. Papalexakis, and E. P. Xing, *FlexiFaCT: Scalable flexible factorization of coupled tensors on Hadoop.*, in Proc. SIAM Intl. Conf. Data Mining, SDM'14, SIAM, 2014, pp. 109–117, https://doi.org/10.1137/1.9781611973440.13.

[10] E. Candès and B. Recht, *Exact matrix completion via convex optimization*, Commun. ACM, 55 (2012), pp. 111–119, https://doi.org/10.1145/2184319.2184343.

[11] J. Carroll and J.-J. Chang, *Analysis of individual differences in multidimensional scaling via an n-way generalization of "Eckart-Young" decomposition*, Psychometrika, 35 (1970), pp. 283–319, https://doi.org/10.1007/BF02310791.

[12] D. Cheng, R. Peng, I. Perros, and Y. Liu, *SPALS: Fast alternating least squares via implicit leverage scores sampling*, in Advances in Neural Information Processing Systems (NIPS) 30, 2016.

[13] A. Cichocki, A. H. Phan, Q. Zhao, N. Lee, I. V. Oseledets, M. Sugiyama, and D. P. Mandic, *Tensor networks for dimensionality reduction and large-scale optimizations. part 2 applications and future perspectives*, CoRR, abs/1708.09165 (2017), http://arxiv.org/abs/1708.09165, https://arxiv.org/abs/1708.09165.

[14] F. Cong, Q.-H. Lin, L.-D. Kuang, X.-F. Gong, P. Astikainen, and T. Ristaniemi, *Tensor decomposition of eeg signals: A brief review*, Journal of Neuroscience Methods, 248 (2015), pp. 59 – 69,

https://doi.org/https://doi.org/10.1016/j.jneumeth.2015.03.018, http://www.sciencedirect.com/science/article/pii/S0165027015001016.

[15] F. Cong, Q.-H. Lin, L.-D. Kuang, X.-F. Gong, P. Astikainen, and T. Ristaniemi, *Tensor decomposition of EEG signals: A brief review*, Journal of Neuroscience Methods, 248 (2015), pp. 59–69, https://doi.org/10.1016/j.jneumeth.2015.03.018.

[16] I. Davidson, S. Gilpin, O. Carmichael, and P. Walker, *Network discovery via constrained tensor analysis of fMRI data*, in KDD'13: Proceedings of the 19th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, ACM, 2013, pp. 194–202, https://doi.org/10.1145/2487575.2487619.

[17] L. De Lathauwer, B. De Moor, and J. Vandewalle, *A multilinear singular value decomposition*, SIAM J. Matrix Anal. Appl, 21 (2000), pp. 1253–1278, https://doi.org/10.1137/S0895479896305696.

[18] D. L. Donoho and X. Huo, *Uncertainty principles and ideal atomic decomposition*, IEEE Trans. Inf. Theor., 47 (2006), pp. 2845–2862, https://doi.org/10.1109/18.959265.

[19] P. Drineas and M. W. Mahoney, *A randomized algorithm for a tensor-based generalization of the singular value decomposition*, Linear Algebra and its Applications, 420 (2007), pp. 553 – 571, https://doi.org/10.1016/j.laa.2006.08.023.

[20] P. Drineas, M. W. Mahoney, S. Muthukrishnan, and T. Sarlós, *Faster least squares approximation*, Numerische Mathematik, 117 (2011), pp. 219–249, https://doi.org/10.1007/s00211-010-0331-6.

[21] A. Gorodetsky, S. Karaman, and Y. Marzouk, *Efficient high-dimensional stochastic optimal motion control using tensor-train decomposition*, in Proceedings of Robotics: Science and Systems, Rome, Italy, July 2015, https://doi.org/10.15607/RSS.2015.XI.015.

[22] A. A. Gorodetsky, S. Karaman, and Y. M. Marzouk, *Function-train: A continuous analogue of the tensor-train decomposition*, 2015, https://arxiv.org/abs/1510.09088v2.

[23] L. Grasedyck, M. Kluge, and S. Krmer, *Variants of alternating least squares tensor completion in the tensor train format*, SIAM Journal on Scientific Computing, 37 (2015), pp. A2424–A2450, https://doi.org/10.1137/130942401.

[24] N. Halko, P. G. Martinsson, and J. A. Tropp, *Finding structure with randomness: Probabilistic algorithms for constructing approximate matrix decompositions*, SIAM Rev., 53 (2011), pp. 217–288, https://doi.org/10.1137/090771806.

[25] R. A. Harshman, *Foundations of the PARAFAC procedure: Models and conditions for an "explanatory" multi-modal factor analysis*, UCLA Working Papers in Phonetics, 16 (1970).

[26] K. Hayashi, G. Ballard, Y. Jiang, and M. J. Tobia, *Shared-memory parallelization of mttkrp for dense tensors*, in Proceedings of the 23rd ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming, PPoPP '18, New York, NY, USA, 2018, ACM, pp. 393–394, https://doi.org/10.1145/3178487.3178522, http://doi.acm.org/10.1145/3178487.3178522.

[27] F. L. Hitchcock, *The expression of a tensor or a polyadic as a sum of products*, J. Math. Phys, 6 (1927), pp. 164–189, https://doi.org/10.1002/sapm192761164.

[28] S. Holtz, T. Rohwedder, and R. Schneider, *The alternating linear scheme for tensor optimization in the tensor train format*, SIAM Journal on Scientific Computing, 34 (2012), pp. A683–A713, https://doi.org/10.1137/100818893, https://doi.org/10.1137/100818893, https://arxiv.org/abs/https://doi.org/10.1137/100818893.

[29] B. Huber, R. Schneider, and S. Wolf, *A Randomized Tensor Train Singular Value Decomposition*, Springer International Publishing, Cham, 2017, ch. 9, pp. 261–290, https://doi.org/10.1007/978-3-319-69802-1_9.

[30] R. Jaffé, K. M. Cawley, and Y. Yamashita, *Applications of excitation emission matrix fluorescence with parallel factor analysis (EEM-PARAFAC) in assessing environmental dynamics of natural dissolved*

organic matter (DOM) in aquatic environments: A review, in Advances in the Physicochemical Characterization of Dissolved Organic Matter: Impact on Natural and Engineered Systems, vol. 1160 of ACS Symposium Series, American Chemical Society (ACS), 2014, pp. 27–73, https://doi.org/10.1021/bk-2014-1160.ch003.

[31] W. Johnson and J. Lindenstrauss, *Extensions of Lipschitz mappings into a Hilbert space*, in Conference in Modern Analysis and Probability (New Haven, Conn., 1982), vol. 26 of Contemporary Mathematics, American Mathematical Society, 1984, pp. 189–206, https://doi.org/10.1007/BF02764938.

[32] A. K. Smilde, R. Bro, and P. Geladi, *Multi way analysis applications in chemical sciences*, (2004).

[33] O. Kaya, Y. Robert, and B. Uçar, *Computing Dense Tensor Decompositions with Optimal Dimension Trees*, Research Report RR-9080, Inria, July 2017, https://hal.inria.fr/hal-01562399.

[34] O. Kaya and B. Uar, *High performance parallel algorithms for the tucker decomposition of sparse tensors*, 08 2016.

[35] V. Khoromskaia and B. N. Khoromskij, *Tensor numerical methods in quantum chemistry : from hartree-fock energy to excited states*, 2015.

[36] T. G. Kolda and B. W. Bader, *Tensor decompositions and applications*, SIAM Review, 51 (2009), pp. 455–500, https://doi.org/10.1137/07070111X.

[37] J. Li, C. Battaglino, I. Perros, J. Sun, and R. Vuduc, *An input-adaptive and in-place approach to dense tensor-times-matrix multiply*, in Proc. of the ACM/IEEE Conference on Supercomputing (SC '15), Austin, TX, USA, 2015, https://doi.org/10.1145/2807591.2807671.

[38] J. Li, J. Choi, I. Perros, J. Sun, and R. Vuduc, *Model-driven sparse cp decomposition for higher-order tensors*, in 2017 IEEE International Parallel and Distributed Processing Symposium (IPDPS), vol. 00, May 2017, pp. 1048–1057, https://doi.org/10.1109/IPDPS.2017.80, doi.ieeecomputersociety.org/10.1109/IPDPS.2017.80.

[39] K. Maruhashi, F. Guo, and C. Faloutsos, *MultiAspectForensics: Pattern mining on large-scale heterogeneous networks with tensor analysis*, in 2011 International Conference on Advances in Social Networks Analysis and Mining (ASONAM), IEEE, 2011, pp. 203–210, https://doi.org/10.1109/asonam.2011.80.

[40] K. R. Murphy, C. A. Stedmon, D. Graeber, and R. Bro, *Fluorescence spectroscopy and multi-way techniques. PARAFAC*, Analytical Methods, 5 (2013), p. 6557, https://doi.org/10.1039/c3ay41160e.

[41] C. Navasca and D. N. Pompey, *Random Projections for Low Multilinear Rank Tensors*, Springer International Publishing, 2015, pp. 93–106, https://doi.org/10.1007/978-3-319-15090-1_5.

[42] A. Novikov, D. Podoprikhin, A. Osokin, and D. P. Vetrov, *Tensorizing neural networks*, in Advances in Neural Information Processing Systems 28, C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, and R. Garnett, eds., Curran Associates, Inc., 2015, pp. 442–450, http://papers.nips.cc/paper/5787-tensorizing-neural-networks.pdf.

[43] I. Oseledets, *Tensor-train decomposition*, SIAM Journal on Scientific Computing, 33 (2011), pp. 2295–2317, https://doi.org/10.1137/090752286, https://doi.org/10.1137/090752286, https://arxiv.org/abs/https://doi.org/10.1137/090752286.

[44] I. Oseledets and E. Tyrtyshnikov, *Tt-cross approximation for multidimensional arrays*, 432 (2010), pp. 70–88.

[45] S. Östlund and S. Rommer, *Thermodynamic limit of density matrix renormalization*, Phys. Rev. Lett., 75 (1995), pp. 3537–3540, https://doi.org/10.1103/PhysRevLett.75.3537, https://link.aps.org/doi/10.1103/PhysRevLett.75.3537.

[46] E. E. Papalexakis, C. Faloutsos, and N. D. Sidiropoulos, *ParCube: Sparse parallelizable tensor decompositions.*, in Machine Learning and Knowledge Discovery in Databases (European Conference, ECML PKDD 2012), vol. 7523 of Lecture Notes in Computer Science, Springer, 2012, pp. 521–536, https://doi.org/10.1007/978-3-642-33460-3_39.

[47] I. Perros, R. Chen, R. Vuduc, and J. Sun, *Sparse hierarchical tucker factorization and its application to healthcare*, IEEE International Conference on Data Mining (ICDM), (2015).

[48] A. H. Phan, A. Cichocki, A. Uschmajew, P. Tichavský, G. Luta, and D. P. Mandic, *Tensor networks for latent variable analysis. part i: Algorithms for tensor train decomposition*, CoRR, abs/1609.09230 (2016).

[49] A.-H. Phan, P. Tichavský, and A. Cichocki, *CANDECOMP/PARAFAC decomposition of high-order tensors through tensor reshaping*, IEEE Transactions on Signal Processing, 61 (2013), pp. 4847–4860, https://doi.org/10.1109/TSP.2013.2269046.

[50] M. J. Reynolds, A. Doostan, and G. Beylkin, *Randomized alternating least squares for canonical tensor decompositions: Application to a PDE with random data*, SIAM Journal on Scientific Computing, 38 (2016), pp. A2634–A2664, https://doi.org/10.1137/15M1042802.

[51] V. Rokhlin and M. Tygert, *A fast randomized algorithm for overdetermined linear least-squares regression*, Proceedings of the National Academy of Sciences, 105 (2008), pp. 13212–13217, https://doi.org/10.1073/pnas.0804869105.

[52] N. D. Sidiropoulos, L. D. Lathauwer, X. Fu, K. Huang, E. E. Papalexakis, and C. Faloutsos, *Tensor decomposition for signal processing and machine learning*, IEEE Transactions on Signal Processing, 65 (2017), pp. 3551–3582, https://doi.org/10.1109/TSP.2017.2690524.

[53] S. Smith and G. Karypis, *Accelerating the tucker decomposition with compressed sparse tensors*, in Euro-Par 2017: Parallel Processing, F. F. Rivera, T. F. Pena, and J. C. Cabaleiro, eds., Cham, 2017, Springer International Publishing, pp. 653–668.

[54] S. Smith, N. Ravindran, N. Sidiropoulos, and G. Karypis, *SPLATT: Efficient and parallel sparse tensor-matrix multiplication*, in Proceedings of the 29th IEEE International Parallel & Distributed Processing Symposium, IPDPS, 2015, https://doi.org/10.1109/IPDPS.2015.27.

[55] Z. Song, D. P. Woodruff, and H. Zhang, *Sublinear time orthogonal tensor decomposition*, in Advances in Neural Information Processing Systems (NIPS) 30, 2016, https://papers.nips.cc/paper/6495-sublinear-time-orthogonal-tensor-decomposition.pdf.

[56] E. Stoudenmire and D. J. Schwab, *Supervised learning with tensor networks*, in Advances in Neural Information Processing Systems 29, D. D. Lee, M. Sugiyama, U. V. Luxburg, I. Guyon, and R. Garnett, eds., Curran Associates, Inc., 2016, pp. 4799–4807, http://papers.nips.cc/paper/6211-supervised-learning-with-tensor-networks.pdf.

[57] J. A. Tropp, A. Yurtsever, M. Udell, and V. Cevher, *Practical sketching algorithms for low-rank matrix approximation*, SIAM J. Matrix Analysis Applications, 38 (2017), pp. 1454–1485, https://doi.org/10.1137/17M1111590, https://doi.org/10.1137/17M1111590.

[58] C. E. Tsourakakis, *MACH: Fast Randomized Tensor Decompositions*, 2010, pp. 689–700, https://doi.org/10.1137/1.9781611972801.60.

[59] L. R. Tucker, *Some mathematical notes on three-mode factor analysis*, Psychometrika, 31 (1966), pp. 279–311, https://doi.org/10.1007/BF02289464.

[60] N. Vannieuwenhoven, R. Vandebril, and K. Meerbergen, *A new truncation strategy for the higher-order singular value decomposition*, SIAM Journal on Scientific Computing, 34 (2012), pp. A1027–A1052, https://doi.org/10.1137/110836067.

[61] N. Vervliet and L. De Lathauwer, *A randomized block sampling approach to canonical polyadic decomposition of large-scale tensors*, IEEE J. Sel. Top. Signal Process., 10 (2016), pp. 284–295, https://doi.org/10.1109/JSTSP.2015.2503260.

[62] N. Vervliet, O. Debals, and L. D. Lathauwer, *Tensorlab 3.0: Numerical optimization strategies for large-scale constrained and coupled matrix/tensor factorization*, in 2016 50th Asilomar Conference on Signals, Systems and Computers, Nov 2016, pp. 1733–1738, https://doi.org/10.1109/ACSSC.2016.7869679.

[63] Y. Wang, R. Chen, J. Ghosh, J. C. Denny, A. Kho, Y. Chen, B. A. Malin, and J. Sun, *Rubik: Knowledge guided tensor factorization and completion for health data analytics*, in Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD '15, New York, NY, USA, 2015, ACM, pp. 1265–1274, https://doi.org/10.1145/2783258.2783395, http://doi.acm.org/10.1145/2783258.2783395.

[64] Y. Wang, H.-Y. Tung, A. J. Smola, and A. Anandkumar, *Fast and guaranteed tensor decomposition via sketching*, in Advances in Neural Information Processing Systems (NIPS) 28, 2015, pp. 991–999, http://papers.nips.cc/paper/5944-fast-and-guaranteed-tensor-decomposition-via-sketching.pdf.

[65] S. R. White, *Density matrix formulation for quantum renormalization groups*, Phys. Rev. Lett., 69 (1992), pp. 2863–2866, https://doi.org/10.1103/PhysRevLett.69.2863, https://link.aps.org/doi/10.1103/PhysRevLett.69.2863.

[66] D. P. Woodruff, *Sketching as a tool for numerical linear algebra*, Found. Trends Theor. Comput. Sci., 10 (2014), pp. 1–157, https://doi.org/10.1561/0400000060.

[67] R. Yu, S. Zheng, A. Anandkumar, and Y. Yue, *Long-term forecasting using tensor-train rnns*, CoRR, abs/1711.00073 (2017), http://arxiv.org/abs/1711.00073, https://arxiv.org/abs/1711.00073.

[68] G. Zhou, A. Cichocki, and S. Xie, *Decomposition of big tensors with low multilinear rank*, 2014, https://arxiv.org/abs/1412.1885.