A THIRD-ORDER GENERALIZATION OF THE MATRIX SVD AS A PRODUCT OF THIRD-ORDER TENSORS *

MISHA E. KILMER†, CARLA D. MARTIN‡, AND LISA PERRONE§

Abstract. Traditionally, extending the Singular Value Decomposition (SVD) to third-order tensors (multiway arrays) has involved a representation using the outer product of vectors. These outer products can be written in terms of the n-mode product, which can also be used to describe a type of multiplication between two tensors. In this paper, we present a different type of third-order generalization of the SVD where an order-3 tensor is instead decomposed as a product of order-3 tensors. In order to define this new notion, we define tensor-tensor multiplication in such a way so that it is closed under this operation. This results in new definitions for tensors such as the tensor transpose, inverse, and identity. These definitions have the advantage they can be extended, though in a non-trivial way, to the order-p ($p > 3$) case [31]. A major motivation for considering this new type of tensor multiplication is to devise new types of factorizations for tensors which could then be used in applications such as data compression. We therefore present two strategies for compressing third-order tensors which make use of our new SVD generalization and give some numerical comparisons to existing algorithms on synthetic data.

Key words. multilinear algebra, tensor decomposition, singular value decomposition, multidimensional arrays

AMS subject classifications. 15A69, 65F30

1. Introduction.

The Singular Value Decomposition (SVD) of a matrix gives us important information about a matrix such as its rank, an orthonormal basis for the column or row space, and reduction to diagonal form. In applications, especially those involving multiway data analysis, information about the rank and reduction of tensors to have fewer nonzero entries are useful concepts to try to extend to higher dimensions. However, many of the powerful tools of linear algebra such as the SVD do not, unfortunately, extend in a straight-forward way to tensors of order three or higher. There have been several such extensions of the matrix SVD to tensors in the literature, many of which are used in a variety of applications such as chemometrics [39], psychometrics [27], signal processing [10, 38, 8], computer vision [42, 43, 44], data mining [37, 2], graph analysis [25], neuroscience [6, 33, 34], and more. The models used most often in these areas include the CANDECOMP/PARAFAC (CP) model [7, 17] and the TUCKER model [41] or Higher-Order SVD (HOSVD) [11]. A thorough treatment of these models, other SVD extensions to tensors, special cases, applications, and additional references can be found in [26].

In this paper, we present a new way of extending the matrix SVD to tensors. Specifically, we define a new type of tensor multiplication that allows a third-order tensor to be written as a product of third-order tensors (as opposed to a linear combination of outer product of vectors). This new decomposition is analogous to the SVD in the matrix case (i.e. the case when the third dimension of the tensor is one). Although it is possible to extend our method to higher-order tensors through recursion, we do not discuss that work here, as it is beyond the scope of the present work.

*This work was supported by NSF grant DMS-0552577 and by a Tufts University Summer Faculty Research Award.

†Mathematics, Tufts University, 113 Bromfield-Pearson Bldg., Medford, MA 02155, misha.kilmer@tufts.edu,

‡Mathematics and Statistics, James Madison University, 112 Roop Hall, MSC 1911, Harrisonburg, VA 22807, carlam@math.jmu.edu

§Mathematics, Hawaii Pacific University, lcperrone@gmail.com

We use the accepted notation where an order-p tensor is indexed by p indices and can be represented as a multidimensional array of data [22]. That is, an order-p tensor, $\mathcal{A}$, can be written as

$$\mathcal{A} = (a_{i_1 i_2 \ldots i_p}) \in \mathbb{R}^{n_1 \times n_2 \times \cdots \times n_p}.$$

Thus, a matrix is considered a second-order tensor, and a vector is a first-order tensor. A third-order tensor can be pictured as a "cube" of data (see Figure 1.1). While the orientation of higher-order tensors is not unique, it is convenient to refer to the "faces" of the tensor as the tensor formed by holding the last index constant. For example, if $\mathcal{A} = (a_{i_1 i_2 i_3})$ is a third-order tensor, the k-th face is denoted using the MATLAB notation $\mathcal{A}(:, :, k)$.

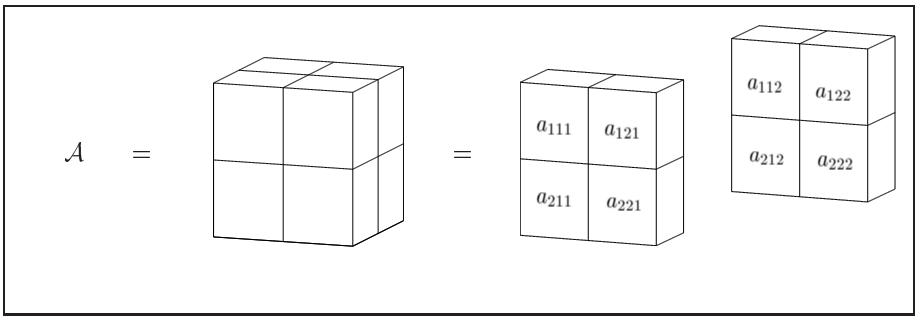


FIG. 1.1. *Illustration of a* $2 \times 2 \times 2$ *tensor as a cube of data*

The entries of a tensor $\mathcal{A}$ can also be rearranged to correspond to viewing the multiway array in different ways (see Figure 1.2 for an illustration of a $2 \times 2 \times 2$ tensor). "Flattening" the different orientations shown in Figure 1.2 correspond to *unfolding matrices* or *matricization* described in [26].

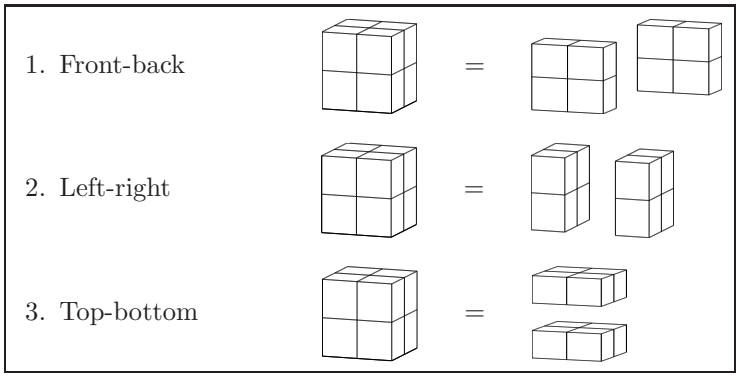


FIG. 1.2. *Three ways to cut a cube of data for a third-order tensor*

In order to define our new tensor factorization, we first define a notion of tensor-tensor multiplication that is closed under multiplication, and address the issue of invertibility. Similar to matrices, we formulate a definition such that the product of two $n \times n \times n$ tensors yields an $n \times n \times n$ tensor. We can show the operation

is associative, identify the notion of an identity operator and give properties for an inverse. It follows that the set of all such invertible $n \times n \times n$ tensors under this operation forms a group. We expand our list of definitions to include the concepts of transpose and orthogonality.

Our presentation is organized as follows. In Section 2, we describe the existing outer product representation that is traditionally used to extend the matrix SVD and briefly describe some of its properties. We use Section 3 to introduce some notation that will allow us to easily describe our new tensor operations. In Section 4 we define a new notion of tensor-tensor multiplication with the properties noted above. We also provide definitions of a tensor inverse, transpose, and orthogonality, and prove some properties based on these definitions. In Section 5, we use our new definitions to derive a tensor factorization that is a product of third-order tensors and show that it has some desirable features. We present some possible tensor compression strategies in 6, and we give a qualitative assessment and a quantitative comparison between the most promising of the two strategies and existing compression methods in 7. Finally, in Section 8, we discuss how our new definitions can also lead to the extension of other well-known matrix factorizations, such as the QR factorization, and give concluding remarks in 9.

2. Tensor Background. There are multiple ways to extend the matrix SVD to higher dimensions. In this section, we describe the most widely used model that involves an outer product representation.

In two dimensions, if the SVD of a matrix, $A \in \mathbb{R}^{n_1 \times n_2}$, is given by $A = U\Sigma V^T$, then A can be written as

$$A = \sum_{i=1}^{r} \sigma_i (u^{(i)} \circ v^{(i)}), \tag{2.1}$$

where $u^{(i)}$ and $v^{(i)}$ are the i-th columns of orthogonal matrices $U \in \mathbb{R}^{n_1 \times n_1}$, $V \in \mathbb{R}^{n_2 \times n_2}$, respectively, σ_i is the i-th diagonal of the diagonal matrix, Σ, r is the rank of A, and "$\circ$" denotes the outer product. Since $u^{(i)} \circ v^{(i)}$ is a rank-1 matrix, the SVD can be thought of as decomposing a matrix as a sum of rank-1 matrices. Extending the SVD to three-dimensions, therefore, involves a sum of rank-1, third-order tensors. There are two widely used SVD extensions of this type. We briefly describe each below for the third order case.

2.1. The CP model. If $\mathcal{A} \in \mathbb{R}^{n_1 \times n_2 \times n_3}$ is a third-order tensor, the goal is to find matrices $U \in \mathbb{R}^{n_1 \times n_1}$, $V \in \mathbb{R}^{n_2 \times n_2}$, and $W \in \mathbb{R}^{n_3 \times n_3}$ so that

$$\mathcal{A} = \sum_{i=1}^{r} \sigma_i (u^{(i)} \circ v^{(i)} \circ w^{(i)}), \tag{2.2}$$

and σ_i is the i-th super-diagonal from a diagonal tensor $\Sigma \in \mathbb{R}^{n_1 \times n_2 \times n_3}$. The vectors $u^{(i)}$, $v^{(i)}$, and $w^{(i)}$ are the i-th columns from matrices U, V, W, respectively. The outer product

$$u^{(i)} \circ v^{(i)} \circ w^{(i)}$$

is known as a rank-1 tensor. Ideally, r in (2.2) is minimal. The minimum such r so that (2.2) holds is known as the *tensor rank*.

A decomposition of the form (2.2) is called a CANDECOMP-PARAFAC (CP) decomposition (CANonical DECOMPosition or PARAllel FACtors model), and was

independently proposed in [7] and [17]. The CP decomposition is used widely in psychometrics [27] and chemometrics [39], where the data follow a tri-linear model. We emphasize several differences between the matrix SVD and (2.2).

1. The matrices U, V, W in (2.2) are not constrained to be orthogonal. Furthermore, an orthogonal decomposition of this form may not exist for higher-order tensors [15].
2. There is no known closed-form solution to determine the rank of a tensor *a priori*. Rank determination in practice is a trial-and-error approach based on known information about the data [26].
3. The best rank-K ($K < r$) approximation to a tensor may not always exist [14], whereas the best rank-K approximation to a matrix is given directly by the SVD [16, p.72].

Rank determination of a tensor is a widely-studied problem [26]. It is known that the maximum possible rank of a tensor is not given directly from the dimensions, as is the case with matrices (the maximum possible rank of an $n_1 \times n_2$ matrix is $\min(n_1, n_2)$). However, loose upper bounds do exist for higher-order tensors. Specifically, the maximum possible rank of an $n_1 \times n_2 \times n_3$ tensor is bounded by $\min(n_1 n_2, n_1 n_3, n_2 n_3)$ [28]. In the special case of $n \times n \times 2$ tensors, it has been shown that the maximum possible rank is $\lfloor 3n/2 \rfloor$ (see [19, 28, 30, 40]).

Despite these differences, (2.2), is very important in explaining interactions in multi-way data. This model also extends in a straight-forward way to order-p tensors.

2.2. The TUCKER3 model and the HOSVD. While some applications use the nonorthogonal decomposition (2.2), other applications need orthogonality of the matrices for better interpretation of the data [35, 37, 42, 43, 44]. Therefore, a more general form is often used to guarantee existence of an orthogonal decomposition as well as to better model certain data. If $\mathcal{A}$ is a third order, $n_1 \times n_2 \times n_3$ tensor, its TUCKER3 decomposition [41] has the form

$$\mathcal{A} = \sum_{i=1}^{n_1} \sum_{j=1}^{n_2} \sum_{k=1}^{n_3} \sigma_{ijk} (u^{(i)} \circ v^{(j)} \circ w^{(k)}), \tag{2.3}$$

where $u^{(i)} \in \mathbb{R}^{n_1}$, $v^{(i)} \in \mathbb{R}^{n_2}$, and $w^{(i)} \in \mathbb{R}^{n_3}$. Orthogonality constraints are not required in the general TUCKER3 decomposition. The tensor $\Sigma = (\sigma_{ijk})$ is called the *core tensor*. In general, the core tensor Σ is dense and the decomposition (2.3) does not reveal the rank. In the special case where Σ is *diagonal* (i.e., $\sigma_{ijk} = 0$ unless $i = j = k$) then (2.3) reduces to (2.2).

If $u^{(i)}, v^{(j)}, w^{(k)}$ are columns of the orthogonal matrices U, V, W (i.e. orthgonality constraints are added to 2.3), then (2.3) is referred to as the Higher-Order Singular Value Decomposition (HOSVD), [11]. Just as in the general TUCKER3 decomposition, it is usually the case that the core tensor is dense and not diagonal.

The HOSVD can be computed directly by computing the SVD of three matrices which are obtained by flattening the tensor in various ways, using the results to assemble the core. However, an alternating least squares (ALS) approach is most often used for the general TUCKER3 model [26]. The state-of-the-art has been implemented in a MATLAB toolbox [3, 4] that allows the user to specify orthogonality constraints as well as the amount of compression desired, as specified below.

By choosing $k_1 < n_1$, $k_2 < n_2$, $k_3 < n_3$, one obtains the approximate TUCKER3

factorization

$$\mathcal{A} \approx \sum_{i=1}^{k_1} \sum_{j=1}^{k_2} \sum_{k=1}^{k_3} \sigma_{ijk}(u^{(i)} \circ v^{(j)} \circ w^{(k)}) \equiv \mathcal{A}^{(k_1,k_2,k_3)}. \tag{2.4}$$

When we specify orthogonality constraints, we refer to the resulting factorization as the truncated-HOSVD (THOSVD). There are two ways to compute a THOSVD, and they do not necessarily give the same factorization. One is to compute the HOSVD, which is a direct factorization method, and truncate terms, keeping the terms for which the core entries are largest in magnitude. This approach was taken in [35] and we refer to it later as THOSVD-1. Another is to determine the solution, using an alternating least squares algorithm [26], to

$$\min_{\substack{\sigma_{ijk},\, u_i, v_j, w_k; \\ i=1,\ldots,k_1, j=1,\ldots,k_2, k=1,\ldots,k_3}} \|\mathcal{A} - \mathcal{A}^{(k_1,k_2,k_3)}\|_F, \tag{2.5}$$

subject to orthogonality constraints. We refer to this method later as THOSVD-2.

Other types of compressed representations of the TUCKER3 model have been used. These include, for example, maximum variance of squares [18], maximum sums of squares of the diagonals of each face of a tensor [27]., and maximum sums of squares of the diagonals of a third-order tensor [21, 13, 32]. A greedy algorithm to compute an orthogonal tensor decomposition has been proposed by [24]. Algorithms have also been developed to compute the nearest rank-1 tensor to a given tensor (see [12, 23, 24, 46]).

In Section 7 we compare our compression algorithm with THOSVD-2. We give an approximate flop comparison among the factorization and approximate factorization strategies later in Table 6.3.

2.3. Tensor multiplication using the contracted product. Multiplication between tensors and matrices has been defined using the n-mode product [5, 11, 22]. While we do not go into detail here, the n-mode product can also be used to describe a type of multiplication between two tensors. While there are multiple ways to multiply tensors, the most common method is the *contracted product*. The name "contracted product" can be a little misleading: indeed, the contracted product of an $\ell \times n_2 \times n_3$ tensor and an $\ell \times m_2 \times m_3$ tensor in the first mode is an $n_2 \times n_3 \times m_2 \times m_3$ tensor. However, the contracted product of an $\ell_1 \times \ell_2 \times n$ with an $\ell_1 \times \ell_2 \times m$ tensor in the first two modes, results in a $n \times m$ tensor (matrix). In summary, the order of the resulting tensor depends on the modes where the multiplication takes place. We refer the reader to the explanation in [5] for details. We emphasize that the contracted product of two order-p tensors does not necessarily result in an order-p tensor, and hence does not preserve order.

Unfortunately, since the dimensions are not preserved under the contracted product, it implies that the set of all third-order tensors is not closed under this type of operation. Furthermore, this type of operation does not allow us to specify a notion of inverse, since the operation is not invertible.

3. Notation. We use circulant matrices extensively in our new definitions. Recall that if

$$v = \begin{bmatrix} v_0 & v_1 & v_2 & v_3 \end{bmatrix}^T$$

then

$$\text{circ}(v) = \begin{bmatrix} v_0 & v_3 & v_2 & v_1 \\ v_1 & v_0 & v_3 & v_2 \\ v_2 & v_1 & v_0 & v_3 \\ v_3 & v_2 & v_1 & v_0 \end{bmatrix}$$

is a circulant matrix. Note that all the matrix entries are defined once the first column is specified. Therefore, we adopt the convention that $\text{circ}(v)$ refers to the circulant matrix obtained with the vector v as the first column.

Circulant matrices can be diagonalized with the normalized Discrete Fourier Transform (DFT) matrix [16, p.202], which is unitary. In particular, if v is $n \times 1$ and F_n is the $n \times n$ DFT matrix, then

$$F_n \ \text{circ}(v) F_n^{-1}$$

is diagonal. The following, well-known, simple fact ([1]) is used to compute this diagonal using the fast Fourier transform (FFT):

Fact 1 The diagonal of $F_n \ \text{circ}(v) F_n^{-1} = \texttt{fft}(v)$, where $\texttt{fft}(v)$ denotes the Fast Fourier Transform of v.

We also make use of the `fold` and `unfold` operators. Suppose that $\mathcal{A} \in \mathbb{R}^{n_1 \times n_2 \times n_3}$, and that each frontal face of the tensor is defined by $n_1 \times n_2$ matrices $\mathcal{A}(:,:,1)$, ..., $\mathcal{A}(:,:,n_3)$. Then `unfold` is defined by

$$\texttt{unfold}(\mathcal{A}, 1) = \begin{bmatrix} \mathcal{A}(:,:,1) \\ \mathcal{A}(:,:,2) \\ \vdots \\ \mathcal{A}(:,:,n_3) \end{bmatrix} \in \mathbb{R}^{n_1 n_3 \times n_2}.$$

The second argument of `unfold` specifies which orientation of the tensor to unfold (see Figure 1.2). For example, $\texttt{unfold}(\mathcal{A}, 1)$ unstacks the tensor according to its front-back faces. Similarly, $\texttt{unfold}(\mathcal{A}, 2)$ refers to unstacking by faces defined by slicing the tensor side to side and $\texttt{unfold}(\mathcal{A}, 3)$ unstacks by slicing top to bottom. See Figure 3.1 for an example.

An optional third index argument can be made to indicate unfolding in a different ordering. For example, for a $3 \times 3 \times 3$ tensor, $\texttt{unfold}(\mathcal{A}, 1, [1,3,2])$ would stack the first face first, then the last face, followed by the middle face. The operation that folds back into a tensor in the same ordering is given by $\texttt{fold}(\mathcal{A}, i)$, $i = 1, 2,$ or 3. We point out that [11] defines a *matrix unfolding* of a tensor in a slightly different way than `unfold`. We refer the reader to [11] for details. We use `unfold` so that our new definition of tensor multiplication, defined in the next section, is easily explained.

It is possible to create a block-circulant matrix from the slices of a tensor. For example, if $\mathcal{A} \in \mathbb{R}^{n_1 \times n_2 \times n_3}$ with $n_1 \times n_2$ frontal faces $A_1 = \mathcal{A}(:,:,1)$, ..., $A_{n_3} = \mathcal{A}(:,:,n_3)$ then

$$\text{circ}(\ \texttt{unfold}(\mathcal{A}, 1)) = \begin{bmatrix} A_1 & A_{n_3} & A_{n_3-1} & \dots & A_2 \\ A_2 & A_1 & A_{n_3} & \dots & A_3 \\ \vdots & \ddots & \ddots & \ddots & \vdots \\ A_{n_3} & A_{n_3-1} & \ddots & A_2 & A_1 \end{bmatrix}.$$

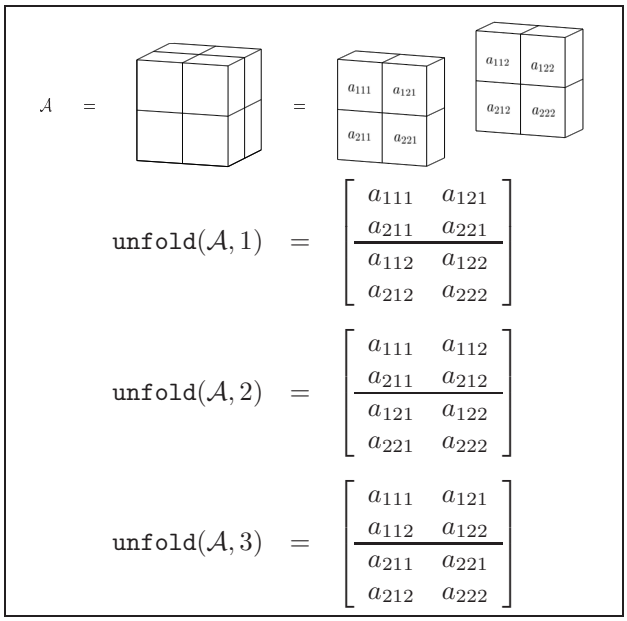


FIG. 3.1. *The* `unfold` *operator applied to a* $2 \times 2 \times 2$ *tensor*

Just as circulant matrices can be diagonalized by the DFT, block-circulant matrices can be block-diagonalized. Suppose $\mathcal{A}$ is $n_1 \times n_2 \times n_3$ and F_{n_3} is the $n_3 \times n_3$ DFT matrix. Then

$$(F_{n_3} \otimes I_{n_1}) \cdot \text{ circ}(\texttt{unfold}(\mathcal{A},1)) \cdot (F_{n_3}^* \otimes I_{n_2}) = \begin{bmatrix} D_1 & & & \\ & D_2 & & \\ & & \ddots & \\ & & & D_{n_3} \end{bmatrix}$$

where "$\otimes$" denotes the Kronecker product and F^* denotes the conjugate transpose of F and "$\cdot$" means standard matrix product. Note that each D_i could be dense and furthermore most will be complex unless certain symmetry conditions hold.

To compute the product in the preceding paragraph, assuming n_3 is a power of 2, can be done in $O(n_1 n_2 n_3 log_2(n_3))$ flops using the FFT and Fact 1. Indeed, using stride permutations and Fact 1, it is straightforward to show that there is no need to lay out the data in order to compute the matrices D_i, and we arrive at the following fact:

Fact 2 The D_i are the faces of the tensor $\mathcal{D}$, where $\mathcal{D}$ is computed by applying FFT's along each "fiber" of $\mathcal{A}$ and a fiber is a column vector obtained by holding i, j fixed and taking $\mathcal{A}(i, j, :)$.

4. New Tensor Definitions. We now extend several concepts from linear algebra to tensors. In particular, we define tensor multiplication, tensor transpose, a tensor identity, and tensor inverse. We use these definitions to define orthogonal tensors. Note that our definition of tensor multiplication preserves size. While we present our definitions and algorithms with third-order tensors, all concepts in this paper can be extended for general order-p tensors in a recursive manner [31]. Furthermore, all definitions collapse to the standard linear algebra definitions when the

tensor is order-2.

DEFINITION 4.1. *Let $\mathcal{A}$ be $n_1 \times n_2 \times n_3$ and $\mathcal{B}$ be $n_2 \times \ell \times n_3$. Then the product $\mathcal{A} * \mathcal{B}$ is the $n_1 \times \ell \times n_3$ tensor*

$$\mathcal{A} * \mathcal{B} = \texttt{fold}\left(\text{circ}(\texttt{unfold}(\mathcal{A}, 1)) \cdot \texttt{unfold}(\mathcal{B}, 1), 1\right).$$

EXAMPLE 4.2. *Suppose $\mathcal{A} \in \mathbb{R}^{n_1 \times n_2 \times 3}$ and $\mathcal{B} \in \mathbb{R}^{n_2 \times \ell \times 3}$. Then*

$$\mathcal{A} * \mathcal{B} = \texttt{fold}\left(\begin{bmatrix} A_1 & A_3 & A_2 \\ A_2 & A_1 & A_3 \\ A_3 & A_2 & A_1 \end{bmatrix} \begin{bmatrix} B_1 \\ B_2 \\ B_3 \end{bmatrix}, 1\right) \in \mathbb{R}^{n_1 \times \ell \times 3}.$$

Tensors and matrices can be multiplied in various modes using the contracted product, provided the dimensions agree. For example, when a tensor is multiplied by a matrix in say, mode 1, it means that each face of the tensor is multiplied by that matrix. However, notice there is no interaction among the resulting faces of the new tensor. Our motivation in using Definition 4.1 was to ensure that all possible pairs of faces (i.e. slices in mode 1) from the two tensors being multiplied were computed, but that there was also an interaction among those results in the third dimension as well. So although the result of the multiplication will depend on the orientation of the two tensors involved, it is inherently more "three-dimensional" in nature than other options described in the literature. Further, as we show momentarily, this definition possesses many desirable properties as well.

Before we describe the theoretical properties of this product, we mention a word about the computation of this product. If the tensors are sparse, we may choose to compute this product as it is written. If the tensors are dense, naively computing this product would cost $O(\ell n_1 n_2 n_3^2)$ flops. However, since circ($\texttt{unfold}(\mathcal{A}, 1)$) can be block diagonalized, we can choose to compute this product as

$$(F_{n_3}^* \otimes I_{n_1}) \left((F_{n_3} \otimes I_{n_1}) \cdot \text{ circ}(\texttt{unfold}(\mathcal{A}, 1)) \cdot (F_{n_3}^* \otimes I_{n_2})\right) (F_{n_3} \otimes I_{n_2}) \texttt{unfold}(\mathcal{B}, 1), 1).$$

It is readily shown that $(F_{n_3} \otimes I_{n_2})$ $\texttt{unfold}(\mathcal{B}, 1)$ can be computed in $O(\ell n_2 n_3 \log_2(n_3))$ flops by applying FFTs along the fibers of $\mathcal{B}$: we call the result $\tilde{\mathcal{B}}$. If we take the FFT of each fiber of $\mathcal{A}$, using Fact 2, we obtain $\mathcal{D}$. Thus, it remains to multiply each face of $\mathcal{D}$ with each face of $\tilde{\mathcal{B}}$, then take an inverse FFT along the fibers of the result. We arrive at the following fact regarding this multiplication.

Fact 3 The product in Definition 4.1 can be computed in at most $O(n_1 n_2 \ell n_3)$ flops by making use of the FFT along mode 3.

If n_3 is not a power of two, we may still employ FFTs in the multiplication by noting that the block circulant matrix can be embedded in a larger block circulant matrix where the number of blocks in a block row can be increased to the next largest power of two greater than $2n_3 - 1$ by the addition of zero blocks and repetition of previous blocks in an appropriate fashion. Likewise, once $\mathcal{B}$ is unfolded, it can be conformally extended by zero blocks. The product is computed using FFTs, and the result is then truncated appropriately. This is a commonly used trick in the literature for fast multiplication with Toeplitz or block Toeplitz matrices, and will not be described further here.

Now we discuss some of the desirable theoretical properties of this definition of tensor multiplication. First, tensor multiplication as we have defined it, is associative, as the next lemma shows.

LEMMA 4.3. $\mathcal{A} * (\mathcal{B} * \mathcal{C}) = (\mathcal{A} * \mathcal{B}) * \mathcal{C}$.

Proof. The proof follows naturally from the definition of $*$ and the fact that matrix-matrix multiplication is associative. □

We also define the concept of an identity tensor.

DEFINITION 4.4. *The $n \times n \times \ell$ identity tensor $\mathcal{I}_{nn\ell}$ is the tensor whose front face is the $n \times n$ identity matrix, and whose other faces are all zeros. That is,*

$$\texttt{unfold}(\mathcal{I}_{nn\ell}, 1) = \begin{bmatrix} I_n \\ \mathbf{0} \\ \vdots \\ \mathbf{0} \end{bmatrix}.$$

We can now define the tensor inverse.

DEFINITION 4.5. *An $n \times n \times n$ tensor $\mathcal{A}$ has an inverse $\mathcal{B}$ provided that*

$$\mathcal{A} * \mathcal{B} = \mathcal{I}, \qquad \text{and} \qquad \mathcal{B} * \mathcal{A} = \mathcal{I}.$$

From Definitions 4.1, 4.4, 4.5 and Lemma 4.3, we have the following lemma.

LEMMA 4.6. *The set of all invertible $n \times n \times n$ tensors form a group under the $*$ operation.*

It is also true that the set of invertible $n \times n \times n$ tensors forms a ring under standard tensor addition (component-wise addition) and $*$. The motivation for developing an operation that is closed under multiplication was to develop a new way of representing tensor factorizations reminiscent of matrix factorizations.

Next, it is convenient to give the notion of the transpose operation for tensors

DEFINITION 4.7. *If $\mathcal{A}$ is $n_1 \times n_2 \times n_3$, then $\mathcal{A}^T$ is the $n_2 \times n_1 \times n_3$ tensor obtained by transposing each of the front-back faces and then reversing the order of transposed faces 2 through n_3. In other words*

$$\mathcal{A}^T = \texttt{fold}(\ \texttt{unfold}(\mathcal{A}, 1, [1, n_3 : -1 : 2]), 1).$$

EXAMPLE 4.8. *If $\mathcal{A} \in \mathbb{R}^{n_1 \times n_2 \times 4}$ and its frontal faces are given by the $n_1 \times n_2$ matrices A_1, A_2, A_3, A_4, then*

$$\mathcal{A}^T = \texttt{fold}\left(\begin{bmatrix} A_1^T \\ A_4^T \\ A_3^T \\ A_2^T \end{bmatrix}, 1 \right).$$

The tensor transpose has the same property as the matrix transpose.

LEMMA 4.9. *Suppose $\mathcal{A}, \mathcal{B}$ are two tensors such that $\mathcal{A} * \mathcal{B}$ and $\mathcal{B}^T * \mathcal{A}^T$ is defined. Then $(\mathcal{A} * \mathcal{B})^T = \mathcal{B}^T * \mathcal{A}^T$.*

Proof. Follows directly from Definitions 4.1 and 4.7 □

Next we define permutation tensors.

DEFINITION 4.10. *A permutation tensor is an $n \times n \times \ell$ tensor $\mathcal{P} = (p_{ijk})$ with exactly n entries of unity, such that if $p_{ijk} = 1$, it is the only non-zero entry in row i, column j, and slice k.*

EXAMPLE 4.11. *A permutation tensor* $\mathcal{P} \in \mathbb{R}^{3\times 3\times 2}$*:*

$$\mathcal{P} = \texttt{fold}\left(\left\{\begin{bmatrix}1 & 0 & 0\\ 0 & 0 & 0\\ 0 & 0 & 1\end{bmatrix}, \begin{bmatrix}0 & 0 & 0\\ 0 & 1 & 0\\ 0 & 0 & 0\end{bmatrix}\right\}, 1\right)$$

Before we define the notion of orthogonality for tensors, we present the definition of the Frobenius norm of a tensor, previously defined in [11].

DEFINITION 4.12. *Suppose* $\mathcal{A} = (a_{ijk})$ *is size* $n_1 \times n_2 \times n_3$*. Then*

$$\|\mathcal{A}\|_F = \sqrt{\sum_{i=1}^{n_1}\sum_{j=1}^{n_2}\sum_{k=1}^{n_3} a_{ijk}^2}.$$

We are now ready to define orthogonality for tensors.

DEFINITION 4.13. *An* $n \times n \times \ell$ *real-valued tensor* $\mathcal{Q}$ *is orthogonal if* $\mathcal{Q}^T * \mathcal{Q} = \mathcal{Q} * \mathcal{Q}^T = \mathcal{I}$*.*

It follows that the identity tensor and permutation tensors are orthogonal. Note that if $\mathcal{Q}$ is an orthogonal tensor, then it does not follow that each face of $\mathcal{Q}$ is necessarily orthogonal. Another nice feature of orthogonal tensors is that they preserve the Frobenius norm:

LEMMA 4.14. *If* $\mathcal{Q}$ *is an orthogonal tensor,*

$$\|\mathcal{Q} * \mathcal{A}\|_F = \|\mathcal{A}\|_F.$$

Proof. From definitions 4.1, 4.7, and 4.12, it follows that

$$\|\mathcal{A}\|_F^2 = \text{trace}((\mathcal{A} * \mathcal{A}^T)_{(:,:,1)}) = \text{trace}((\mathcal{A}^T * \mathcal{A})_{(:,:,1)}),$$

where $(\mathcal{A} * \mathcal{A}^T)_{(:,:,1)}$ is the front face of $\mathcal{A} * \mathcal{A}^T$ and $(\mathcal{A}^T * \mathcal{A})_{(:,:,1)}$ is the front face of $\mathcal{A}^T * \mathcal{A}$. Therefore,

$$\begin{aligned}\|\mathcal{Q} * \mathcal{A}\|_F^2 &= \text{trace}([(\mathcal{Q} * \mathcal{A})^T * (\mathcal{Q} * \mathcal{A})]_{(:,:,1)})\\ &= \text{trace}([\mathcal{A}^T * \mathcal{Q}^T * \mathcal{Q} * \mathcal{A}]_{(:,:,1)})\\ &= \|\mathcal{A}\|_F^2.\end{aligned}$$

□

We can also define a notion of partial orthogonality, similar to saying that a tall, thin matrix has orthogonal columns. In this case if $\mathcal{Q}$ is $p \times q \times n$ and **partially orthogonal**, we mean $\mathcal{Q}^T * \mathcal{Q}$ is well defined and equal to the a $q \times q \times n$ identity.

If the tensor is two-dimensional (i.e. $n_3 = 1$, so the tensor is a matrix), Definitions 4.1, 4.4, 4.5, 4.10, 4.12, and 4.13 are consistent with standard matrix algebra operations and terminology.

5. New Higher-order Extension of the SVD. We say a tensor is "f-diagonal" if each front-back face is diagonal. Likewise, a tensor is f-upper triangular or f-lower triangular if each front-back face is upper or lower triangular, respectively.

THEOREM 5.1. *Let* $\mathcal{A}$ *be an* $n_1 \times n_2 \times n_3$ *real-valued tensor. Then* $\mathcal{A}$ *can be factored as*

$$\mathcal{A} = \mathcal{U} * \mathcal{S} * \mathcal{V}^T, \tag{5.1}$$

where $\mathcal{U}, \mathcal{V}$ are orthogonal $n_1 \times n_1 \times n_3$ and $n_2 \times n_2 \times n_3$ respectively, and $\mathcal{S}$ is a $n_1 \times n_2 \times n_3$ f-diagonal tensor.

Proof. The proof is by construction. Recall that

$$(F_{n_3} \otimes I_{n_1}) \cdot \text{ circ}(\texttt{ unfold}(\mathcal{A}, 1)) \cdot (F_{n_3}^* \otimes I_{n_2}) = \begin{bmatrix} D_1 & & & \\ & D_2 & & \\ & & \ddots & \\ & & & D_{n_3} \end{bmatrix}$$

Next, we compute the SVD of each D_i as $D_i = U_i \Sigma_i V_i^T$. Then

$$\begin{bmatrix} D_1 & & \\ & \ddots & \\ & & D_{n_3} \end{bmatrix} = \begin{bmatrix} U_1 & & \\ & \ddots & \\ & & U_{n_3} \end{bmatrix} \begin{bmatrix} \Sigma_1 & & \\ & \ddots & \\ & & \Sigma_{n_3} \end{bmatrix} \begin{bmatrix} V_1^T & & \\ & \ddots & \\ & & V_{n_3}^T \end{bmatrix}. \quad (5.2)$$

Since

$$(F_{n_3}^* \otimes I_{n_1}) \begin{bmatrix} U_1 & & \\ & \ddots & \\ & & U_{n_3} \end{bmatrix} (F_{n_3} \otimes I_{n_1}),$$

$$(F_{n_3}^* \otimes I_{n_1}) \begin{bmatrix} \Sigma_1 & & \\ & \ddots & \\ & & \Sigma_{n_3} \end{bmatrix} (F_{n_3} \otimes I_{n_2}),$$

and

$$(F_{n_3}^* \otimes I_{n_2}) \begin{bmatrix} V_1^T & & \\ & \ddots & \\ & & V_{n_3}^T \end{bmatrix} (F_{n_3} \otimes I_{n_2}),$$

are circulant matrices, we can obtain an expression for $\texttt{unfold}(\mathcal{A}, 1)$, by applying the appropriate matrix $(F_{n_3}^* \otimes I)$ to the left and the appropriate matrix $(F_{n_3} \otimes I)$ to the right of each of the matrices in (5.2), and folding up the result. This gives a decomposition of the form $\mathcal{U} * \mathcal{S} * \mathcal{V}^T$.

It remains to show that $\mathcal{U}$ and $\mathcal{V}$ are orthogonal. However, this is easily proved by forming the necessary products (e.g. $\mathcal{U}^T * \mathcal{U}$) and using the same forward, backward matrix transformation to the Fourier domain as was used to compute the factorization, and the proof is complete. □

This decomposition can be computed using the fast Fourier transform utilizing Fact 1 from above. One version of Matlab psuedocode to compute this decomposition is provided below.

Algorithm T-SVD
Input: $n_1 \times n_2 \times n_3$ tensor $\mathcal{A}$
$\mathcal{D} = \texttt{fft}(\mathcal{A},[\,],3)$;
for $i = 1 \ldots n_3$
$[u, s, v] = \texttt{svd}(\mathcal{D}(:,:,\text{i}))$;
$\mathcal{U}(:,:,i) = u$; $\mathcal{V}(:,:,i) = v$; $\mathcal{S}(:,:,i) = s$
$\mathcal{U} = \texttt{ifft}(\mathcal{U},[\,],3)$; $\mathcal{V} = \texttt{ifft}(\mathcal{V},[\,],3)$;$\mathcal{S} = \texttt{ifft}(\mathcal{S},[\,],3)$;

The number of flops to compute this tensor SVD is given in Table 6.3 and compared to the cost of computing a full HOSVD, under the assumption that $n_1 = n_2 = n_3 = n$. The dominant cost for our algorithm on an $n \times n \times n$ cube is the n SVD's of the D_i, so the tensor SVD computation costs $O(n^4)$ flops, although the step involving the FFT along each fiber is reduced when n is a power of 2. Note that algorithm T-SVD for computing our tensor SVD is a direct algorithm, and is decidedly cheaper to compute than the full HOSVD[1].

Note that if $\mathcal{A}$ is real, the decomposition (5.1) is composed of real tensors even though the proof of Theorem 5.1 involves computations over the complex field. The complex computations result when computing the D_i matrices in (5.2). In particular, these D_i matrices will be complex unless there are very specific symmetry conditions imposed on the original tensor.

Furthermore, by Lemma 4.14 we have

$$\|\mathcal{A}\|_F = \|\mathcal{S}\|_F.$$

Due to the nature of the construction of this factorization, it is also possible to form "reduced" factorizations (where f-diagonal tensor is a cube) by using partially orthogonal tensors.

There are some perhaps surprising relations between the factorization defined this way and a matrix counterpart of $\mathcal{A}$. If $A_1, \ldots, A_{n_3}$ are the front faces of $\mathcal{A}$, we have the following relationship between our tensor SVD and $\sum_{k=1}^{n_3} A_k$.

LEMMA 5.2. *Suppose $\mathcal{A} = \mathcal{U} * \mathcal{S} * \mathcal{V}^T$. Then*

$$\sum_{k=1}^{n_3} A_k = \left(\sum_{k=1}^{n_3} U_k\right)\left(\sum_{k=1}^{n_3} S_k\right)\left(\sum_{k=1}^{n_3} V_k^T\right), \tag{5.3}$$

where U_k, S_k, and V_k^T are the k-th frontal faces of $\mathcal{U}$, $\mathcal{S}$, and $\mathcal{V}^T$, respectively.

Furthermore, (5.3) gives an SVD for $\sum A_k$ in the sense that $\sum U_k, \sum V_k$ are orthogonal and $\sum S_k$ is diagonal).

Proof. Clearly $\sum S_k$ is a diagonal matrix (the entries can be made positive by an appropriate scaling) and $\sum R_k$ is an upper-triangular matrix. Now, all the remains to show is that if $\mathcal{U}$ is an orthogonal tensor, then $\sum U_k$ is an orthogonal matrix.

Suppose $\mathcal{U}$ is orthogonal. Then we have that

$$\mathcal{U} * \mathcal{U}^T = \mathcal{I}_{n_1 n_1 n_3}$$

which means that

$$\sum_{k=1}^{n_3} U_k U_k^T = I_{n_3} \qquad \text{and} \qquad \sum_{i \neq j} U_i U_j^T = \mathbf{0}_{n_3}. \tag{5.4}$$

Equation (5.4) means that

$$\left(\sum_{k=1}^{n_3} U_k\right)\left(\sum_{k=1}^{n_3} U_k\right)^T = I_{n_3}$$

which completes the proof. □

[1]The full HOSVD is computed using a direct algorithm whereas the low rank truncated approximation we refer to as THOSVD-2 is computed using ALS.

6. Tensor Compression Strategies. In this section, we consider two possible strategies for data compression. While the first choice is perhaps the obvious choice, given our new SVD generalization, the second choice is shown to be more practical, for several reasons, and is therefore explored in some level of detail.

6.1. Strategy 1. Suppose that $\mathcal{A} \in \mathbb{R}^{n_1 \times n_2 \times n_3}$. If its T-SVD is given by $\mathcal{A} = \mathcal{U} * \mathcal{S} * \mathcal{V}^T$, then it is easy to show that

$$\mathcal{A} = \sum_{i=1}^{\min(n_1,n_2)} \mathcal{U}(:,i,:) * \mathcal{S}(i,i,:) * \mathcal{V}(:,i,:)^T. \tag{6.1}$$

A straightforward way to compress the tensor is to choose some $k < \min(n_1, n_2)$ and compute

$$\mathcal{A} \approx \sum_{i=1}^{k} \mathcal{U}(:,i,:) * \mathcal{S}(i,i,:) * \mathcal{V}(:,i,:)^T. \tag{6.2}$$

This is justified since

$$||\mathcal{S}(1,1,:)||_F \geq ||\mathcal{S}(2,2,:)||_F \geq \cdots \geq ||\mathcal{S}(\min(n_1,n_2),\min(n_1,n_2),:)||_F.$$

In general (6.2) is not useful in determining tensor rank, since the rank of $\mathcal{A}$ is likely larger than $\min(n_1, n_2)$. Furthermore, (6.2) is limited in that the number of terms used in the approximation is restricted by $\min(n_1, n_2)$ (see discussion below).

6.2. Strategy 2. Suppose that $\mathcal{A} \in \mathbb{R}^{n_1 \times n_2 \times n_3}$ and its T-SVD is given by $\mathcal{A} = \mathcal{U} * \mathcal{S} * \mathcal{V}^T$. The idea in this section is to use Lemma 5.2 for compression. First we sum the faces of each tensor in our decomposition. That is, set

$$A = \sum_{k=1}^{n_3} \mathcal{A}(:,:,k) \tag{6.3}$$

and analogously for $\mathcal{U}$, $\mathcal{S}$, and $\mathcal{V}$:

$$U = \sum_{k=1}^{n_3} \mathcal{U}(:,:,k) \qquad S = \sum_{k=1}^{n_3} \mathcal{S}(:,:,k) \qquad V = \sum_{k=1}^{n_3} \mathcal{V}(:,:,k).$$

From Lemma 5.2 we have that $A = USV^T$. Now choose $k_1 << n_1$ and $k_2 << n_2$ and compute the truncated matrix SVD $\tilde{A} = \tilde{U}\tilde{S}\tilde{V}^T$ where $\tilde{U} = U(:, 1 : k_1)$, $\tilde{S} = S(1 : k_1, 1 : k_2)$, $\tilde{V} = V(:, 1 : k_2)$. To compress $\mathcal{A}$, we can now pre- and post-multiply each face, $\mathcal{A}(:,:,k)$ for $k = 1, \ldots, n_3$ by $\tilde{U}^T$ and $\tilde{V}$. In other words, set $\mathcal{T} \in \mathbb{R}^{k_1 \times k_2 \times n_3}$ as

$$\mathcal{T}(:,:,k) = \tilde{U}^T \mathcal{A}(:,:,k) \tilde{V} \tag{6.4}$$

for $k = 1, \ldots, n_3$. The compressed tensor, $\mathcal{A}_{compressed}$ can be computed with

$$\mathcal{A} \approx \mathcal{A}_{compressed} = \sum_{i=1}^{k_1} \sum_{j=1}^{k_2} \tilde{U}(:,i) \circ \tilde{V}(:,j) \circ \mathcal{T}(i,j,:). \tag{6.5}$$

The interesting feature of Strategy 2 is that although it is based on compressing this new SVD generalization, we do **not** need to run algorithm T-SVD to compute

CP	full HOSVD	THOSVD-2	full tensor SVD	Strat2
$O(n_{it}n^6)$	$O(n^6)$	$O(n_{it}n^6)$	$O(n^4)$	$O(n^3k)$

TABLE 6.1

Comparison of flop count in terms of parameters that define each decomposition for an $n \times n \times n$ tensor. In compression strategy two, we use $k_1 = k_2 = k$ and assume that a full SVD of the $n \times n$ matrix is computed first, although that could be replaced with the computation of a rank-k approximation at this phase. Note that both CP and THOSVD-2 also depend on the number of iterations, n_{it}, required to reach some user-defined convergence tolerance.

full HOSVD	THOSVD-2	full tensor SVD	Strat1	Strat 2
$O(n^3)$	$O(k^2n + 2nk + n^2)$	$O(n^3)$	$O(n^2k)$	$O(k^2n + 2nk)$

TABLE 6.2

Comparison in terms of floating point numbers required to store the final representation, in terms of the parameters that define each decomposition, for an $n_1 \times n_2 \times n_3$ tensor. In compression strategy two, we use $k_1 = k_2 = k$. For the closest comparsion, in the THSOVD-2, we used $k_3 = n$, $k_1 = k_2 = k$, where k is assumed to be known a-priori.

the approximation, provided the truncation parameters have been decided on a priori. Indeed, in light of Lemma 5.2, we can first compute A as the sum across the faces of $\mathcal{A}$, then compute its SVD, form $\tilde{U}$, $\tilde{V}$ by truncating U, V, respectively, and form the core tensor with n_3 triple matrix products as given in (6.4). The algorithm to generate the $\tilde{U}, \tilde{V}$ and $\mathcal{T}$ is given in `Matlab` psuedo-code as follows.

T-SVD Compression 2
Given $n_1 \times n_2 \times n_3$ array $\mathcal{A}$, cutoffs k_1, k_2
$A = \texttt{sum}(\mathcal{A}, 3)$
$[U, S, V] = \texttt{svd}(A, 0)$
$\tilde{U} = U(:, 1 : k_1); \tilde{V} = V(:, 1 : k_2)$
for $k = 1 : n_3$
$\mathcal{T}(:, :, k) = \tilde{U}^T \mathcal{A}(:, :, k)\tilde{V}$

6.3. Discussion. In Table 6.3, we give the flop counts for various methods. We highlight the flop counts for the direct methods for computing the full T-SVD and the HOSVD. These two methods are easily compared, since the dimensions are known in advance. In contrast, as the tensor rank r is usually unknown, it is not possible to give a definitive cost associated with computing the decomposition (2.2). Similarly, we need to be careful when comparing the flops for the compression strategies. While the two strategies we propose are both direct methods, as is THOSVD-1, THOSVD-2 and CP approximations are computed using alternating-least-squares. Thus, we report their respective flop count in terms of the number of iterations until convergence.

Storage is compared in Table 6.3 in terms of the various parameters that define the approximate factorizations. Note that our compression strategy 2 requires the storage of a $k_1 \times k_2 \times n_3$ core tensor, along with a $n_1 \times k_1$ orthogonal matrix and a $n_2 \times k_2$ orthogonal matrix. Strategy 1 requires storage of an $n_1 \times k \times n_3$ tensor, a $n_2 \times k \times n_3$ tensor, and kn_3 numbers from $\mathcal{S}$. Hence, it is possible to achieve better compression from Strategy 2, and therefore we consider only this strategy in our numerical results.

7. Interpretation and Analysis. A limitation of decomposing a third-order tensor into a product of third-order tensors as in Theorem 5.1 is that the decom-

position is directly dependent on the orientation of the tensor. In contrast, the CP and TUCKER/HOSVD decompositions are not dependent on a specific orientation, a priori. This would suggest that such models are more suited towards data that is not orientation-specific, such as data arising in chemometrics or psychometrics.

However, there are other applications based on time series data, where the orientation of the tensor is fixed. One such example is video compression. Analogous to compressing a two-dimensional image using the matrix SVD (a classic linear algebra example, with detailed writeup in [20]), the compression strategy in (6.2) can be used to compress several images in time, such as a movie. Since the images of a movie do not change substantially from frame to frame, we expect our compression strategies to have better results than performing a matrix SVD on each separate image frame to compress the movie since our strategies take into account the tensor as a whole.[2]

7.1. Comparison with Existing Models. Given an $n_1 \times n_2 \times n_3$, the compression ratio for strategy 2 in (6.5) is

$$CR = ((k_1 \times k_2 \times n_3) + (n_1 \times k_1) + (n_2 \times k_2))/n_1 n_2 n_3$$

The compression ratio for the same tensor using the truncated HOSVD (see (2.4)) is

$$CR = (n_1 k_1 + n_2 k_2 + n_3 k_3 + k_1 k_2 k_3)/n_1 n_2 n_3.$$

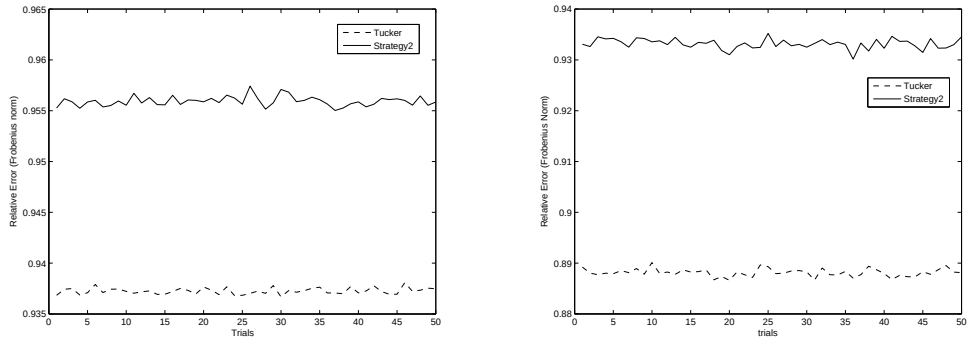
Thus, to adequately compare the performance between these two methods, we set $k_3 = n_3$ for the THOSVD approach, in which case the ratios of both are similar since the dominant term in the numerator for both corresponds to storage of the core tensor. Note that this comparison is also reasonable in that with $k_3 = n_3$, it corresponds to compression in only the first two tensor modes, similar to what we do in strategy 2. Furthermore, both compression strategies feature outer products of vectors with orthogonality properties. Therefore, it seems reasonable to numerically compare these two compression strategies: both now have some orientation dependence (compression is only in the first two modes) and both enforce a type of orthogonality constraint on the vectors involved. The CP model, as noted above, is not orientation dependent. Further, orthogonality constraints are not enforced. In fact, in many applications, such as the data arising in chemometrics, it is **undesirable** to enforce orthogonality conditions on the vectors, as doing so would compromise the qualities one is searching for in the data. Given these considerations, it did not seem appropriate to compare our compression strategy with CP in this work.

As mentioned, (6.5) is analogous to compressing a tensor in the first two dimensions, which has been a useful tool in data mining. For example, in [37], classification of handwritten digits was performed in (2.4) with the THOSVD-1 algorithm, by compressing the digit tensor of handwritten digits in the pixel (first) and variations (second) dimensions. Thus, we believe our strategy 2 approach will likely have applications in compressing certain types of data.

While the the THOSVD-2 (2.4) and (6.5) have similar expressions, they do not give the same compressed tensor when $k_3 = n_3$ and when using the same values of k_1, k_2 in both algorithms. Recall that when the THOSVD-2 representation of (2.4) is generated, it is computed iteratively using alternating least squares to solve optimization problem (2.5), while (6.5) is computed directly using the algorithm defined by

[2]Examples of video compression using Strategy 1 can be found at `http://www.math.jmu.edu/~carlam/research/videocompression.html`.

strategy 2. Therefore, one might expect for the THOSVD-2 algorithm to give better compression over a collection of randomly generated, third order tensors. However, we compared the relative errors of the approximations generated by each of the two algorithms on large numbers of randomly generated, third order tensors of various size using various truncation parameters. We found that uniformly, the two algorithms give comparable relative error in the Frobenius norm and that the THOSVD-2 algorithm usually wins in this measure by only a small margin, perhaps small enough that the extra work and slightly higher storage content for THOSVD-2 may not be justified. It also suggests that even when the THOSVD-2 is preferred, the number of iterations of the TUCKER algorithm might be reduced by initially running the compression strategy in algorithm T-SVD Compression 2. As noted previously, there is no significance to the dimensions of the tensors chosen in the two displayed results, as the results are similar across tensors of different dimensions in the sense that the margin of difference between the two methods is consistently quite small.



(a) $25 \times 50 \times 100$ tensor with $k_1 = k_2 = 10$ (b) $75 \times 75 \times 10$ tensor with $k_1 = k_2 = 20$

FIG. 7.1. *Relative Error (in Frobenius norm) of compression using THOSVD-2 and Strategy 2. Fifty trials were run on each size.*

7.2. Weighting. While THOSVD-2 has lower relative error for randomly generated tensors, our Strategy 2 algorithm can be easily adjusted if there is a known dependence of certain faces of the tensor *a priori*. In particular, the first step in (6.3) of summing over the faces of the tensor can be modified slightly. Consider a situation, such as medical time series data, where emphasis is needed in some faces. The advantage to this new type of SVD extension is that appropriating weighting can be easily performed in the compression. That is, if $\mathcal{A}$ represents the $n_1 \times n_2 \times n_3$ raw data, and d is a length-n_3 vector containing the weights associated with each (front-back) data slice, we simply replace (6.3) with

$$A = \sum_{k=1}^{n_3} d(k)\mathcal{A}(:,:,k),$$

and proceed with the rest of the compression as we did before.

8. Extending other matrix factorizations. Using the definitions in Section 4, other matrix factorizations can be extended in a similar way as the matrix SVD.

The proofs are very similar to the proofs in Section 5 and are therefore omitted. First, we give a higher-order generalization of the QR factorization.

THEOREM 8.1. *Let $\mathcal{A}$ be a real, $n_1 \times n_2 \times n_3$ tensor. Then $\mathcal{A}$ can be factored as*

$$\mathcal{A} = \mathcal{Q} * \mathcal{R},$$

where $\mathcal{Q}$ is orthogonal $n_1 \times n_1$ and $\mathcal{R}$ is a $n_1 \times n_2$ f-upper triangular tensor.

Analogous to Lemma 5.2, we have a similar result for the QR factorization.

LEMMA 8.2. *Suppose $\mathcal{A} = \mathcal{Q} * \mathcal{R}$. Then*

$$\sum_{k=1}^{n_3} A_k = \left(\sum_{k=1}^{n_3} Q_k\right)\left(\sum_{k=1}^{n_3} R_k\right) \tag{8.1}$$

where Q_k and R_k are the k-th frontal faces of $\mathcal{Q}$ and $\mathcal{R}$, respectively.

Furthermore, (8.1) gives a QR for $\sum A_k$ in the sense that $\sum Q_k$ is orthogonal and $\sum R_k$ is upper triangular.

Given a square matrix A, QR iteration can be used to find the eigenvalues of A [16, p.352]. Using Theorem 8.1, a similar algorithm can be used to obtain a third-order generalization of an eigenvalue decomposition.

THEOREM 8.3. *Let $\mathcal{A}$ be an $n \times n \times n$ tensor. Then A can be factored as*

$$\mathcal{A} = \mathcal{Q} * \mathcal{B} * \mathcal{Q}^T$$

where $\mathcal{Q}$ is an $n \times n \times n$ orthogonal tensor and $\mathcal{B}$ is an $n \times n \times n$ f-upper quasitriangular tensor.

Summing the faces of $\mathcal{A}$ gives the expected result.

LEMMA 8.4. *Suppose $\mathcal{A} = \mathcal{Q} * \mathcal{B} * \mathcal{Q}^T$. Then*

$$\sum_{k=1}^{n} A_k = \left(\sum_{k=1}^{n} Q_k\right)\left(\sum_{k=1}^{n} B_k\right)\left(\sum_{k=1}^{n} Q_k^T\right) \tag{8.2}$$

where Q_k and B_k are the k-th frontal faces of $\mathcal{Q}$ and $\mathcal{B}$, respectively.

Furthermore, (8.2) gives an eigendecomposition for $\sum A_k$ in the sense that $\sum Q_k$ is orthogonal and $\sum B_k$ is upper quasitriangular and has the same eigenvalues as $\sum A_k$.

9. Concluding Remarks. In this paper we have presented new notions of tensor factorizations based on different ideas of multiplication, orthogonality and diagonalizability of tensors. The new definitions have allowed greater flexibility in designing algorithms that do not rely on so-called tensor "flattening" in the traditional sense, and we have presented several algorithms to compress a tensor and compared those with existing methods. The result was a new way to generalize the matrix SVD to third-order tensors, in that a third-order tensor can now be written as a product of third-order tensors. The algorithm to compute the new decomposition is based on the fast Fourier transform and is computationally efficient. Extensions to the matrix QR decomposition and eigenvalue decomposition were also presented.

A restriction in the presented algorithms is that they are dependent on the orientation of the tensor. While this is not ideal in many applications, we mentioned several applications (time series data, video compression, handwritten digit classification) in which the orientation of the tensor is fixed and where our method should therefore be applicable. Furthermore, the compression strategies presented can also

be viewed as a compression in the first two dimensions. We explained how one of the compression strategies can be modified slightly to allow for weighting of the various faces of the tensor as might be relevant in certain applications.

In the discussion following the definition, we noted that the idea of using the fast Fourier transform to actually perform the product of two tensors is not particularly efficient from a computational point of view if there is special structure in one of the data tensors such as sparsity. However, by the definition, we can compute the product without invoking the FFT and capitalize on the sparsity of the corresponding block matrices instead. Furthermore, for the second compression strategy, such transforms are not needed explicitly, and we can take advantage of some of the sparsity here, using iterative methods to compute approximate rank-k factorizations of the matrix A that appears in the algorithm. In future work, we will investigate alternatives to the T-SVD when the tensors themselves are sparse or otherwise structured.

10. Acknowledgements. We would like to thank James Madison University summer 2008 REU students (NSF grant DMS-0552577) Emily Miller and Scott Ladenheim for their contribution to the coding and testing of the algorithms, as well as their manipulation of the data for video compression and handwriting data analysis described in the beginning of Section 7.

REFERENCES

[1] R. Chan and M. Ng, Conjugate Gradient Methods for Toeplitz Systems. *SIAM Review*, 38 (1996) 427-482.

[2] E. Acar, S.A. Çamtepe, M.S. Krishnamoorthy, and B. Yener, Modeling and multiway analysis of chatroom tensors. *Proceedings of the IEEE International Conference on Intelligence and Security Informatics (EMBS 2007)*, Vol. 3495 of Lecture Notes in Computer Science, Springer (2005) 256-268.

[3] C.A. Andersson and R. Bro, Improving the speed of multi-way algorithms: Part I. Tucker3. *Chemometrics and Intelligent Laboratory Systems*, 42 (1998) 93-103.

[4] C.A. Andersson and R. Bro, The N-way Toolbox for MATLAB. *Chemometrics and Intelligent Laboratory Systems*, 52 (2000) 1-4.

[5] B. Bader and T.G. Kolda, Algorithm 862: MATLAB tensor classes for fast algorithm prototyping. *ACM Transactions on Mathematical Software*, 32 (2006) No. 4 635-653.

[6] C. Beckmann and S. Smith, Tensorial extensions of the independent component analysis for multisubject FMRI analysis. *NeuroImage*, 25 (2005) 294-311.

[7] J.D. Carroll and J. Chang, Analysis of individual differences in multidimensional scaling via an N-way generalization of "Eckart-Young" decomposition. *Psychometrika*, 35 (1970) 283-319.

[8] P. Comon, Tensor decompositions: State of the art and applications. *Mathematics in Signal Processing V*, J. McWhirter and I. Proudler, eds., Oxford University Press (2001) 1-24.

[9] P. Comon, G. Golub, L.-H. Lim, and B. Mourrain, Symmetric tensors and symmetric tensor rank, *SCCM Technical Report 06-02*, Stanford University, 2006.

[10] L. De Lathauwer and B. De Moor, From matrix to tensor: Multilinear algebra and signal processing. *M*athematics in Signal Processing IV, J. McWhirter and I. Proudler, eds., Clarendon Press, Oxford (1998) 1-15.

[11] L. De Lathauwer, B. De Moor, and J. Vandewalle, A multilinear singular value decomposition. *SIAM Journal of Matrix Analysis and Applications*, 21 (2000) 1253-1278.

[12] L. De Lathauwer, B. De Moor, and J. Vandewalle, On the best rank-1 and rank-$(R_1, \ldots, R_N)$ approximation of higher-order tensors. *SIAM J. Matrix Anal. Appl.*, 21 (2000) 1324-1342.

[13] L. De Lathauwer, B. De Moor, J. Vandewalle, Computation of the Canonical Decomposition by Means of a Simultaneous Generalized Schur Decomposition. *SIAM J. Matrix Anal. Appl.*, 26 (2004) 295-327.

[14] V. De Silva and L.-H. Kim, Tensor rank and the ill-posedness of the best low-rank approximation problem. *SIAM J. Matrix Anal. Appl.* (Accepted 2006) To Appear.

[15] J.B. Denis and T. Dhorne, Orthogonal Tensor Decomposition of 3-way Tables, in R. Coppi and S. Bolasco, eds., *Multiway Data Analysis*, (Elsevier, Amsterdam, 1989) 31-37.

[16] G.H. Golub and C.F. Van Loan. *Matrix Computations*. Third Edition, Johns Hopkins University Press, Baltimore, MD, 1996.
[17] R.A. Harshman, Foundations of the PARAFAC procedure: Model and conditions for an 'explanatory' multi-mode factor analysis. *UCLA Working Papers in phonetics*, 16 (1970) 1-84.
[18] R. Henrion and C.A. Andersson, A new criteria for simple-structure transformations of core arrays in N-way principal component analysis. *Chemom. Intell. Lab. Syst.*, 47 (1999) 190-204.
[19] J. Ja' Ja', Optimal Evaluation of Pairs of Bilinear Forms, *SIAM Journal on Computing*, **8** (1979), 443-461.
[20] D. Kalman, A Singularly Valuable Decomposition: The SVD of a Matrix. *The College Mathematics Journal*, 27 (2006), No. 1 2-23.
[21] Henk A.L. Kiers, Tuckals Core Rotations and Constrained Tuckals Modelling. *Statistica Applicata*, 4 (1992) 661-667.
[22] Henk A.L. Kiers, Towards a standardized notation and terminology in multiway analysis, *J. Chemometrics*, **14** (2000) 105-122.
[23] E. Kofidis and P.A. Regalia, On the best rank-1 approximation of Higher-Order Supersymmetric Tensors, *SIAM J. Matrix Anal. Appl.*, 23 (2001), 863-884.
[24] T.G. Kolda, Orthogonal Tensor Decompositions. *SIAM Journal of Matrix Analysis and Applications*, 23 (2001) 243-255.
[25] T.G. Kolda and B.W. Bader, Higher-order web link analysis using multilinear algebra, *Proceedings of the 5th IEEE International Conference on Data Mining (ICDM 2005)*, IEEE Computer Society (2005) 242-249.
[26] T.G. Kolda and B.W. Bader, Tensor Decompositions and Applications. *SIAM Review* (Accepted June 2008), To Appear.
[27] P.M. Kroonenberg, *Three-mode principal component analysis: Theory and applications*. DSWO Press, Leiden. 1983.
[28] J.B. Kruskal, Rank, Decomposition, and Uniqueness for 3-way and N-way arrays, in R. Coppi and S. Bolasco, eds., *Multiway Data Analysis*, (Elsevier, Amsterdam, 1989) 7-18.
[29] N. Liu, B. Zhang, J. Yan, Z. Chen, W. Liu, F. Bai, and L. Chien, Text representation: From vector to tensor. *Proceedings of the 5th IEEE International Conference on Data Mining (ICDM 2005)*, IEEE Computer Society (2005) 725-728.
[30] C.D. Martin, The Rank of a $2 \times 2 \times 2$ Tensor, submitted *MAA Monthly* (2008).
[31] C.D. Martin, Structure Preserving Tensor Multiplication for Tensors of High Order, In Preparation, To be submitted to SIAM Nov. 2008.
[32] C.D.M. Martin and C.F. Van Loan, A Jacobi-type Method for Computing Orthogonal Tensor Decompositions, *SIAM J. Matrix Anal. Appl. Special Issue: Tensor Decompositions* (Accepted 2008), To Appear.
[33] E. Martínez-Montes, P.A. Valdés-Sosa, F. Miwakeichi, R.I. Goldman, and M.S. Cohen, Concurrent EEG/fMRI analysis by multiway partial least squares. *NeuroImage*, 22 (2004) 1023-1034.
[34] F. Miwakeichi, E. Martínez-Montes, P.A. Valdés-Sosa, N. Nishiyama, H. Mizuhara, and Y. Yamaguchi, Decomposing EEG data into space-time-frequency components using parallel factor analysis, *NeuroImage*, 22 (2004) 1035-1045.
[35] J.G. Nagy and M.E. Kilmer, Kronecker product approximation for preconditioning in three-dimensional imaging applications. *IEEE Trans. Image Proc.*, 15 (2006) 604-613.
[36] N. Pitsianis, *The Kronecker product in approximation and fast transform generation*, Ph.D. Thesis, Cornell University, Ithaca, NY, 1997.
[37] B. Savas and L. Eldén, Handwritten digit classification using higher order singular value decomposition. *Pattern Recognition*, 40 (2007) 993-1003.
[38] N. Sidiropoulos, R. Bro, and G. Giannakis, Parallel factor analysis in sensor array processing, *IEEE Transactions on Signal Processing*, 48 (2000) 2377-2388.
[39] A. Smilde, R. Bro, P. Geladi, *Multi-way Analysis: Applications in the Chemical Sciences*, (Wiley, 2004).
[40] J.M.F. Ten Berge, Kruskal's polynomial for $2 \times 2 \times 2$ arrays and a generalization to $2 \times n \times n$ arrays, *Psychometrika*, **56** (1991), 631-636.
[41] L.R. Tucker, Some mathematical notes on three-mode factor analysis. *Psychometrika*, 31 (1966) 279-311.
[42] M.A.O. Vasilescu and D. Terzopoulos, Multilinear analysis of image ensembles: TensorFaces. *Proceedings of the 7th European Conference on Computer Vision (ECCV 2002)*, Vol. 2350 of Lecture Notes in Computer Science, Springer (2002) 447-460.
[43] M.A.O. Vasilescu and D. Terzopoulos, Multilinear Image Analysis for Face Recognition. *Pro-*

ceedings of the International Conference on Pattern Recognition (ICPR 2002), Vol. 2, Quebec City, Canada, Aug, 2002, 511-514.

[44] M.A.O. Vasilescu and D. Terzopoulos, Multilinear subspace analysis of image ensembles. *Proceedings of the 2003 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR 2003)* 2003 93-99.

[45] C.F. Van Loan, The Ubiquitous Kronecker Product. *Journal of Computational and Applied Mathematics*, 123 (2000), 85-100.

[46] T. Zhang and G.H. Golub, Rank-one approximation to high order tensors. *SIAM J. Matrix Anal. Appl.*, 23 (2001) 534-550.