Accelerating Big Data Analytics

SCIENTIFIC SEMINAR

submitted by
Tick Son Wang

NEUROSCIENTIFIC SYSTEM THEORY
Technische Universität München

Prof. Dr Jörg Conradt

Supervisor:	Dr. Cristian Axenie
Final Submission:	06.07.2016

In your final hardback copy, replace this page with the signed exercise sheet.

Abstract

"Big data" is a terminology associated with data which are intrinsically big in volume and modality. Due to the multi-modality, these data can be represented naturally as a tensor but tensor representation actually offers more than just convenient data indexing. If these data are first collapsed into bi-modal form before being processed, the latent interconnections of the data between the remaining modes and collapsed modes are lost and this significantly reduces the value of the information being extracted. There is a domain of tensor algebra which can be applied to analyze these data directly in multi-modal form to preserve these latent interconnections. Tensor decomposition is recently a very popular technique used mainly to estimate latent spaces and to extract higher-order components from multi-modal data.

The goal of this work is to expose the formalism of tensor decomposition framework and to explore its usability in terms of its suitability for parallel computing and the current state of available supporting software. Furthermore, this work will also give an overview of the application of tensor decomposition in neuroinformatics, in particular in EEG-related research.

Contents

Chapter 1

Introduction to Big Data

Due to the recent explosive advancement in various aspects of information technology such as faster computing platform, cheaper storage, more integrated sensors, more accessible internet connections, we are now generating, gathering and processing data at an amazing rate. Currently, we are generating more data over the past two years than in the entire previous history of human race. However, volume isn't the only aspect that is growing, data are also being harvested in more modalities. For example, in a internet mouse-click analysis, not only the websites and the frequency of clicks are being recorded but also the time of the clicks, the geological position of the users and even the accessing devices can be recorded.

The availability of these multi-modal data encourages data scientist to explore the possibility to extract higher-order latent relationships between the different modalities of the data. A method to achieve this is by applying multi-linear analysis onto the data. For example, a multi-linear analysis on the web-clicks data of frequency, time and geological position modes would not only be able to find out the nationalities of the users and the peak times of the website traffic but also exactly the times at which users of specific regions are active. This evidently adds on important cues to the information obtained from the data.

To perform a n-way multi-linear analysis, the data have to be represented in a form of a n-dimensional array, which is a tensor. Despite its potential in containing higher-order semantics, the tensor representation bears its downsides in multiple aspects. Its first problem is its volume. A 3-d tensor of 1000 entries in each dimensions already requires 10^9 storage units. Secondly, there is no closed form solution for $n > 3$-dimensional tensor decomposition. The only way to find the definite solution, if it even exists and if it is even unique, is by brute-forcing through all permutations of the decomposition. Fortunately, there are ways to approximate approximate it.

Chapter 2

Tensor Decomposition

Useful References:
[DL08], [DLN08], [KB09], [SP14], [Cic13], [FKS07], [SK12]

2.1 Definition of Tensor

Tensors are n-dimensional arrays used to represent the linear mapping between input and output quantities. The **order** (also *degree* or *mode*) of a tensor refers to the minimum number of indices required to address each element of the tensor. For example, the subclasses of tensors are scalars S (0-th order) , vectors V_i (first order), matrices M_{ij} (second-order) and higher-order tensors, for example T_{ijk} for third-order tensors. Although the mathematical fundamentals and algorithms for solving different order tensor decompositions are different, they extends from that of the third-order. Hence, the following sections will only concisely explain that.

2.2 Concept of 3D Tensor decomposition

Simple matrix decomposition is the main applicable decomposition concept for higher-order tensor because other types of decomposition concepts like SVD[1], EVD[2], QR[3] and etc. are simply either not-defined or NP-hard for higher-orders. A simple matrix, $M_{simple} = \{a \circ b^\intercal \in \mathbb{R}^{m,n} : \ \forall a \in \mathbb{R}^m, b \in \mathbb{R}^n\}$, is a rank-1 matrix, of which

[1]Singular value decomposition
[2]Eigenvalue decomposition
[3]QR-decomposition

all the rows or columns are linearly dependent.

If matrix $X \in \mathbb{R}^{m,n}$ is rank-N, where $N \leq min(m,n)$ then it can be decomposed into N simple matrices:

$$X = \sum_{n=1}^{N} M^n_{simple} = \sum_{n=1}^{N} a^n \circ b^n = a^1 \circ b^1 + ... + a^N \circ b^N = AB^\intercal$$

where $A = [a^1 ... a^N]$ and $B = [b^1 ... b^N]$, or expressed element-wise:

$$X_{ij} = \sum_{n=1}^{N} a^n_i b^n_j = a^1_i b^1_j + a^2_i b^2_j + ... + a^N_i b^N_j$$

where a^n_i, b^n_j are elements of $a^n \in \mathbb{R}^m, b^n \in \mathbb{R}^n$.

Following the same concept, the 3D Tensor Decomposition of a tensor $T \in \mathbb{R}^{m,n,p}$ of supposedly rank-N is defined as:

$$T = \sum_{n=1}^{N} a^n \circ b^n \circ c^n = a^1 \circ b^1 \circ c^1 + ... + a^N \circ b^N \circ c^N = A \otimes B \otimes C$$

Intuitively, this means that this rank N tensor T can be decomposed into N rank-1 tensor, which each can be further decomposed into 3 vectors a^n, b^n, c^n. By concatenating these vectors into the corresponding matrices A, B, C, this decomposition can be rephrase into the matrix Kronecker product of A, B, C.

An analytical method to compute the above decomposition is not defined but rather it is done through numerical least square optimization of the error between T and $A \otimes B \otimes C$. However, this model poses very weak constraints because the tensor is assumed to be arbitrary, which make the optimization inefficient. In practice, tensors are assumed to adopt special structures, typically the **Tucker** or the **Kruskal**, so that the corresponding optimization models can be applied to approximate the decomposition.

Unlike matrix, determining rank(T) is NP-hard and if T is a IxJxK tensor then max(I,J,K) $\leq$ rank(T) $\leq$ min(IJ, IK, JK). Other difficulty in this aspect is the fact that a tensor can even have multiple ranks. However, the rule of thumb in practice is to assume the rank of the tensor to be as high as such that the decomposition is computationally affordable and as low as such that the reconstruction error is acceptable.

2.3 CP(CANDECOMP/PARAFAC)

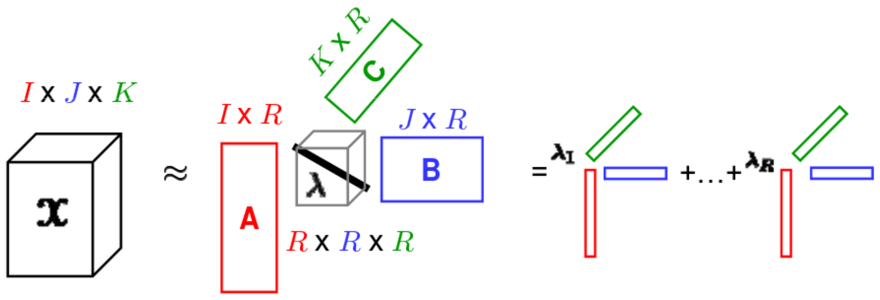

Figure 2.1: **Kruskal** model (image taken from [FKS07])

The **Kruskal** model is intuitively a higher dimensional eigenvalue decomposition. The tensor is assumed to have rank R and similar to the model described above, this tensor decomposes into R simple tensors, each assigned with a weight λ_r. This can be reformulated into the 3 corresponding matrices $A \in \mathbb{R}^{I,R}, B \in \mathbb{R}^{J,R}, C \in \mathbb{R}^{K,R}$, and an additional super-diagonal core component $\lambda \in \mathbb{R}^{R,R,R}$. In general the expression for a **Kruskal** tensor of rank R is

$$X = \sum_{r}^{R} \lambda_r a_r \otimes b_r \otimes c_r$$

where λ is corresponding weight of the r-th simple tensor, which are formed by the outer-products of vectors a_r, b_r, c_r. Refer to Algorithm A.1 for the algorithm to approximate the CP decomposition.

2.4 Tucker3

The **Tucker** decomposition model, on the other hand, is to be interpreted as a higher-order PCA. As shown in the 2.2, it decomposes into three factor matrices of $A \in \mathbb{R}^{I,R}$, $B \in \mathbb{R}^{J,S}$, $C \in \mathbb{R}^{K,T}$ and a dense core tensor $G \in \mathbb{R}^{R,S,T}$, where $\{R, S, T\}$ indicate the rank of the tensor in the corresponding dimension. The formula for this representation is

$$X = \sum_{i}^{I} \sum_{j}^{J} \sum_{k}^{K} g_{i,j,k} a_i \otimes b_j \otimes c_k$$

Refer to Algorithm A.2 for the algorithm to approximate the CP decomposition.

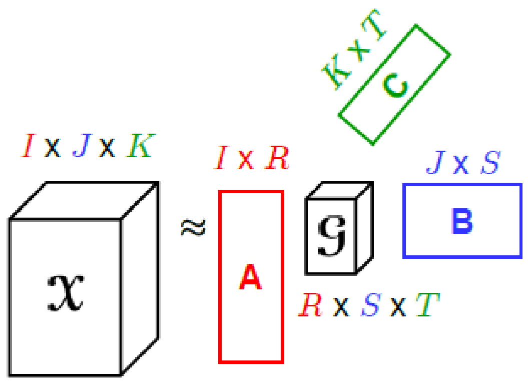

Figure 2.2: **Tucker** model (image taken from [FKS07])

2.5 Comparison between CP and Tucker3

	CP	Tucker3
Intuition	Decomposes into N rank 1 tensor, each representing one factor tensor with a certain weight. The factor tensors represent the latent spaces of the tensor.	Decomposes into 3 matrices containing the subspaces of the corresponding mode, and a core tensor which describes the linear relationship between the dimensions of the modes.
Application	Latent parameters estimation	Subspace estimation, tensor compression
Characteristics	1 Super-diagonal core. 2 A, B, C are not necessarily linearly independent. 3 Decomposition is generally unique 4 Only requires the specification of rank R.	1 G is a dense core. 2 A, B, C are expected to be orthonormal 3 Decomposition is generally non-unique. 4 Requires the specification of rank R, S, T.

Table 2.1: Comparison of different aspects of CP and Tucker3 decomposition

Chapter 3

Parallelizing tensor decomposition

3.1 Alternating Least Squares ALS

ALS is a common method used to approximate a matrix X by assuming that X can be decomposed into $AB^\intercal$. The approximation aims to minimize $\|X - AB^\intercal\|$, by iteratively computing the below operations until convergence.

$$\begin{aligned} A_{new} &= \underset{A}{\operatorname{argmin}} \|X - AB^T\|_2 \text{ using fixed } B_{old} \\ B_{new} &= \underset{B}{\operatorname{argmin}} \|X - AB^T\|_2 \text{ using fixed } A_{old} \\ X_{new} &= A_{new} B_{new}^\intercal \end{aligned}$$

There are a few important observations from the algorithm which displays the potential for parallelized implementation. Firstly, two alternating part of the algorithm are actually identical operations onto different variables. Secondly, the algorithm is to be repeated to update A, B and X in each iteration until $AB^\intercal$ converges to X, which is the repetition of the same process onto changing data. This form of algorithm fits naturally into the Single-Intstruction-Multiple-Thread SIMT computation model and can be efficiently implemented on a GPU.

As presented in Algorithm A.1 and Algorithm A.2. The tensor decompositions of both models are formulated into similar ALS problems. Since the parallelized implementation scheme of both algorithm is similar for both, the following section briefly demonstrate the parallelizable aspect of tensor decomposition using the CP decomposition model.

3.2 Parallelizing Tensor decomposition

As shown in the algorithm A.1, CP decomposition is formulated into an ALS-problem and the fundamental mathematical operation to solve the ALS is in fact matrix multiplications. CUDA is a technology for certain series of NVIDIA GPU which can automatically parallelize matrix multiplication by distributing the computation of parts of the matrix to different thread blocks. However, to optimize the parallel computation of tensor decomposition, one needs to understand the dependencies of the steps in the algorithm to efficient execution paths for the parallel threads.

In general, the algorithm can be simplified into the update expressions below for the ease of understanding:

$$A^{k+1} = X_a^k(C^k \odot B^k)((C^k)^\intercal C^k * (B^k)^\intercal B^k)^+ \Lambda^{-1} \tag{3.1}$$
$$B^{k+1} = X_b^k(C^k \odot A^k)((C^k)^\intercal C^k * (A^k)^\intercal A^k)^+ \Lambda^{-1} \tag{3.2}$$
$$C^{k+1} = X_c^k(B^k \odot A^k)((B^k)^\intercal B^k * (A^k)^\intercal A^k)^+ \Lambda^{-1} \tag{3.3}$$
$$\Lambda^{k+1} = diag[\{\lambda_i\}_{i=1\ldots R}] \ , \ \lambda_i = \|\hat{a}_i + \hat{b}_i + \hat{c}_i\|^2 \tag{3.4}$$
$$vec(X^{k+1}) = (C^{k+1} \odot B^{k+1} \odot A^{k+1})\Lambda^{k+1} \tag{3.5}$$

First of all, the above algorithm has to be repeated until the convergence of the variables. which means that on the very top level of the computation, it is a conditional loop. Within this loop, step 3.4 and 3.5 depends on the results of step 3.1, 3.2, 3.3, hence they have to be executed thereafter. However, step 3.4 and 3.5 are merely simple vector and matrix operations and the usage of CUDA to parallelize them is straightforward.

Due to the amount of dependencies and the amount of computation, steps 3.1-3.3 are the bottle neck in terms of computation and memory in this algorithm. However, they are fully parallelizable because they each computes $A^{k+1}, B^{k+1}, C^{k+1}$ using the same set of operations onto the same set of variables estimated from previous iteration $A^k, B^k, C^k, X^k, \Lambda^k$. Figure 3.2 shows the dependencies of step 3.1 -3.3 in different stages of its execution path.

If observed carefully, all step 3.1-3.3 consist of the same components, which are:

$$\begin{array}{lllll} A^{k+1} = & [X_a^k] & [(C^k \odot B^k)] & [((C^k)^\intercal C^k * (B^k)^\intercal B^k)^+] & [\Lambda^{-1}] \\ \text{Update} = & [\text{SlabMatrix}] * & [\text{Khatri-Rao Product}] * & [\text{Pseudo Inverse}] * & [\Lambda^{-1}] \end{array}$$

Each component is independent of the other and can be computed in parallel. These components are depicted as blocks of different colors in in Figure 3.2. The orange block is the computation of the pseudo-inverse part, the light blue block is the Khatri-Rao product part, the pink block is the slab matrix part and the green block is the Σ^{-1} part. The yellow block the final matrix multiplication.

The **Slab Matrix (pink)** part actually doens't require any additional computation. The tensor X is linearized and X_a, X_b, X_c can be accessed simply by changing the indexing scheme correspondingly. See Figure 3.1 for illustration.

The **Khatri-Rao Product (light blue)** part is for each variable requires the other two variables as input and computes the corresponding Khatri-Rao product which decomposes into a series of scalar-vector multiplication. The Σ^{-1} **green** part is simply the reciprocal of each of the diagonal elements of Σ^{-1} which only requires very little computation.

The **Pseudo-Inverse (orange)** part is doubtlessly the most computational intensive part, which decomposes into the forming of the quadratic matrix form. The quadratic for is of course symmetric, and hence the computation and storage of the lower triangle half is sufficient. Then the Hadamard-product, which is simple the element-wise product between the quadratic forms of the two variables, which is also symmetric and only the lower triangle half has to be computed. Next, coming into the true computational bottle neck of the whole operation: the pseudo inverse of the product so far.

After that, all the parts will be multiplied together in the **yellow** block to get the update the corresponding variables. For each variable, there are three matrix multiplications, each can be parallelized directly by CUDA. Note that the direction of the computation of this series of matrix multiplication depends on the dimension of the tensor in the mode of variable and the number of components of the decomposed tensor. For example, let $A \in \mathbb{R}^{I,R}$, if $T > R$ then it is more efficient to multiply from the right and vise-versa.

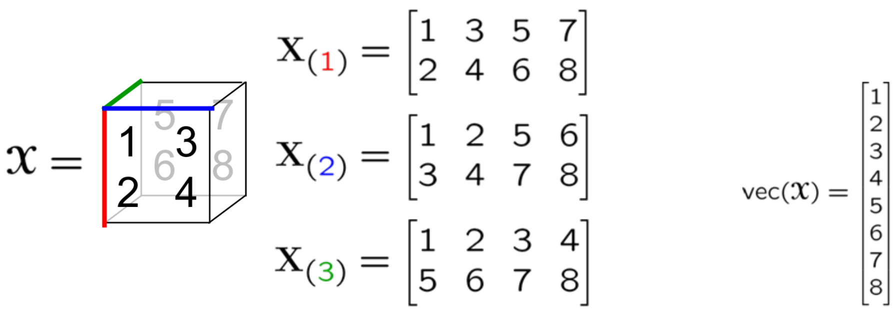
$\mathcal{X} =$
$\mathbf{X}_{(1)} = \begin{bmatrix} 1 & 3 & 5 & 7 \\ 2 & 4 & 6 & 8 \end{bmatrix}$
$\mathbf{X}_{(2)} = \begin{bmatrix} 1 & 2 & 5 & 6 \\ 3 & 4 & 7 & 8 \end{bmatrix}$
$\mathbf{X}_{(3)} = \begin{bmatrix} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \end{bmatrix}$
$\text{vec}(\mathcal{X}) = [1\ 2\ 3\ 4\ 5\ 6\ 7\ 8]^T$

Figure 3.1: Slab of a tensor (image taken from [FKS07])

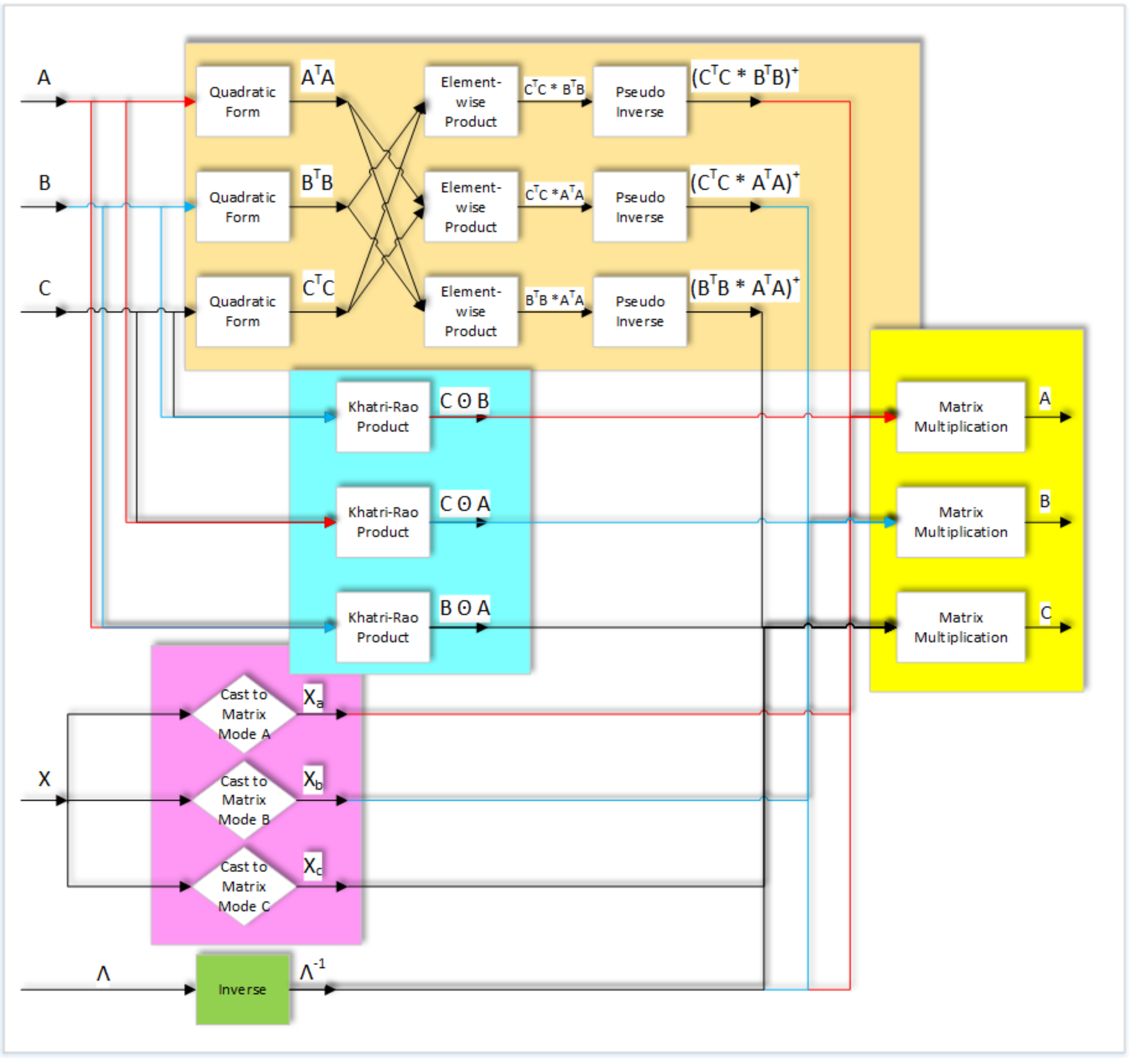

Figure 3.2: Parallel Computation of CP Decomposition

3.3 Parallel Matrix Multiplication

Useful References:
[Kre13], [FSH04], [TCF16], [KWm12], [Dur]

CUDA is an API created by NVIDIA which allows developers to use CUDA-enabled GPU for parallel computing. GPU has the capability to perform vectorized operations which makes the computation extremely fast. This means that in order to maximize the potential of the parallel computation, the data feed has to be able to keep up. Loading from local registers or shared memory takes one clock-cycle but loading from global memory takes from 1 to 1000 clock cycles. If the size of the data doesn't fit into the shared memory, then computation will have to be stalled until the data is loaded from the global memory. Hence, in order to achieve high efficiency, the computation pipeline has to maximize the number of floating point operations per global floating point loads (flop/gfpl). Figure 3.3 shows the generic memory layers architecture of a GPU.

For large-scale matrix multiplications, its BLAS(Basic Linear Algebra Subprograms) library, cuBLAS, implements matrix multiplication following the GEMM(General Matrix-Matrix Multiplication) paradigm to speed up large scale matrix-matrix multiplication. The basic idea is to divide large matrices into tiles of sub-matrices and computes each sub-matrix in a thread block simultaneously. Its effect in terms of increasing flop/gfpl will be demonstrated in the following.

Naive Matrix Multiplication

Let A and B be two large matrices $\in \mathbb{R}^{NxN}$ and C=A*B $\in \mathbb{R}^{NxN}$. The kernel to compute C would be:

```
for(int i=0; i<N; i++){
  for(int j=0; j<N; j++){
    C[i*N+j] =0;
    for(int k=0; k<N; k++){
      C[i*N+j]+=A[i*N+k]*B[k*N+j];
    }
  }
}
```

Using this kernel, the complexity of computation is $O(N^3)$. For each element, it requires $2N$ global memory floating point loads (a row of N elements from A and a column of N elements from B) but it only does N *Mul* and N *Add*, which means 1 flop/gfpl (floating point operations / global floating point loads).

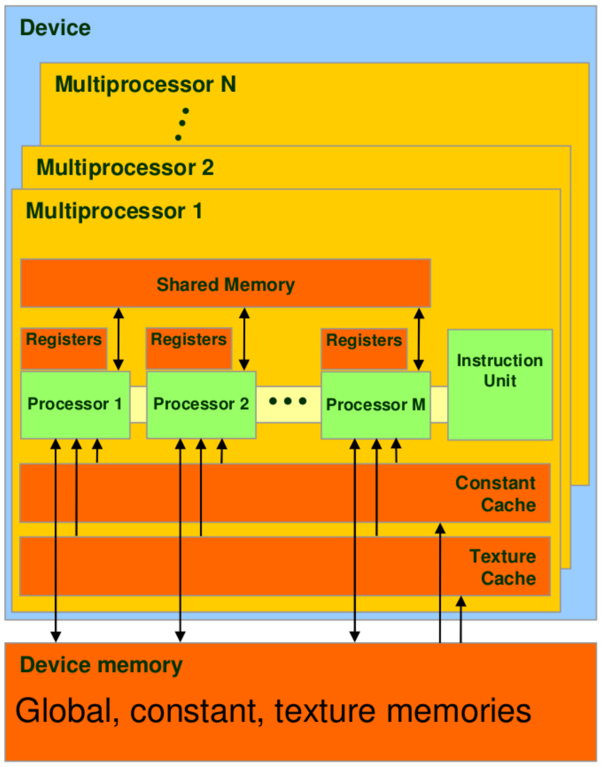

Figure 3.3: GPU Memory Architecture (image taken from [Dur])

Tiled Matrix Multiplication(GEMM)

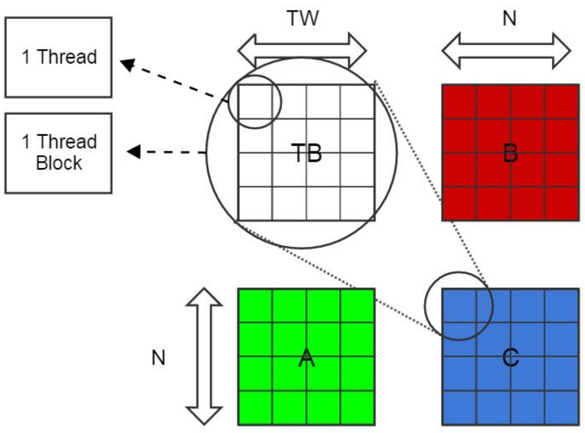

Figure 3.4: Threads and thread block

Keywords in the context of GEMM:

- A kernel refers to a function which is called by many parallel threads. In the case of GEMM, it is the function multiply(i,j) which multiplies row-i of sub-matrix A' with column-j of sub-matrix B'. The $thread_{ij}$ will call multiply(i,j)
- Each $thread_{ij}$ in a thread block loads a row from sub-matrix A' and a column from sub-matrix B' and then computes the intermediate result of the $element_{ij}$ using the multiplication kernel.
- Each thread block has a shared memory which is shared by all of its (TW^2) threads. (see Figure 3.4)

GEMM divides C into $(N/TW)^2$ tiles and process each tile with a thread blocks. A thread block consists of TW^2 threads, each computing the result of the corresponding element of C. At the start, each thread block will load in the corresponding data block from A and B from global memory into its shared memory and each thread will load from shared memory, compute and store the intermediate result in the local register. The same computation scheme has to be repeated for (N/TW) phases, iterating though the remaining data blocks to complete the matrix multiplication. (see Figure 3.5.)

The crucial difference of using GEMM is that now, in each phase, each thread block does $(2 * TW^2)$ global memory floating point loads and stores the data in shared memory. It then performs $(TW^2 * 2 * TW)$ MUL or Add operations by repeatedly reusing the data from the shared memory. This amounts to TW flop/gfpl and essentially means TW times less global memory access latency.

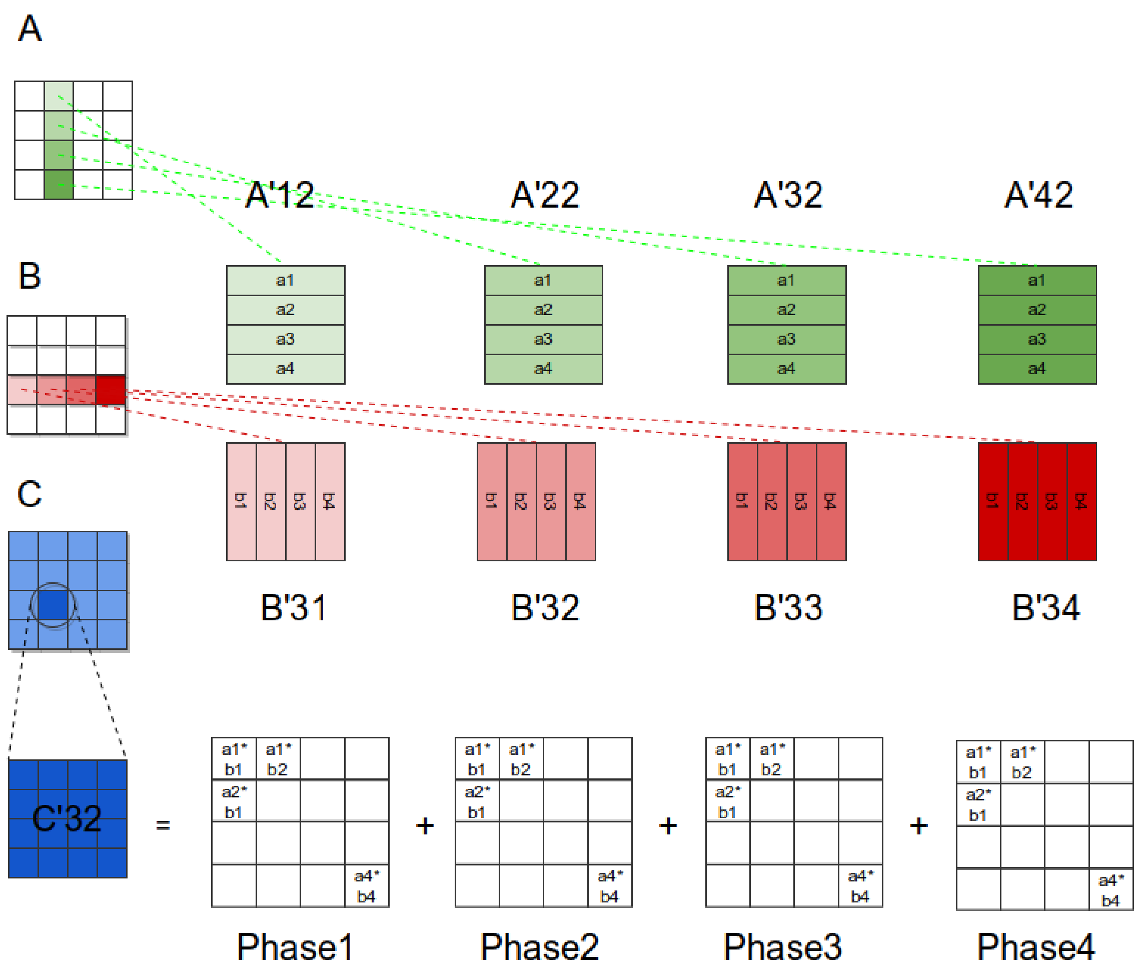

Figure 3.5: GEMM

Chapter 4

Tensor in Time Series Data

Due to the capability of tensor decomposition to factorize the inter-modal relationships in a mult-modal data, it is a useful tool in time-series data analysis. The typical applications of tensor decomposition in time-series data analysis are:

1 **Noise removal** using tensor decomposition involves the process of decomposing the tensors into individual components, removing the components of low significance and then reconstruction. Compared to separate noise reduction of w.r.t different modes of the data, this method only reduces noise that is insignificant w.r.t all modes, which avoids the removal of potentially useful inter-modal information.

2 **Multi-aspect trend analyis** is a method of finding important correlations within multi-modal time-series data. A typical scenario is the analyis of web clicks data, which encorporates 3 modes: url-mode, time-mode, user-mode. Such analysis onto this data aims to discover some trends in the data, for example, which type of website is likely to be accessed by which group of user at which time. Each component of the CP decomposition is one of this type of three-way relationships.

3 **Multi-aspect feature discovery** is also a very important capability of tensor decomposition. CP decomposition decomposes time-series tensors into independent components which can be deemed as (independent) events in the time series. A type of event or a combination of events which consistently appear in a set of time series data can be used as features for the classification of that set of data.

4.1 Tensor Decomposition in EEG-related Research and Application

Electroencephalography(EEG) data are multi-channel recordings of the electrical activity of different regions of the brain. It is typically represented in the form of multi-channel time-series data and hence intrinsically inherit a temporal mode and a spacial mode. In addition to that, depending on the requirements of the experiment, more additional modes can be introduced to the data such as:

1 subject node: across different subjects

2 experimental condition node: across different conditions of the experiment

3 spectral node: across different frequencies

4 trial node: across different experiment trials

As a result, the tensor representation of EEG is required in order to analyze the inter-modal relationships in the data. In EEG-related research, tensor decomposition is mainly used to localize the origin of certain distinctive brain activities, to investigate certain hypotheses in cognitive or clinical neuroscience, and to research or design certain mind-machine interface. The tucker model decomposes the tensor into components, which are related by the core tensor. Problems or experiments which uses Tucker model for the decomposition requires more complex analysis to describe the relationships between the decomposed components. The CP model, on the other hand, decomposes the tensor into independent rank-1 tensors, which are viewed as different features or events in the data. Due to this convenient property of CP model, it is in general the preferred model. On a more technical level, the typical tensor decomposition applications using CP are:

- **Canonical Polyadic decomposition(CPD)** of a spatial-temporal-spectral EEG tensor using CP model decomposes the data series into components of events. Each event is a three-way coupling of a certain distinctive spatial, temporal, spectral behavior which is potentially connected to a certain neurological event or a certain disease. This CPD is generalizable to higher order tensors, see Figure 4.1. CPD also has the properties that each component is independent of all other components in all modes, this provides a convenient way to extract or filter unwanted signals.

- **Feature Analysis** in EEG in a process of analyzing the relationship between a feature and a condition. It usually extends the normal spatial-temporal-spectral EEG tensors into higher-order tensors by including sample or subject mode and experimental condition mode. For example, if the EEG tensor is extended with a subject mode, which covers a range of subjects of different age, the CPD of this tensor will reveal the correlation between certain spatial-temporal-spectral behavior and the age of subjects. This is done by analyzing the subject mode of the corresponding spatial-temporal-spectral behavior, see Figure 4.1. By the same intuition, an additional experiment condition mode can reveal the contribution of different experiment condition in inducing different spatial-temporal-spectral behavior.

- **Feature Selection** of EEG data is process of discovering significant features. After the CPD of the EEG-data of many different subjects of a certain neurological disorder, the independent components of the multi-modal EEG data are considered as different independent features of the data. Machine learning schemes are then applied to select the most important EEG-features to train a classifier of this neurological disorder.

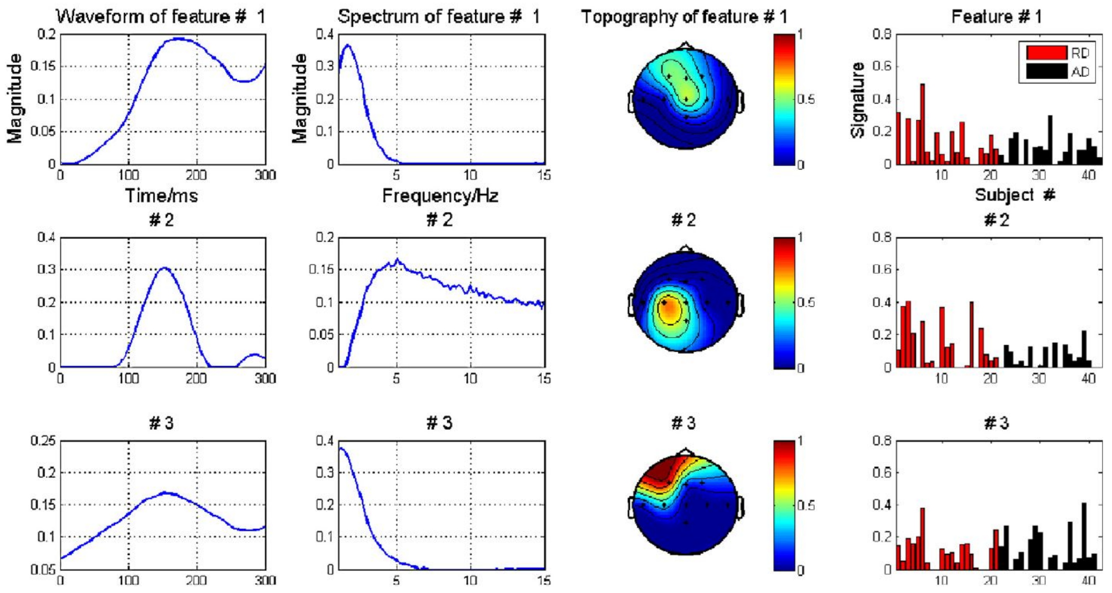

Figure 4.1: Example of CPD of a fourth-order EEG tensor. (image taken from [CLK$^+$15])

4.2 Examples of Different Applications of CPD in EEG-related Research

Many state-of-the-art brain source localization algorithms using EEG measurements face problems when the signal-to-noise ratio is low and when there are multiple active brain regions simultaneously. [BAC+14] proposed a solution for this. The spatial-temporal EEG data is first transformed into spatial-temporal-wave-vector(STWV) tensor using local spatial Fourier transformation. Then CP decomposition is then applied on to the STWV tensor as a preprocessing step before using the decomposed components as input to the proposed DA(Disk Algorithm) source localization algorithm. This paper concluded that the tensor decomposition preprocessing contributes to improved accuracy compared to conventional methods.

[AABB+07] uses tensor decomposition in multiple ways to localize epileptic focus or the origin of a seizure. The ictal EEG signals of seizure samples are transformed into spatial-temporal-frequency epilepsy tensor using continuous wavelet transform. Then by modeling the tensor using a PARAFAC model, the seizure origins are localized based on the spatial signature of the seizure extracted by the PARAFAC model and the spatial signature identified by a neurologist. Apart from that, the paper also proposed a method to extract artifacts using PARAFAC decompositon onto the spatial-scale-temporal tensor of the data. Furthermore, the study also manage to perform artifact removal by means of multi-linear-subspace analysis onto the Tucker-decomposed spatial-scale-temporal tensor.

[DVVDL+07] similarly uses wavelet transformation onto EEG data of seizure activity to produce spatial-temporal-scales tensor. After the CP decomposition, the epileptic components are determined according to the variances of the components, the component with the highest contribution of variance in spatial mode is determined to be the epileptic component and the its spatial mode will give the spatial distribution of the epileptical activity.

[WLL+12] transformed the somatosensory-evoked EEG data of chronic pain and pain-free subjects into spatial-temporal-frequency tensor through continuous wavelet transformation. By analyzing the CP-decomposition of the tensor, the paper reported comparisons of the behaviour of the signals in time, frequency and space domains between chronic pain and pain-free subjects.

[ABB+07] performed CPD on spacial-time-statistical feature tensors of EEG data to evaluate the significance of each proposed statistical feature in seizure recognition. Instead of the usual frequency mode, this paper proposed using statistical features like spectral entropy, spectral skewness, Hjorth's activity and others as the third mode of the tensor to perform seizure recognition. On top of the claimed increased recognition accuracy, this approach also gives insights about the significance of each of the features for the recognition and offers a potential way of doing feature analysis for seizure recognition.

Chapter 5

Tensor Library

Tensor is a common concept in the field of signal processing, machine learning and graphics processing because it provides a fundamental way to index the data over multiple dimensions. As a result, there are many machine learning and signal processing softwares or libraries which support **basic tensor operations**, such as tensor addition and multiplication, tensor slicing, tensor shape manipulation and element-wise mathematical operations. However, there are not many libraries which readily provides functions for **tensor algebra** such as different tensor products(Khatri-Rao), different types of tensor decompositions, multi-linear rank approximation and tensor factorization. A summary of different tensor libraries and their corresponding characteristics is tabulated in Table 5.1. Most of them implicitly support GPU computation.

On the other side of the tensor size spectrum, when the size of the tensor is beyond the storage of a single machine, then the decomposition will have to be distributed across multiple machines. [PFS12] and [PFM$^+$14] are implementations of parallel, distributed tensor decomposition using Matlab. Being one of the most famous big data analytics platform Hadoop is a great platform to perform distributed tensor decomposition. Following the MapReduce paradigm, [KPHF12] and [BTK$^+$14] proposed methods for tensor decomposition on Hadoop. [XCH$^+$10], on the other hand, proposed a tensor factorization library for GraphLab, another distributed computing platform.

Libraries	**Basic Operations**	**Tensor Algebra**	**Language**	**Licence**	**Support**	**Additional Remarks**
Torch	y	n	LuaJit	open source BSD	-well documented -development community	Popular deep learning platform.
Tensor Flow	y	n	- C++ - Python	open source Apache 2.0	-well documented -development community	Popular deep learning platform.
Theano	y	n	- Python	open source	-well documented -development community	popular deep learning platform.
Matlab Tensor Toolbox easy to use/istall	y	y	-Matlab	free licence for researchers available	-well documented -development community	Most popular in tensor research
Matlab Tensorlab easy to use/istall	y	y	Matlab	free licence for researchers available	-well documented -development community	Extensive Tensor Algebra Library
Dynare++ Tensor library	y	n	-Matlab -Octave	open source GNUGPL	-well documented	Primarily used in Dynamic Stochastic General Equilibrium modelling
Ftensor	y	n	-C++		-documented	only proides C++ template for tensor objects
SPLATT easy to use/istall	y	y	-Matlab -Octave -C/C++	open source MIT Licence	-documented	
R: PTAK easy to use/istall	y	y	-R	open source, GPL-2/3	-well documented -development community	Similar to Matlab Tensor Toolbox .
CTF	y	n	-C++	open source	-well documented	Enables explicit control onto the distributed implementation.
vmmlib	y	n	-C++		-documented	Specially designed to support graphics and visualization applications.

Table 5.1: Tensor Libraries

Chapter 6

Conclusion

In general, this report has collected and summarized information regarding the application of tensor decomposition on multi-modal big data. First of all, it briefly introduced the concept and the purpose of tensor in the manipulation of multi-modal data. Then, it concisely presented the mathematical fundamentals of tensor decomposition and the corresponding ALS algorithm. It has revealed aspects, of which one can take advantage, to parallelize the computation. Finally, it summarized brief review on the applications of tensor decomposition in EEG-related research and organized a compilation of existing software libraries for tensor manipulation and decomposition.

Tensor is an important way of representing multi-modal data but currently it is still generally only viewed as an multi-way array object and it is mostly only appreciated for the convenient indexing representation. This is equivalent to viewing matrix simply as a two-way array object and completely disregarding the potential of matrix algebra. However, following the recent rapid development of big data analytics, data scientists are acknowledging the potential of tensor algebra, which leads to the recent sudden rise in the exposure of tensor algebra, in particular tensor decomposition, to the field.

Nonetheless, tensor algebra still remains scarcely-exposed to data scientist because of several reasons. First of all, like matrix algebra, it has a domain of basic properties and basic operations which are never revealed in bachelor level mathematics. Secondly, tensor decomposition is NP-hard, which means, there may not be a solution, and even if there is, it may not be unique. It usually requires application specific experience to know how to parameterize the decomposition which can only be accumulated through more research.

Still, scientist in many different fields are beginning to explore the potential of tensor decomposition in search for ways to extract higher-level or hidden information from multi-modal data. This provides a new direction for innovations and also leads to the surfacing of a number of software packages or libraries which provide the functionality of tensor decomposition. Large data size of tensors and high-complexity of tensor decomposition also lead to the development of software for distributed tensor processing.

In conclusion, tensor is a natural representation form for multi-modal data and tensor decomposition is domain of algebra and methods to perform multi-modal analysis on the data. It uncovers the underlying unknown structure of the data but since it is a form of unsupervised learning, it requires deep knowledge in the parameterization and modeling of the decomposition to obtain the desired structures or relationships in the structure. However, it is a definitely a powerful tool for multi-modal dimension reduction and feature analysis.

Appendix A

ALS Algorithm for CP and Tucker

A.1 ALS Algorithm for CP decompositions

The formal cost function for the approximation of this decomposition is defined as :

$$\underset{A,B,C,\Lambda}{\operatorname{argmin}} \|X - (B \otimes A)\Lambda C^{\intercal}\|_F^2$$

, where $diag(\lambda) = \Lambda \in \mathbb{R}^R$ and this cost function can be reformulated into an ALS[1] problem shown in Algorithm A.1

[1]Alternating Least Square

Algorithm 1 ALS Algorithm for CP (see Figure 3.2 for illustration)

```
1 Initialize A, B, C and $\Lambda$ (randomly).
2 Compute $X^{opt}$ by $vec(X^{opt}) = (C \odot B \odot A)\Lambda$ using fixed A, B, C and $\Lambda$.
3 Compute A = $\underset{A}{\operatorname{argmin}}\|X_a^{opt} - X_a\|_F^2 = X_a^{opt}(C \odot B)(C^\intercal C * B^\intercal B)^+\Lambda^{-1}$, where
  $X_a = A\Lambda(C \odot B)^\intercal$, using fixed B, C and $\Lambda$.
4 Compute B = $\underset{B}{\operatorname{argmin}}\|X_b^{opt} - X_b\|_F^2 = X_b^{opt}(C \odot A)(C^\intercal C * A^\intercal A)^+\Lambda^{-1}$, where
  $X_b = B\Lambda(C \odot A)^\intercal$, using fixed A, C and $\Lambda$.
5 Compute C = $\underset{C}{\operatorname{argmin}}\|X_c^{opt} - X_c\|_F^2 = X_c^{opt}(B \odot A)(B^\intercal B * A^\intercal A)^+\Lambda^{-1}$, where
  $X_c = C\Lambda(B \odot A)^\intercal$, using fixed A, B and $\Lambda$.
6 Compute $\lambda = \{\lambda_i\}_{i=1\ldots R}$, where $\lambda_i = \|\hat{a_i} + \hat{b_i} + \hat{c_i}\|^2$ and $\hat{a_i}, \hat{b_i}, \hat{c_i}$ are the
  normalized columns of A, B, and C.
7 Repeat step 2-6 until convergence.
```

where X_a, X_b and X_c are the slab matrices of the tensor X of the corresponding modes, $*$ denotes the Hadamard product, Z^+ denotes the pseudo inverse of Z, and $\odot$ denotes the Khatri-Rao tensor products.

A.2 ALS Algorithm for CP and Tucker3 decompositions

The formal cost function for the approximation of this decomposition is defined as :

$$\underset{A,B,C,G}{\operatorname{argmin}}\|X - (B \otimes A)GC^\intercal\|_F^2$$

Optimizing this cost function requires the alternating maximization of :

$$A, B, C = \underset{A,B,C}{\operatorname{argmax}} \left[Y = \sum_i^I \sum_j^J \sum_k^K X_{i,j,k} a_i \otimes b_j \otimes c_k \right]$$

of which the solution is obtained via SVD of the slab matrix of Y of different modes.

Algorithm 2 ALS Algorithm for Tucker3

```
1 Initialize A, B, C and G (randomly).
2 Compute $X^{opt}$ by $vec(X^{opt}) = (C \odot B \odot A)vec(G)$ using fixed A, B, C.
3 Compute A = R-leading singular vectors of $Y_a$, where $Y_a = X_a^{opt}(C \otimes B)^\intercal$, using fixed B, C and $X^{opt}$.
4 Compute B = S-leading singular vectors of $Y_b$, where $Y_b = X_b^{opt}(C \otimes A)^\intercal$, using fixed A, C and $X^{opt}$.
5 Compute C = T-leading singular vectors of $Y_c$, where $Y_c = X_c^{opt}(B \otimes A)^\intercal$, using fixed A, B and $X^{opt}$.
6 Repeat step 2-5 until convergence.
7 Compute G = $X \times_1 A \times_2 B \times_3 C$
```

where X_a, X_b and X_c be the slab matrix of the tensor X in the corresponding dimensions, $\odot$ denotes the Khatri-Rao tensor product, $\otimes$ denotes the matrix Kronecker product, and $\times_n$ denotes the tensor n-mode product.

List of Figures

Bibliography

[AABB$^+$07] Evrim Acar, Canan Aykut-Bingol, Haluk Bingol, Rasmus Bro, and Bülent Yener. Multiway analysis of epilepsy tensors. *Bioinformatics*, 23(13):i10–i18, 2007.

[ABB$^+$07] Evrim Acar, Canan Aykut Bingol, Haluk Bingol, Rasmus Bro, and Bulent Yener. Seizure recognition on epilepsy feature tensor. In *Engineering in Medicine and Biology Society, 2007. EMBS 2007. 29th Annual International Conference of the IEEE*, pages 4273–4276. IEEE, 2007.

[BAC$^+$14] Hanna Becker, Laurent Albera, Pierre Comon, Martin Haardt, Gwénaël Birot, Fabrice Wendling, Martine Gavaret, Christian-George Bénar, and Isabelle Merlet. Eeg extended source localization: tensor-based vs. conventional methods. *NeuroImage*, 96:143–157, 2014.

[BTK$^+$14] Alex Beutel, Partha Pratim Talukdar, Abhimanu Kumar, Christos Faloutsos, Evangelos E Papalexakis, and Eric P Xing. Flexifact: Scalable flexible factorization of coupled tensors on hadoop. In *SDM*, pages 109–117. SIAM, 2014.

[Cic13] Andrzej Cichocki. Tensor decompositions: a new concept in brain data analysis? *arXiv preprint arXiv:1305.0395*, 2013.

[CLK$^+$15] Fengyu Cong, Qiu-Hua Lin, Li-Dan Kuang, Xiao-Feng Gong, Piia Astikainen, and Tapani Ristaniemi. Tensor decomposition of eeg signals: a brief review. *Journal of neuroscience methods*, 248:59–69, 2015.

[DL08] Lieven De Lathauwer. Decompositions of a higher-order tensor in block terms-part ii: Definitions and uniqueness. *SIAM Journal on Matrix Analysis and Applications*, 30(3):1033–1066, 2008.

[DLN08] Lieven De Lathauwer and Dimitri Nion. Decompositions of a higher-order tensor in block terms-part iii: Alternating least squares algorithms. *SIAM journal on Matrix Analysis and Applications*, 30(3):1067–1083, 2008.

[Dur] Ramani Duraiswami. Lecture 5: Matrix-matrix product. URL: `http://www.umiacs.umd.edu/~ramani/cmsc828e_gpusci/lecture5.pdf`.

[DVVDL$^+$07] Maarten De Vos, A Vergult, Lieven De Lathauwer, Wim De Clercq, Sabine Van Huffel, Patrick Dupont, Andre Palmini, and Wim Van Paesschen. Canonical decomposition of ictal scalp eeg reliably detects the seizure onset zone. *NeuroImage*, 37(3):844–854, 2007.

[FKS07] Christos Faloutsos, Tamara G Kolda, and Jimeng Sun. Mining large time-evolving data using matrix and tensor tools. In *ICDM Conference*, volume 565, 2007.

[FSH04] Kayvon Fatahalian, Jeremy Sugerman, and Pat Hanrahan. Understanding the efficiency of gpu algorithms for matrix-matrix multiplication. In *Proceedings of the ACM SIGGRAPH/EUROGRAPHICS conference on Graphics hardware*, pages 133–137. ACM, 2004.

[KB09] Tamara G Kolda and Brett W Bader. Tensor decompositions and applications. *SIAM review*, 51(3):455–500, 2009.

[KPHF12] U Kang, Evangelos Papalexakis, Abhay Harpale, and Christos Faloutsos. Gigatensor: scaling tensor analysis up by 100 times-algorithms and discoveries. In *Proceedings of the 18th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 316–324. ACM, 2012.

[Kre13] J Kreutz. Dgemm-tiled matrix multiplication with cuda. *Jülich Forchungszentrum*, 2013.

[KWm12] David B Kirk and W Hwu Wen-mei. *Programming massively parallel processors: a hands-on approach.* Newnes, 2012.

[PFM$^+$14] Evangelos E Papalexakis, Christos Faloutsos, Tom M Mitchell, Partha Pratim Talukdar, Nicholas D Sidiropoulos, and Brian Murphy. Turbo-smt: Accelerating coupled sparse matrix-tensor factorizations by 200x. In *SDM*, pages 118–126. SIAM, 2014.

[PFS12] Evangelos E Papalexakis, Christos Faloutsos, and Nicholas D Sidiropoulos. Parcube: Sparse parallelizable tensor decompositions. In *Machine Learning and Knowledge Discovery in Databases*, pages 521–536. Springer, 2012.

[SK12] Nicholas D Sidiropoulos and Anastasios Kyrillidis. Multi-way compressed sensing for sparse low-rank tensors. *IEEE Signal Processing Letters*, 19(11):757–760, 2012.

[SP14] Nicholas Sidiropoulos and Evangelos Papalexakis. Icassp 2014 tutorial: Factoring tensors in the cloud: A tutorial on big tensor data analytics, May 2014. URL: `http://www.cs.cmu.edu/~epapalex/tutorials/icassp14.html`.

[TCF16] Wei Tan, Liangliang Cao, and Liana Fong. Faster and cheaper: Parallelizing large-scale matrix factorization on gpus. In *Proceedings of the 25th ACM International Symposium on High-Performance Parallel and Distributed Computing*, pages 219–230. ACM, 2016.

[WLL$^+$12] Juan Wang, Xiaoli Li, Chengbiao Lu, Logan J Voss, John PM Barnard, and Jamie W Sleigh. Characteristics of evoked potential multiple eeg recordings in patients with chronic pain by means of parallel factor analysis. *Computational and mathematical methods in medicine*, 2012, 2012.

[XCH$^+$10] Liang Xiong, Xi Chen, Tzu-Kuo Huang, Jeff G Schneider, and Jaime G Carbonell. Temporal collaborative filtering with bayesian probabilistic tensor factorization. In *SDM*, volume 10, pages 211–222. SIAM, 2010.

License