Title

Ivan Čeh
Computer Science
cehmail@gmail.com

Petra Brčić
Applied Mathematics
petrabrcic94@gmail.com

Sandro Lovnički
Computer Science
lovnicki.sandro@gmail.com

May 13, 2018

Abstract: Operations with tensors, or multiway arrays, have become increasingly prevalent in recent years. Traditionally, tensors are represented or decomposed as a sum of rank-1 outer products, but in this paper we explore an alternate representation of tensors which shows promise with respect to the tensor approximation problem. We also discuss implications for extending basic algorithms such as the power method, QR iteration, and Krylov subspace methods. To conclude, we present two applications of this theoretical framework; image deblurring and face recognition.

Keywords: tensor, deblurring, face recognition.

I. Introduction

A tensor is a multi-dimensional array of numbers. For example, we could say that vector $v \in \mathbb{R}^{n_1}$ is a first-order tensor and similarily, matrix $M \in \mathbb{R}^{n_1 \times n_2}$ is a second-order tensor. Therefore, a third-order tensor is $\mathcal{A} \in \mathbb{R}^{n_1 \times n_2 \times n_3}$.

In this paper, we provide a setting in which the familiar tools of linear algebra can be extended to better understand third-order tensors. Significant contributions to extending matrix analysis has been made in the works of Misha E. Kilmer and others in [1] and [2].

In sections that follow, we first introduce the basic notions and definitions of this new framework. We present methods which will be used in applictions and continue with two examples (image deblurring and face recognition), showing our implementation in Octave and human-readable results.

II. Theoretical Framework

In this section, we will first describe notation that is convenient for manipulating third-order tensors and which will be used throughout the paper. Also, we define basic operations on tensors such as multiplication,...

i. Notation and Conversions

Each third-order tensor $\mathcal{A} \in \mathbb{R}^{l \times m \times n}$ can be viewed as n $l \times m$ matrices stacked upon eachother. Firthermore, it can also be viewed as a $l \times m$ matrix of vectors of length n. It is convenient to break tensor into "slices" and "tubal elements".

Definition 1. *Let $\mathcal{A} \in \mathbb{R}^{l \times m \times n}$. We introduce the following notation:*

- *i-th **frontal slice** $\mathcal{A}^{(i)} \equiv \mathcal{A}(:,:,i)$*
- *i-th **lateral slice** $\overrightarrow{\mathcal{A}}_i \equiv \mathcal{A}(:,i,:)$*
- *i,k-th **tubal scalar** $\mathbf{a}_{ik} \equiv \mathcal{A}(i,k,:)$*

In order to define tensor multiplication, we now present some conversions between tensor and matrix objects. To wrap a tensor into a matrix, functions `circ` and `matvec` will be used, and to unwrap a tensor from the return value of `matvec`, we use `foldn` operator. How those operators work on tensors and matrices is now presented.

$$\texttt{circ}(\mathcal{A}) = \begin{bmatrix} \mathcal{A}^{(1)} & \mathcal{A}^{(n)} & \mathcal{A}^{(n-1)} & \dots & \mathcal{A}^{(2)} \\ \mathcal{A}^{(2)} & \mathcal{A}^{(1)} & \mathcal{A}^{(n)} & \dots & \mathcal{A}^{(3)} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ \mathcal{A}^{(n)} & \mathcal{A}^{(n-1)} & \mathcal{A}^{(n-2)} & \dots & \mathcal{A}^{(1)} \end{bmatrix}$$

$$\mathtt{matvec}(\mathcal{A}) = \begin{bmatrix} \mathcal{A}^{(1)} \\ \mathcal{A}^{(2)} \\ \vdots \\ \mathcal{A}^{(n)} \end{bmatrix}$$

Correspondingly, `foldn` operator is defined so that $\mathtt{foldn}(\mathtt{matvec}(\mathcal{A}), n) = \mathcal{A}$.

ii. Tensor Operators

Now, with established notation and conversions, we are equipped to define needed tensor operators.

Definition 2. *Let $\mathcal{A} \in \mathbb{R}^{l\times p\times n}$ and $\mathcal{B} \in \mathbb{R}^{p\times m\times n}$. The t-product*[1] *$\mathcal{A} * \mathcal{B}$ is a $l \times m \times n$ tensor defined as*

$$\mathcal{A} * \mathcal{B} := \mathit{foldn}(\mathit{circ}(\mathcal{A}) \cdot \mathit{matvec}(\mathcal{B}), n)$$

Definition 3. *If $\mathcal{A} \in \mathbb{R}^{l\times m\times n}$, then $\mathcal{A}^T \in \mathbb{R}^{m\times l\times n}$ is obtained by transposing each frontal slice and reversing the order of frontal slices 2 to n.*

It could be observed that $m \times 1 \times n$ tensors are just matrices oriented laterally. Furthermore, we could twist a $m \times n$ matrix to make a $m \times 1 \times n$ tensor. Following this line of thought, we introduce operators `twist` and `squeeze` defined as

$$\mathtt{squeeze}(\overrightarrow{\mathcal{A}}) = A \in \mathbb{R}^{m\times n}$$

$$\mathtt{twist}(\mathtt{squeeze}(\overrightarrow{\mathcal{A}})) := \overrightarrow{\mathcal{A}}$$

Observing the established setting, it is useful to introduce $\mathbb{K}_n^m$ - the set of all $m \times n$ matrices oriented as $m \times 1 \times n$ tensors. For convenience, we write just $\mathbb{K}_n$ when $m = 1$ and use it to represent the set of all tubal scalars of length n. As we mentioned before, $l \times m \times n$ tensor can be viewed as a $l \times m$ matrix of tubal scalars, so it is natural to say that $K_n^{l\times m}$ is the set of all $l \times m \times n$ tensors, i.e. $K_n^{l\times m} \equiv \mathbb{R}^{l\times m\times n}$.

This new definitions have as a consequence the consistency of operations on $\mathbb{K}_n^{l\times m}$ which we stress in the following list:

1. multiplication, factorization, etc. based on t-product reduce to the standard matrix operations and factorizations when $n = 1$.
2. outer-products of matrices are well-defined.
3. given $\overrightarrow{\mathcal{X}}, \overrightarrow{\mathcal{Y}} \in \mathbb{K}_n^m$, $\overrightarrow{\mathcal{X}}^T * \overrightarrow{\mathcal{Y}}$ is a scalar.

III. Image Deblurring

In this section...

i. Subsection

Based on...

IV. Face Recognition

As we could have seen in...

References

[1] Misha E. Kilmer, Karen Braman, Ning Hao. Third Order Tensors as Operators on Matrices: A Theoretical and Computational Framework with Applications in Imaging

[2] Misha E. Kilmer, Carls D. Martin Factorization Strategies for Third-order Tensors

[1] our Octave implementation of t-product is called `tproduct`.