Randomized Algorithms for Canonical Polyadic Decomposition

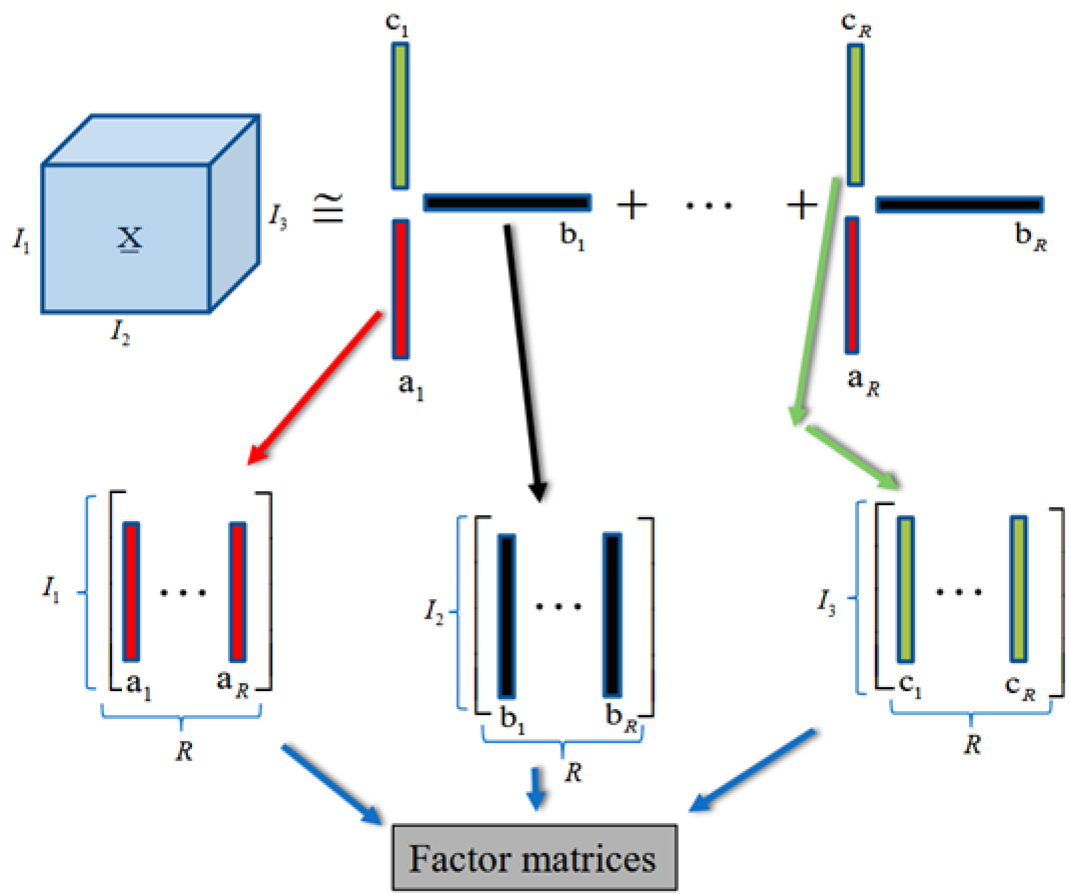
Salman Ahmadi-Asl

December 2019

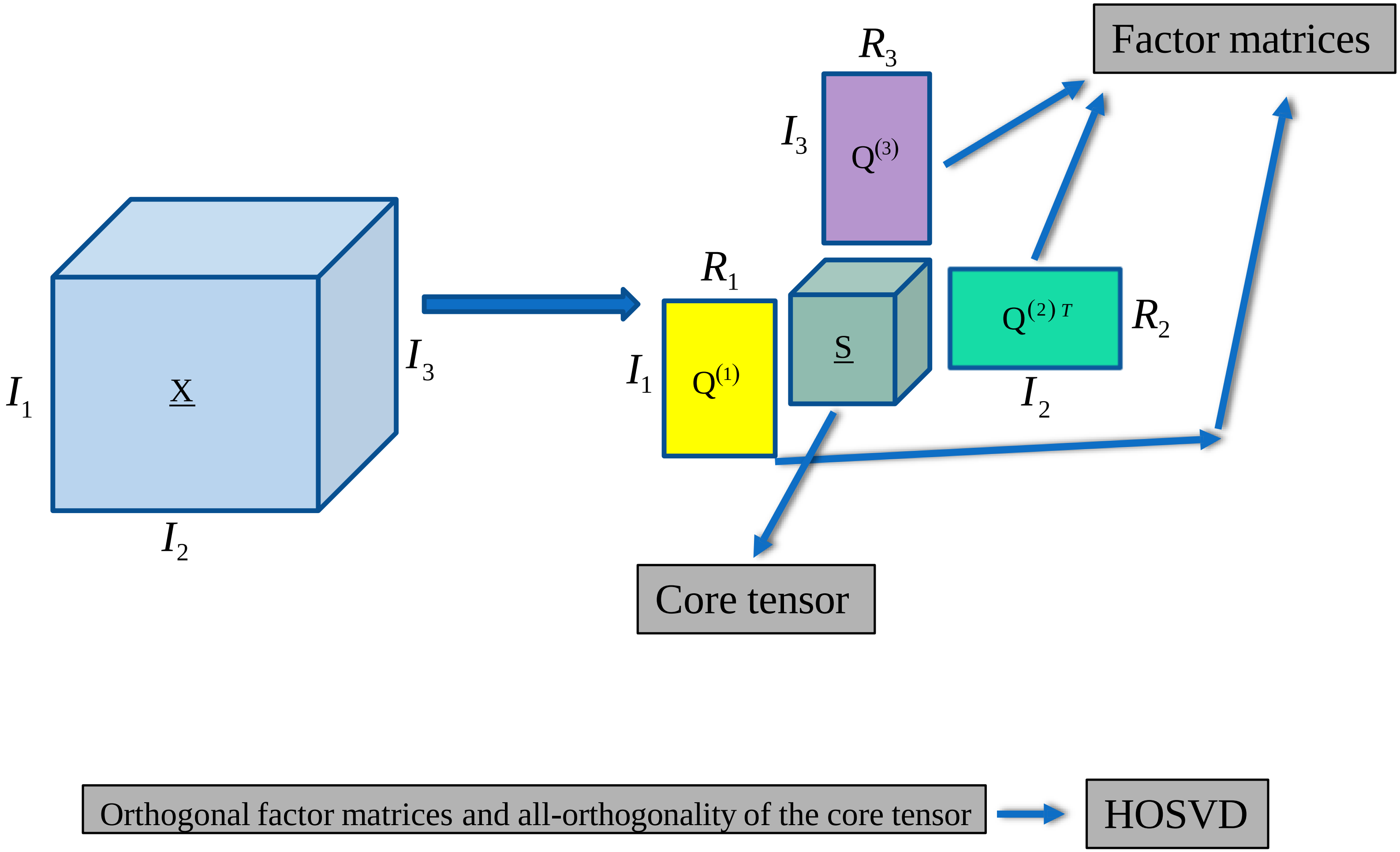
Outline

- **Canonical polyadic decomposition (CPD)**
- **Randomized algorithms for CPD**
  - Randomized CPD with a prior reduction in the Tucker format
  - Randomized block sampling algorithm
  - Randomized ALS algorithm
- **Randomized Parafac2**
- **Simulations**

Canonical Polyadic Decomposition (CPD)



Tucker decomposition



Algorithms for CPD

- Alternating Least Squares (ALS) algorithm
- Gauss-Newton and Levenberg-Marquardt algorithms
- Tensor power iteration algorithm
- Fast damped Gauss-Newton (dGN)
- Simultaneous matrix diagonalization

Alternating least-squares algorithm (ALS)

Fixing all factor matrices and updating one by solving the following least squares problem

$$\underset{\mathrm{A}^{(n)}}{\mathrm{Min}} \left\| Z^{(n)} \mathrm{A}^{(n)T} - \mathrm{X}_{(n)}^{T} \right\|_F, \quad n = 1, 2, \ldots, N$$

$$Z^{(n)} = \mathrm{A}^{(N)} \odot \cdots \odot \mathrm{A}^{(n+1)} \odot \mathrm{A}^{(n-1)} \cdots \odot \mathrm{A}^{(1)}$$

Explicit solution → $\mathrm{A}^{(n)} = \boxed{\mathrm{X}_{(n)} \mathrm{Z}^{(n)}}\, \mathrm{W}^{(n)\dagger}$

$$\mathrm{W}^{(n)} = \mathrm{A}^{(1)T} \mathrm{A}^{(1)} * \ldots * \mathrm{A}^{(n-1)T} \mathrm{A}^{(n-1)} * \mathrm{A}^{(n+1)T} \mathrm{A}^{(n+1)} * \ldots * \mathrm{A}^{(N)T} \mathrm{A}^{(N)}$$

CPD with prior HOSVD compression

Computational complexity

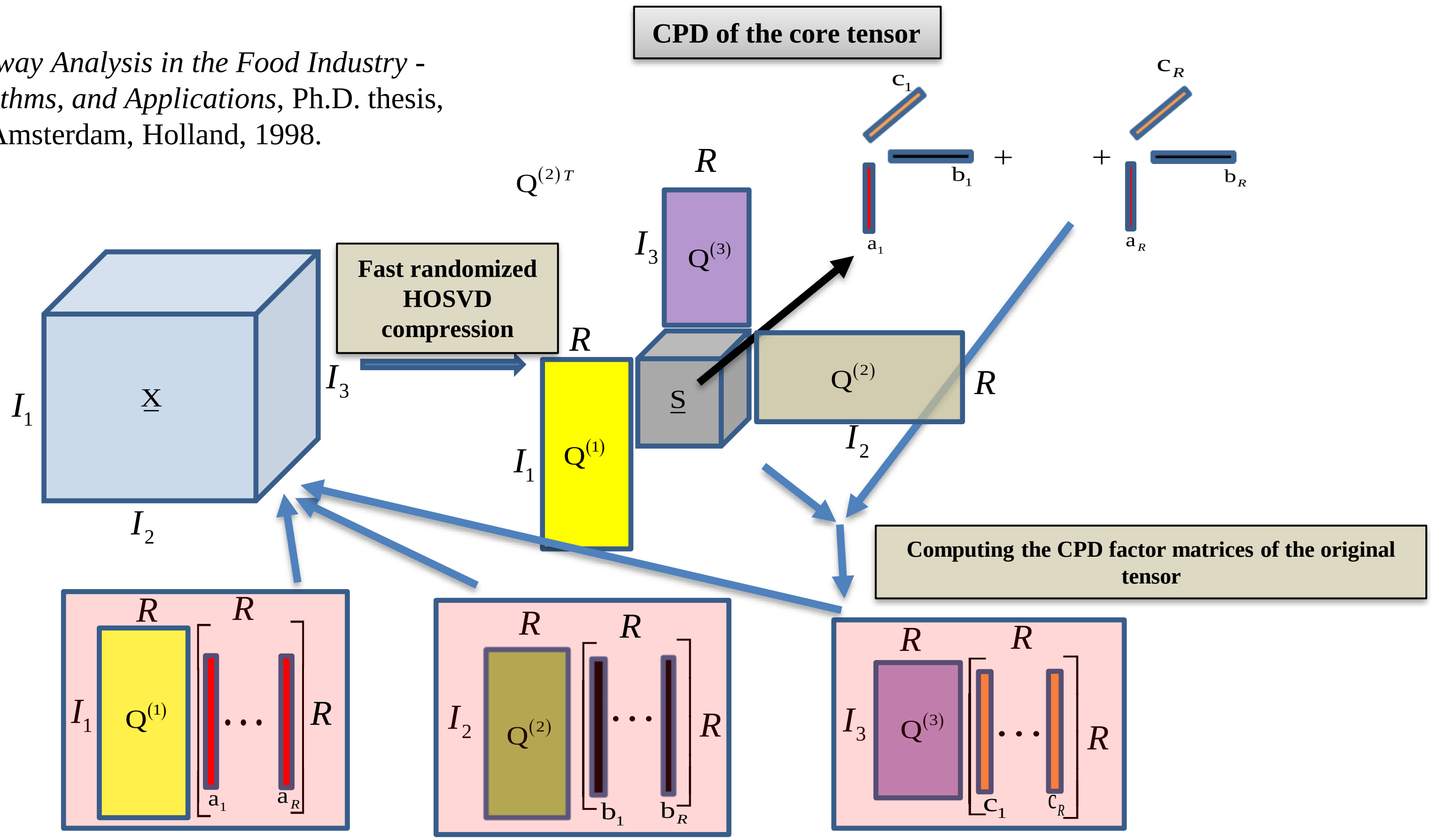
3-th order tensors	$O\left(3R\prod_{i=1}^{3} I_i\right)$	→	$O\left(3R^4\right)$
N-th order tensors	$O\left(NR\prod_{i=1}^{N} I_i\right)$	→	$O\left(NR^{N+1}\right)$

This approach is applicable when a tensor is not of very high order and also the tensor rank is less than the original tensor sizes.

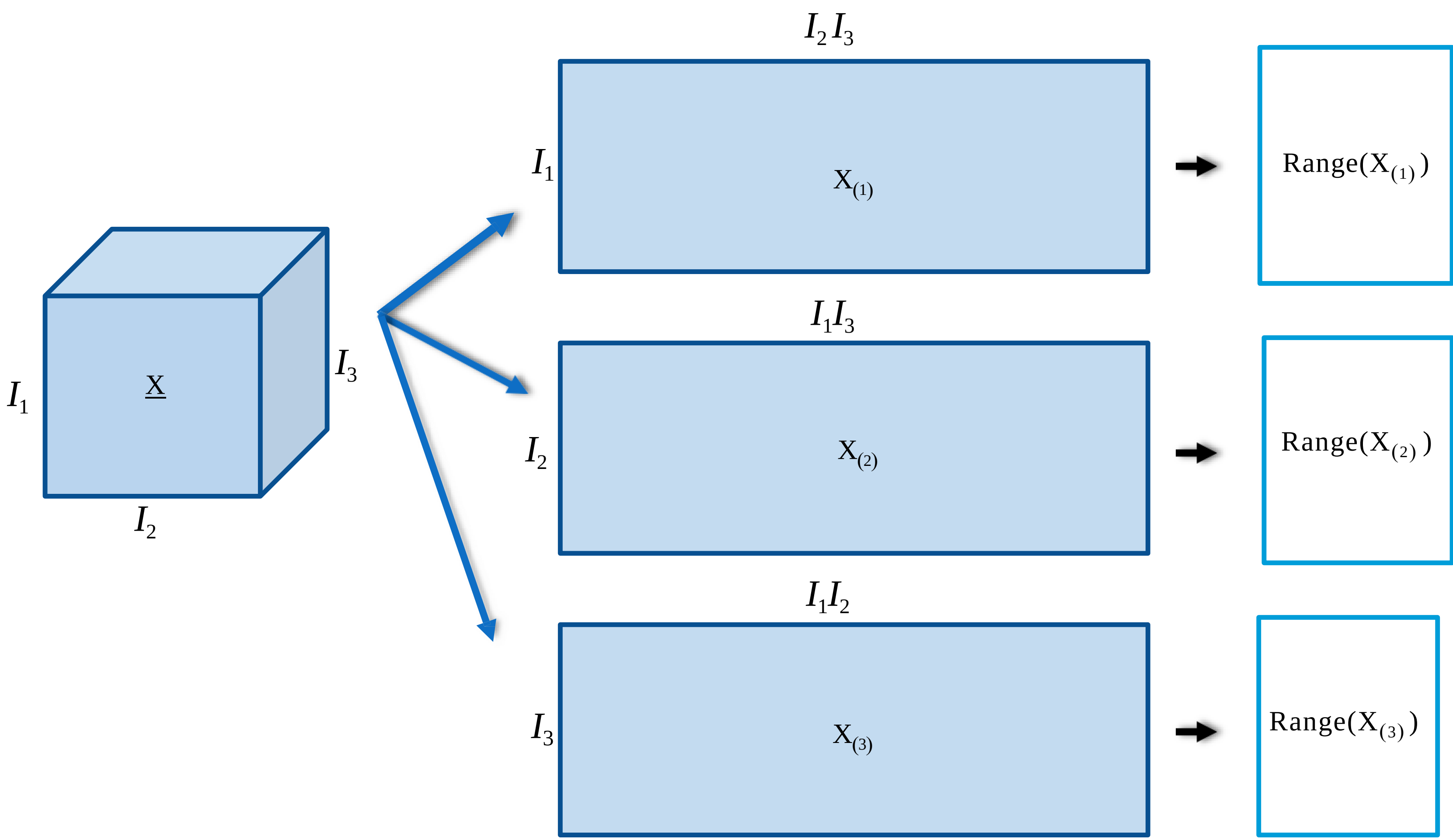
Compression step can be performed by the randomized HOSVD algorithms

CPD with prior HOSVD compression

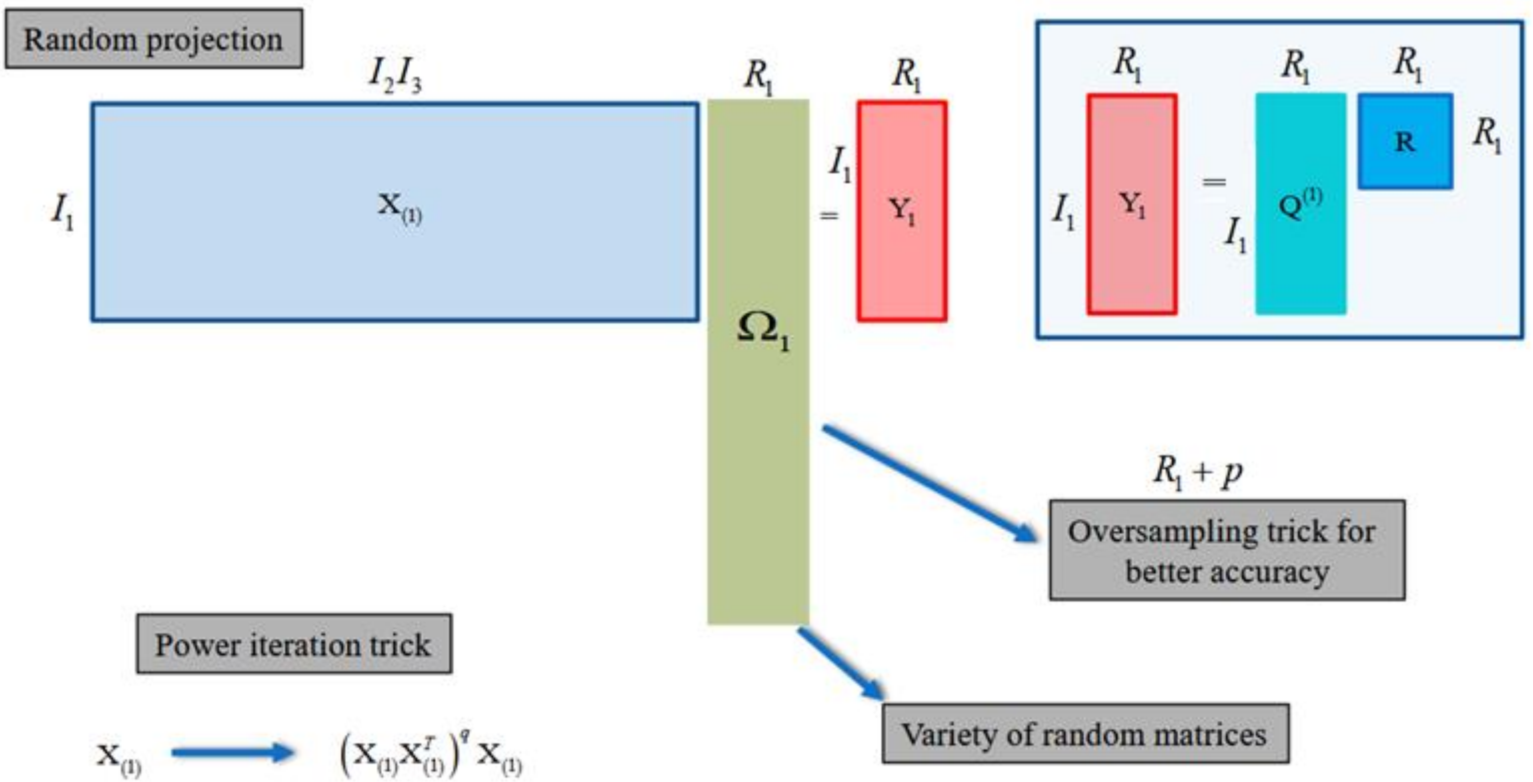
R. Bro, *Multi-way Analysis in the Food Industry - Models, Algorithms, and Applications*, Ph.D. thesis, University of Amsterdam, Holland, 1998.



Randomized algorithms for HOSVD compression



Randomized algorithms for HOSVD compression



Randomized algorithms for HOSVD compression

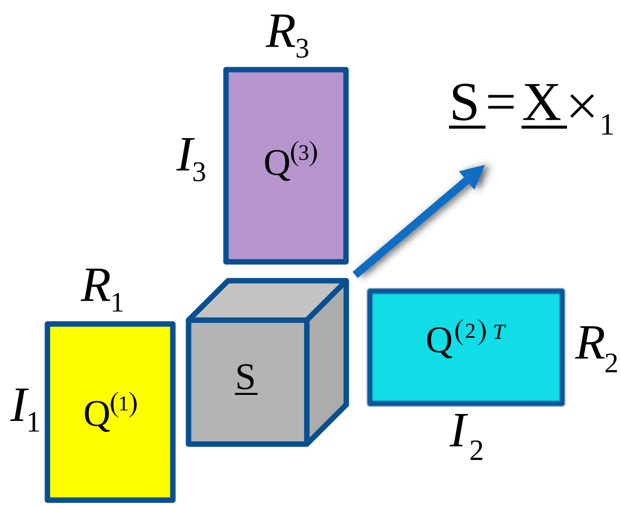
Sparse random matrices

$$\Phi = \begin{cases} -1 & \text{with probability } \frac{1}{6} \\ 0 & \text{with probability } \frac{2}{3} \\ +1 & \text{with probability } \frac{1}{6} \end{cases} \qquad \Phi = \begin{cases} -1 & \text{with probability } \frac{1}{2\sqrt{D}} \\ 0 & \text{with probability } 1-\frac{1}{\sqrt{D}} \\ +1 & \text{with probability } \frac{1}{2\sqrt{D}} \end{cases}$$

where matrix components are i.i.d.

These random matrices are reminiscent the quantization procedure in learning DNNs.

Randomized algorithms for HOSVD compression

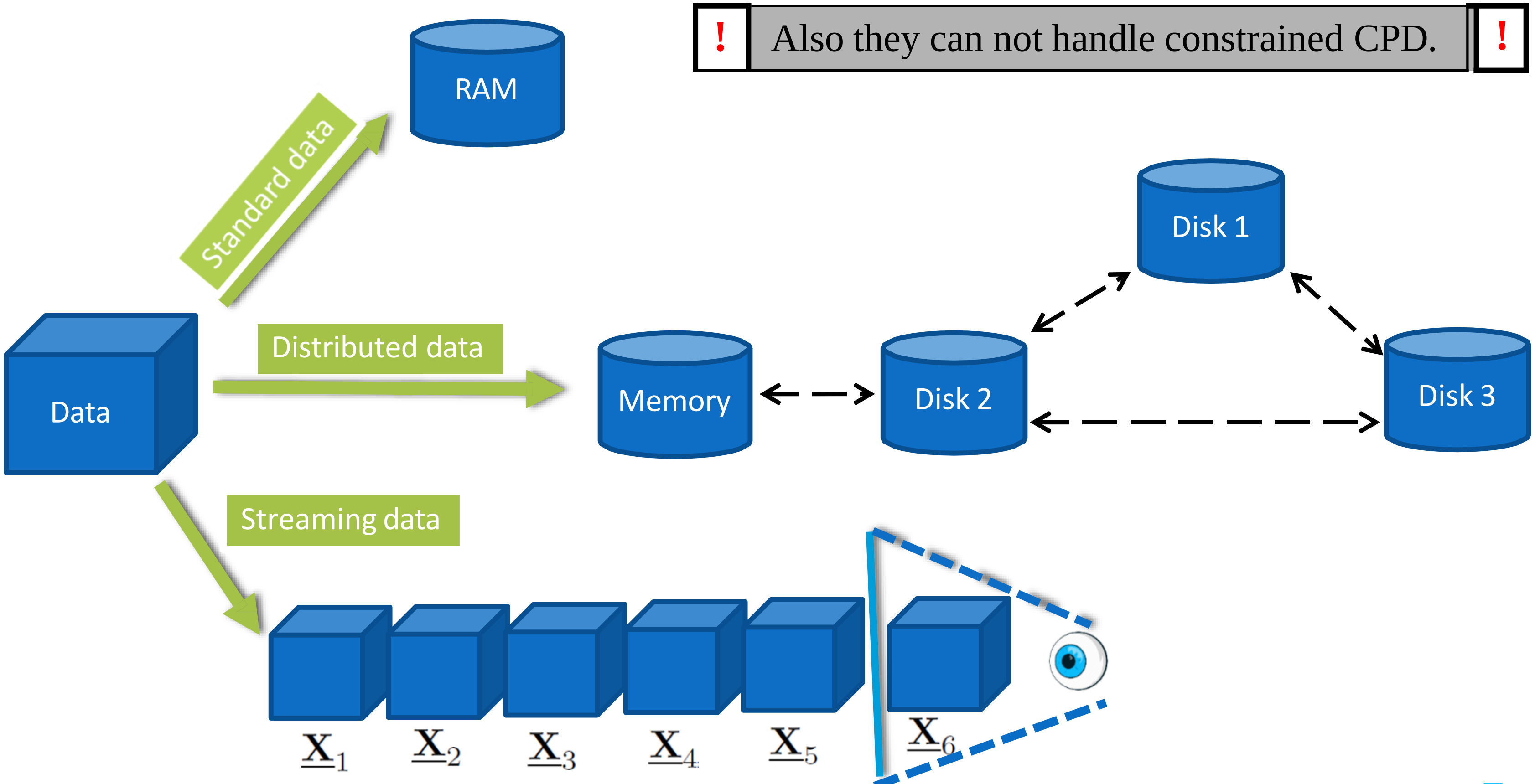


$$\underline{S} = \underline{X} \times_1 Q^{(1)\,T} \times_2 Q^{(2)\,T} \times_3 Q^{(3)\,T}$$

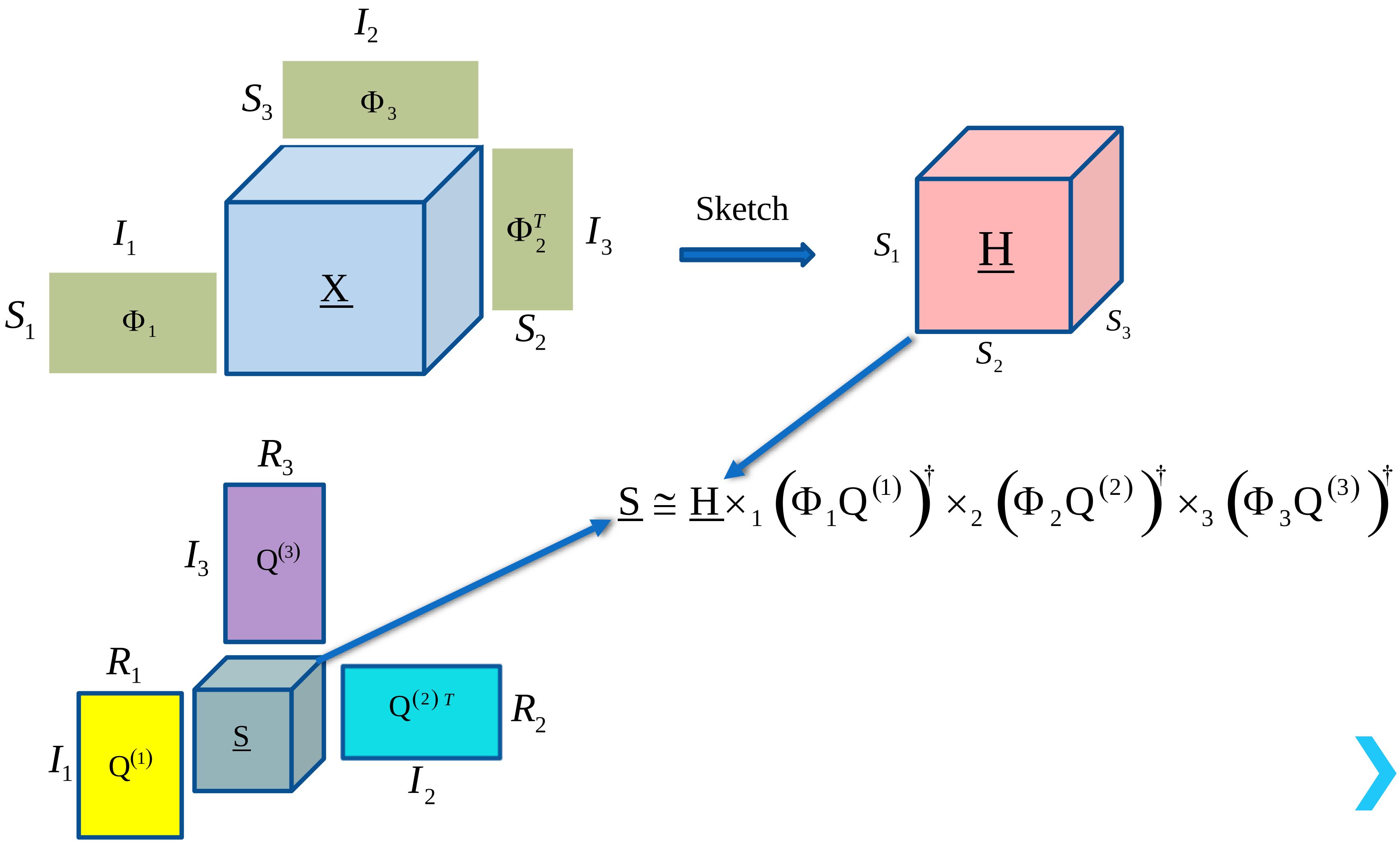
Different types of data

Algorithms presented in this talk are well-suite for first and second ones.

! Also they can not handle constrained CPD. !



Randomized algorithms for HOSVD compression

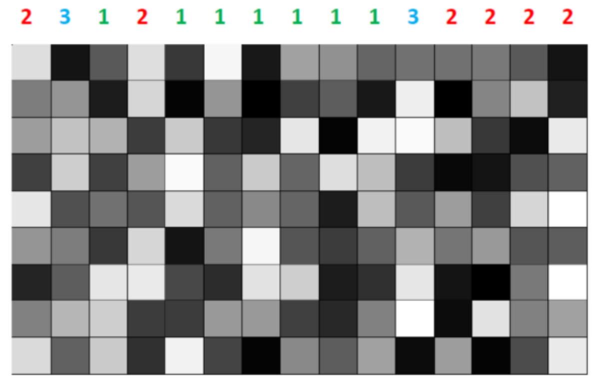


$$\underline{S} \cong \underline{H} \times_1 \left(\Phi_1 Q^{(1)}\right)^{\dagger} \times_2 \left(\Phi_2 Q^{(2)}\right)^{\dagger} \times_3 \left(\Phi_3 Q^{(3)}\right)^{\dagger}$$

Randomized algorithms for HOSVD compression

Count sketch

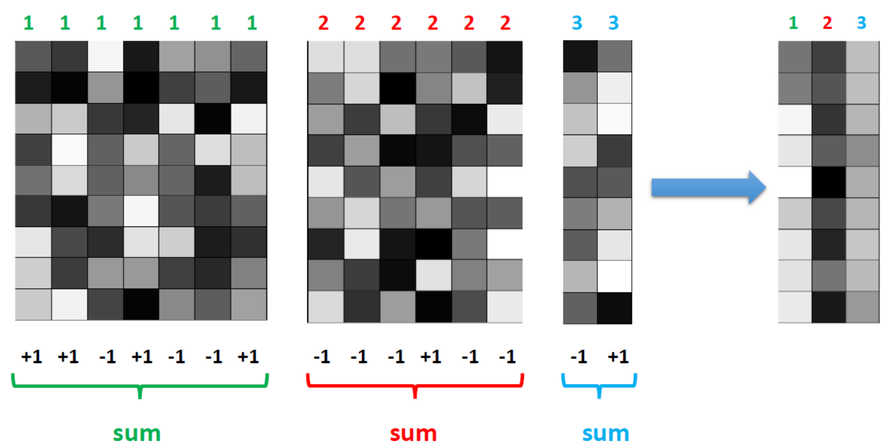
Pictures from:
S. Wang. *A practical guide to randomized matrix computations with MATLAB implementations*. arXiv:1505.07570, 2015.



Example:
- Matrix size 9×15
- Sketch size $s = 3$

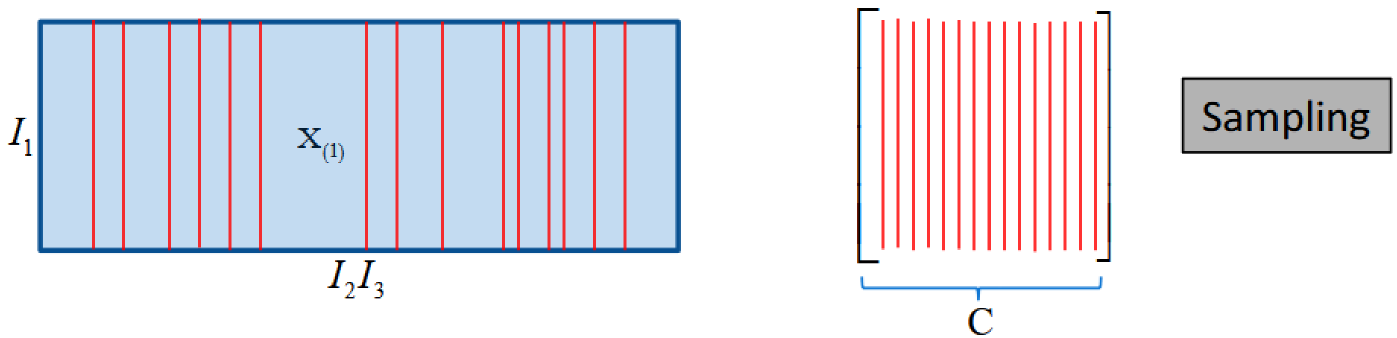
Hashing

(a) Hash each column with a value uniformly sampled from $[s] = \{1, 2, 3\}$.



(b) Flip the sign of each column with probability 50%, and then sum up columns with the same hash value.

Randomized algorithms for HOSVD compression



Sampling columns
- Uniform sampling (with or without replacement)
- Length-squared distribution
- Leverage scores distribution

Randomized algorithms for HOSVD compression

$$\mathbf{X}_{(n)} \in \mathbb{R}^{I_n \times \prod_{m \neq n} I_m}$$

j-th column leverage score

$$l_j = \|\mathbf{V}_R(j,:)\|_2^2, \quad j = 1, 2, \ldots, \prod_{m \neq n} I_m$$

$$\sum_{j=1}^{\prod_{m \neq n} I_m} l_j = R$$

i-th row leverage score

$$l_i = \|\mathbf{U}_R(i,:)\|_2^2, \quad i = 1, 2, \ldots, I_n$$

$$\sum_{j=1}^{I_n} l_i = R$$

Leverage score distribution for column selection

$$p_j = \frac{l_j}{R}, \qquad j = 1, 2, \ldots, \prod_{m \neq n} I_m$$

Maximum of the leverage scores is called coherence of a matrix

Randomized algorithms for HOSVD compression

Fast algorithms for the computations of the leverage scores are proposed in the following papers.

Journal of Machine Learning Research 13 (2012) 3441-3472 Submitted 7/12; Published 12/12

Fast Approximation of Matrix Coherence and Statistical Leverage

Petros Drineas DRINEP@CS.RPI.EDU
Malik Magdon-Ismail MAGDON@CS.RPI.EDU
Department of Computer Science
Rensselaer Polytechnic Institute
Troy, NY 12180

Michael W. Mahoney MMAHONEY@CS.STANFORD.EDU
Department of Mathematics
Stanford University
Stanford, CA 94305

David P. Woodruff DPWOODRU@US.IBM.COM
IBM Almaden Research Center
650 Harry Road
San Jose, CA 95120

Editor: Mehryar Mohri

On Fast Leverage Score Sampling and Optimal Learning

Alessandro Rudi* INRIA – Sierra team, ENS, Paris
Daniele Calandriello* LCSL – IIT & MIT, Genoa, Italy
Luigi Carratino University of Genoa, Genoa, Italy
Lorenzo Rosasco University of Genoa, LCSL – IIT & MIT

Abstract

Leverage score sampling provides an appealing way to perform approximate computations for large matrices. Indeed, it allows to derive faithful approximations with a complexity adapted to the problem at hand. Yet, performing leverage scores sampling is a challenge in its own right requiring further approximations. In this paper, we study the problem of leverage score sampling for positive definite matrices defined by a kernel. Our contribution is twofold. First we provide a novel algorithm for leverage score sampling and second, we exploit the proposed method in statistical learning by deriving a novel solver for kernel ridge regression. Our main technical contribution is showing that the proposed algorithms are currently the most efficient and accurate for these problems.

Randomized algorithms for HOSVD compression

$$\|X - CC^{\dagger}X\| \leq \varepsilon \|X - X_R\|$$

Relative error

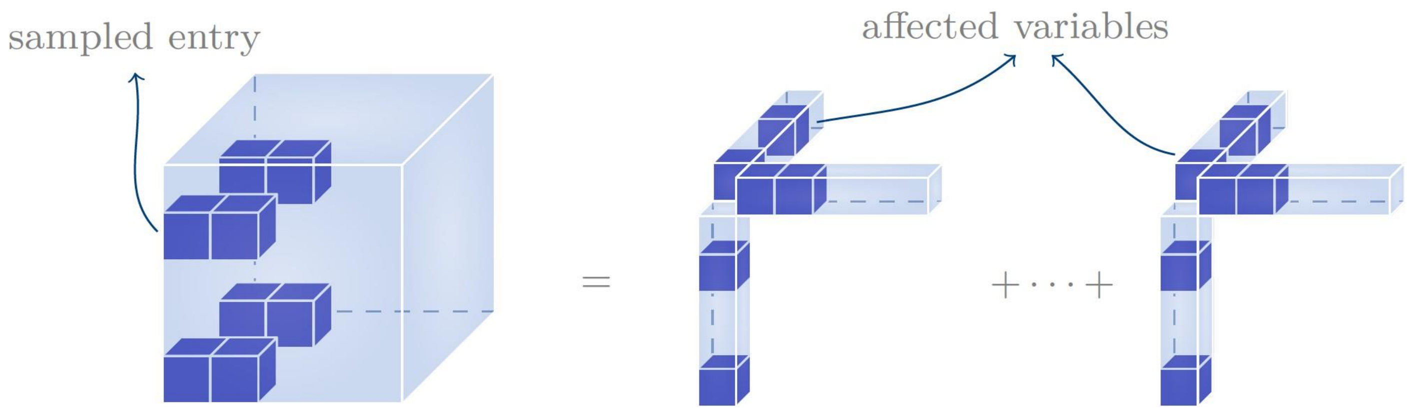
$$\|X - CC^{\dagger}X\| \leq \|X - X_R\| + \varepsilon \|X_R\|$$

Additive error

Randomized rank revealing or equivalently randomized fixed-precision algorithms are applicable when an estimation of matrix rank is unknown and it is approximated adaptively by the underlying algorithms.

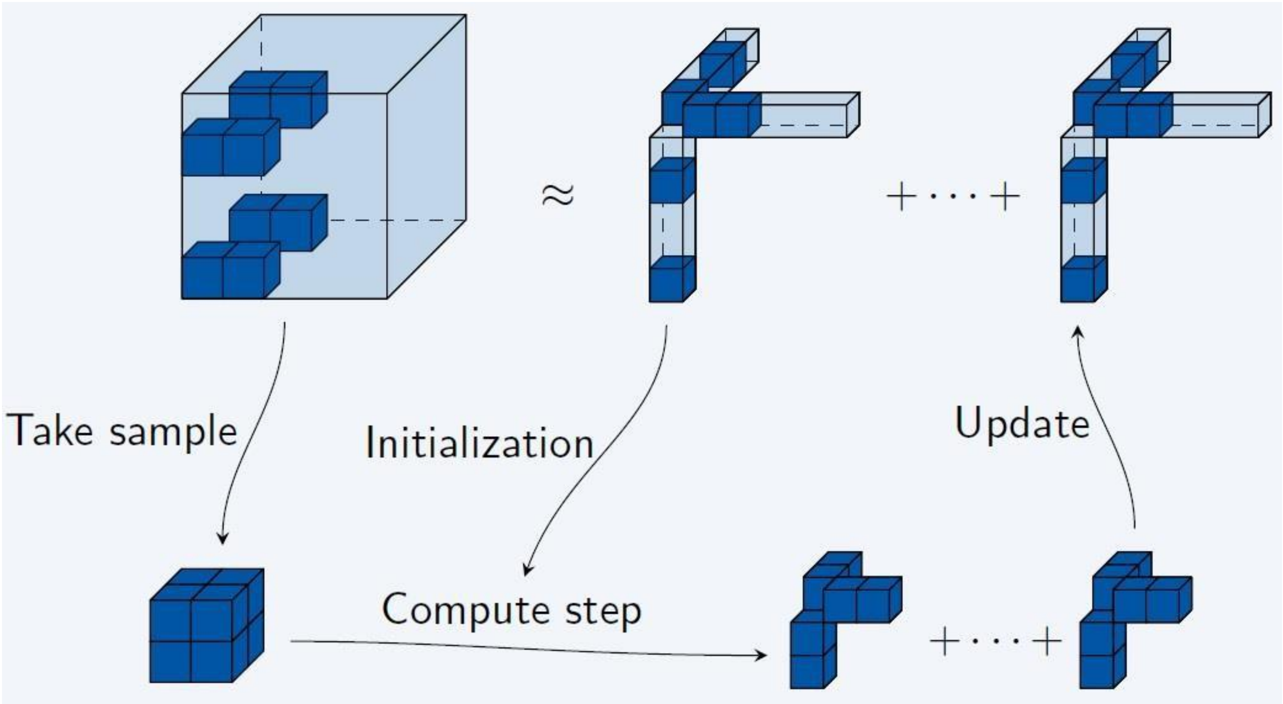
Randomized block sampling algorithm

Randomized block sampling uses the locality property to compute the CPD of a given tensor.



Picture from: N. Vervliet, *Compressed sensing approaches to large-scale tensor decompositions*, PhD thesis, KU Leuven, May 2018.

Randomized block sampling algorithm

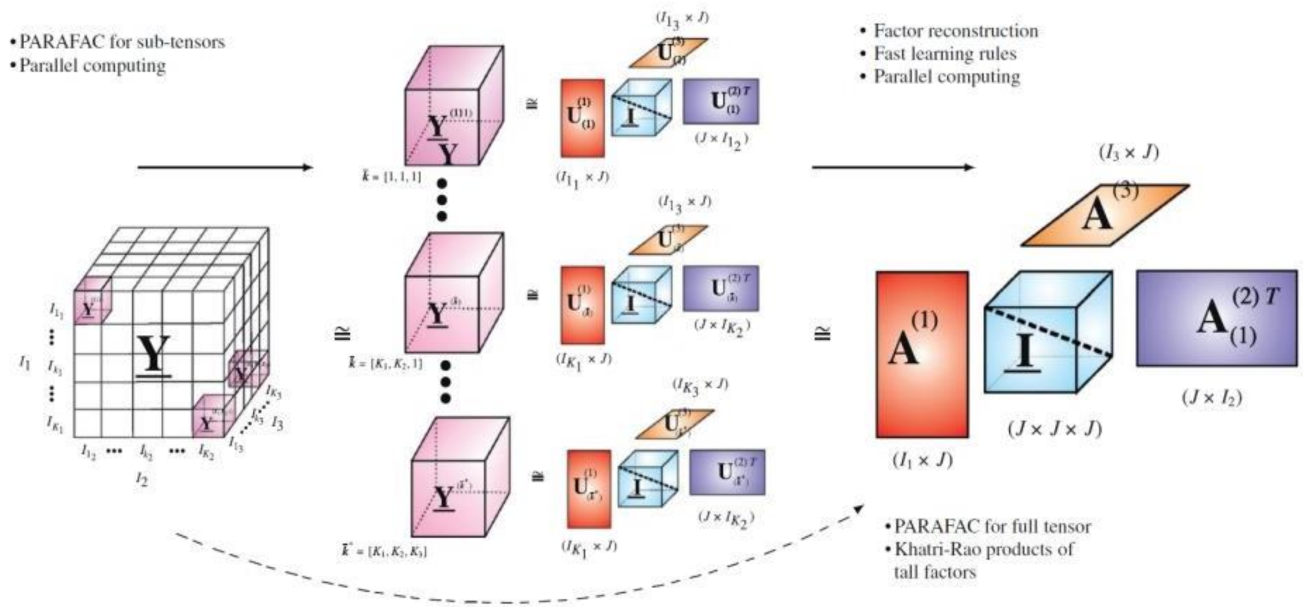


Picture from: N. Vervliet, *Compressed sensing approaches to large-scale tensor decompositions*, PhD thesis, KU Leuven, May 2018.

N. Vervliet, O. Debals, L. Sorber, M. V. Barel, L. D. Lathauwer, Tensorlab 3.0.

Randomized block sampling algorithm

A sampling CPD is also proposing in the following paper

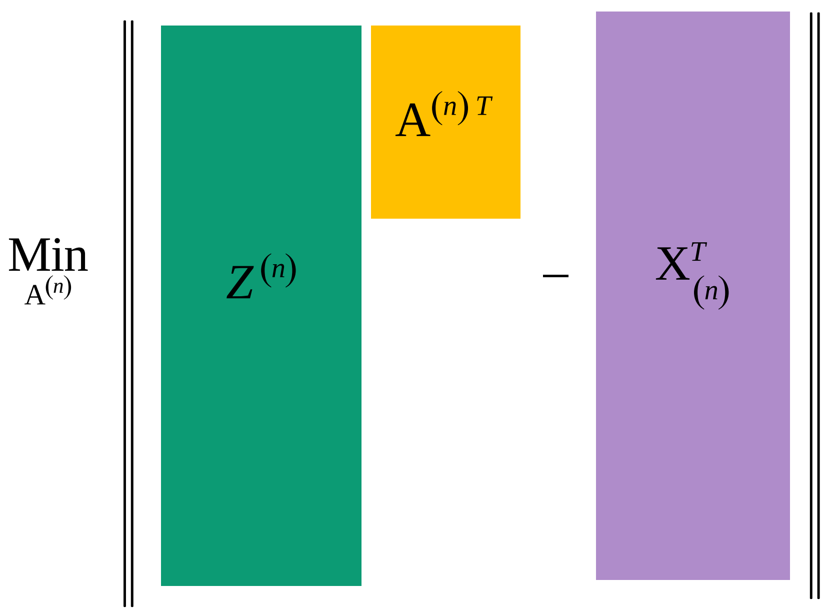


A-H Phan, A Cichocki, *PARAFAC algorithms for large-scale problems*, Neurocomputing, 2011.

Alternating randomized least-squares algorithms

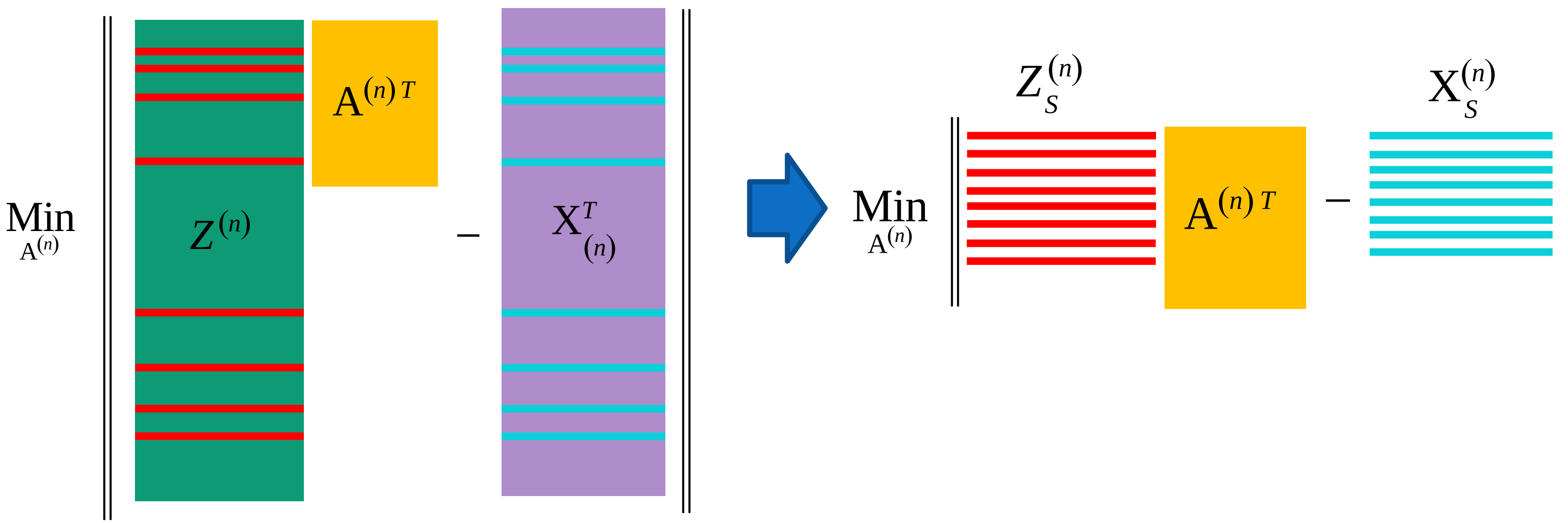
$$\underset{\mathrm{A}}{\mathrm{Min}} \quad \left\| Z^{(n)} \mathrm{A}^{(n)\,T} - \mathrm{X}_{(n)}^{T} \right\|_F, \; n = 1, 2, \ldots, N$$

$$Z^{(n)} = \mathrm{A}^{(N)} \odot \cdots \odot \mathrm{A}^{(n+1)} \odot \mathrm{A}^{(n-1)} \cdots \odot \mathrm{A}^{(1)}$$



$$\mathrm{A}^{(n)} = \mathrm{X}_{(n)} \mathrm{Z}^{(n)} \mathrm{W}^{(n)\,\dagger}$$

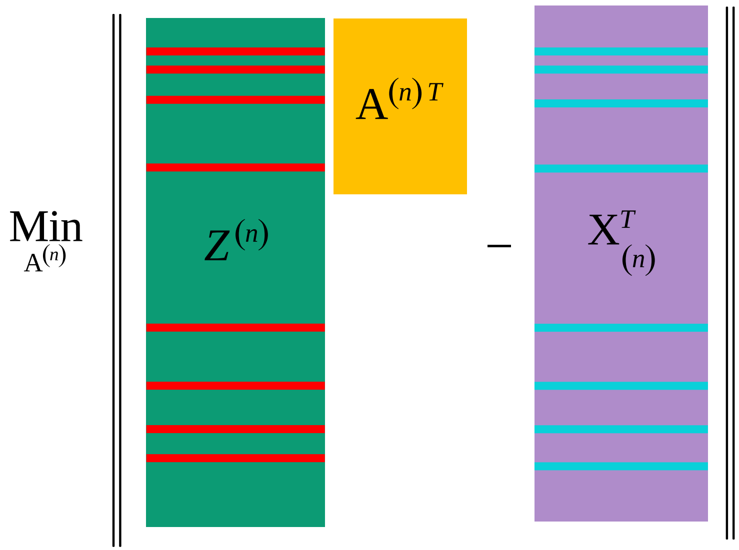
Alternating randomized least-squares algorithms



Coherence of the matrix should be as small as possible

Sampling
- Uniform sampling
- Nonuniform sampling (Leverage scores)

Alternating randomized least-squares algorithms



The row selection does not need computing $Z^{(n)}$ explicitly.

$$Z^{(n)}(j,:) = A^{(1)}(i_1,:) * \cdots * A^{(n-1)}(i_{n-1},:) * A^{(n+1)}(i_{n+1},:) * \cdots * A^{(N)}(i_N,:)$$

$$j = 1 + \sum_{k=1, k\neq n}^{N} (i_k - 1) J_k, \quad J_k = \prod_{m=1, m\neq n}^{N-1} I_m$$

Alternating randomized least-squares algorithms

The nonuniform sampling is used in the following paper.

SPALS: Fast Alternating Least Squares via Implicit Leverage Scores Sampling

Dehua Cheng
University of Southern California
dehua.cheng@usc.edu

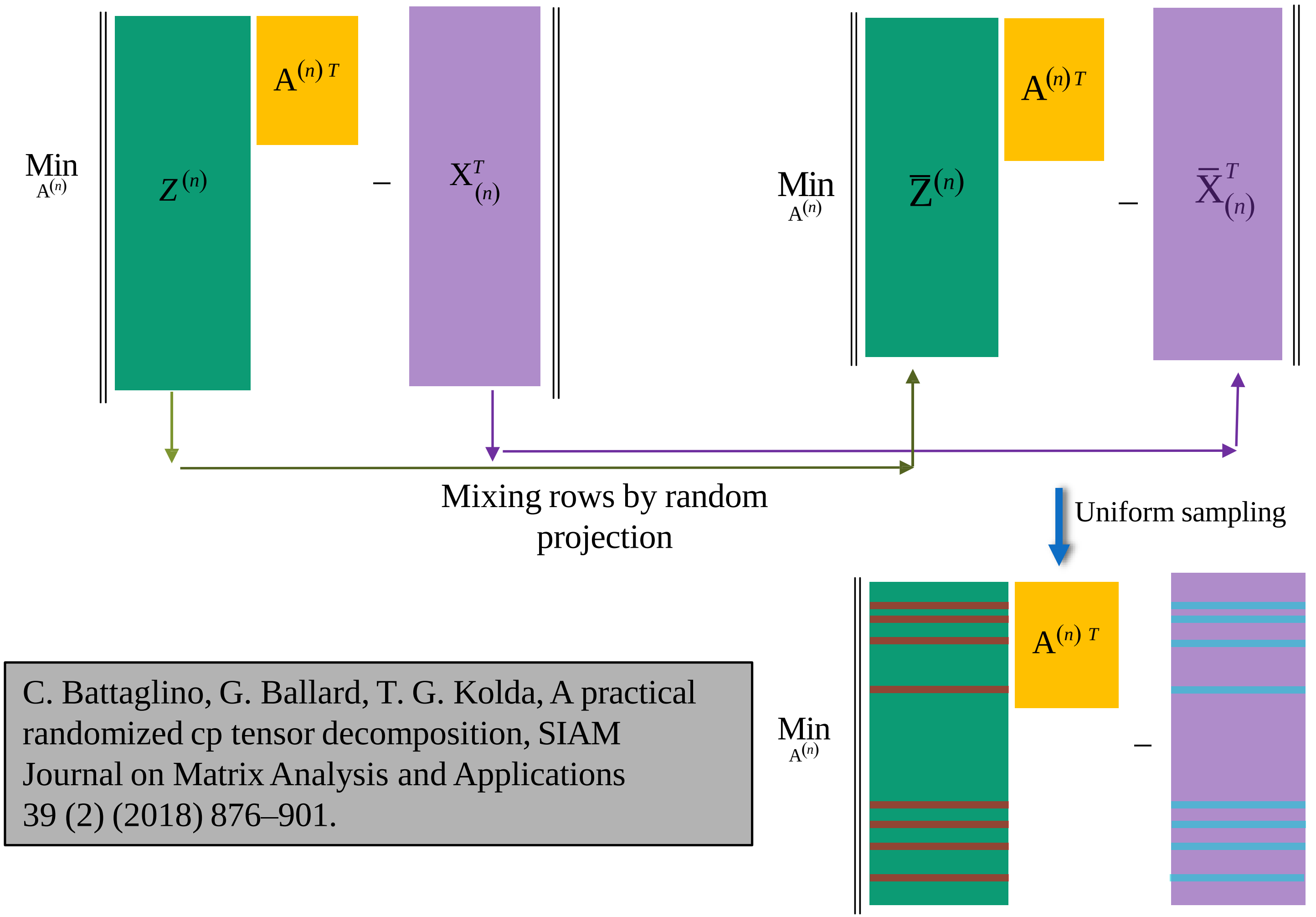
Richard Peng
Georgia Institute of Technology
rpeng@cc.gatech.edu

Ioakeim Perros
Georgia Institute of Technology
perros@gatech.edu

Yan Liu
University of Southern California
yanliu.cs@usc.edu

For both Khatri-Rao and Kroncker product of matrices

Alternating randomized least-squares algorithms



C. Battaglino, G. Ballard, T. G. Kolda, A practical randomized cp tensor decomposition, SIAM Journal on Matrix Analysis and Applications 39 (2) (2018) 876–901.

Alternating randomized least-squares algorithms

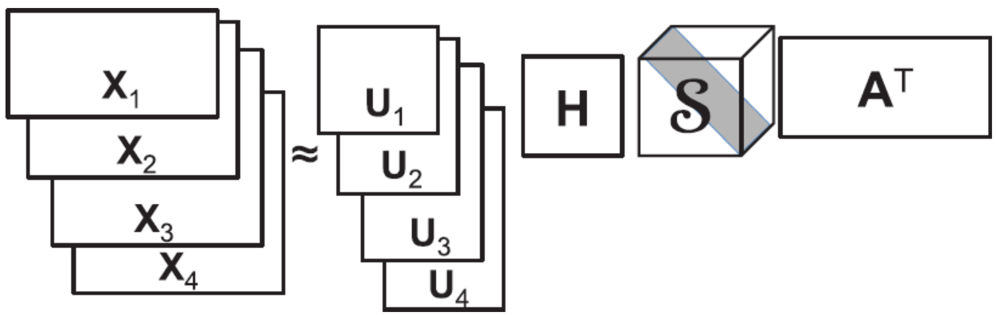
Matlab codes are included in the tensor toolbox (Matlab) and Tensorly (Python)

B. W. Bader, T. G. Kolda, Matlab tensor toolbox version 2.6 (2015).

J. Kossaifi, Y. Panagakis, A. Anandkumar, M. Pantic, Tensorly: *Tensor learning in python*, The Journal of Machine Learning Research 20 (1) (2019) 925–930.

Randomized algorithms for parafac2

Parafac2 model



Picture from: E.E. Papalexakis, C. Faloutsos, N.D. Sidiropoulos, *Tensors for data mining and data fusion: Models, applications, and scalable algorithms*, ACM Trans. on Intelligent Systems and Technology, vol. 8, no. 2, 16:1–16:44, 2016.

Randomized algorithms for parafac2

ALGORITHM 8: ALS Algorithm for PARAFAC2

Input: Multiset $\{\mathbf{X}_k\}$ for $k = 1 : K$ and rank R.
Output: PARAFAC2 Decomposition of $\{\mathbf{X}_k\}$: $\{\mathbf{U}_k\}$, $\mathbf{H}$, $\mathcal{S}$, $\mathbf{V}$.

Initialize:
  $\mathbf{V} \leftarrow R$ principal eigenvectors of $\sum_{k=1}^{K} \mathbf{X}_k^T \mathbf{X}_k$
  $\mathbf{H} \leftarrow \mathbf{I}$
for $k = 1 \cdots K$ **do**
  $\mathcal{S}(:,:,k) \leftarrow \mathbf{I}$, for $k = 1 \cdots K$.
end for
while convergence criterion is not met **do**
  for $k = 1 \cdots K$ **do**
    $[\mathbf{P}_k, \boldsymbol{\Sigma}_k, \mathbf{Q}_k] \leftarrow$ truncated SVD of $\mathbf{H}\mathbf{S}_k\mathbf{V}^T\mathbf{X}_k^T$ at rank R
    $\mathbf{U}_k \leftarrow \mathbf{Q}_k\mathbf{P}_k^T$
  end for
  for $k = 1 \cdots K$ **do**
    Compute $\mathcal{Y}(:,:,k) = \mathbf{U}_k^T \mathbf{X}_k$
  end for
  Run a single iteration of CP ALS (Algorithm 1) on $\mathcal{Y}$ and compute factors $\mathbf{H}, \mathbf{V}, \hat{\mathbf{S}}$.
  for $k = 1 \cdots K$ **do**
    $\mathcal{S}(:,:,k) \leftarrow Diag(\hat{\mathbf{S}}(k,:))$
  end for
end while

A variety of randomized algorithms can be used.

Picture from: E.E. Papalexakis, C. Faloutsos, N.D. Sidiropoulos, *Tensors for data mining and data fusion: Models, applications, and scalable algorithms*, ACM Trans. on Intelligent Systems and Technology, vol. 8, no. 2, 16:1–16:44, 2016.

Simulations

$$\mathbf{A}^{(1)} = \mathrm{randn}\,(500, R)\,, \quad \mathbf{A}^{(2)} = \mathrm{randn}\,(500, R)\,,$$

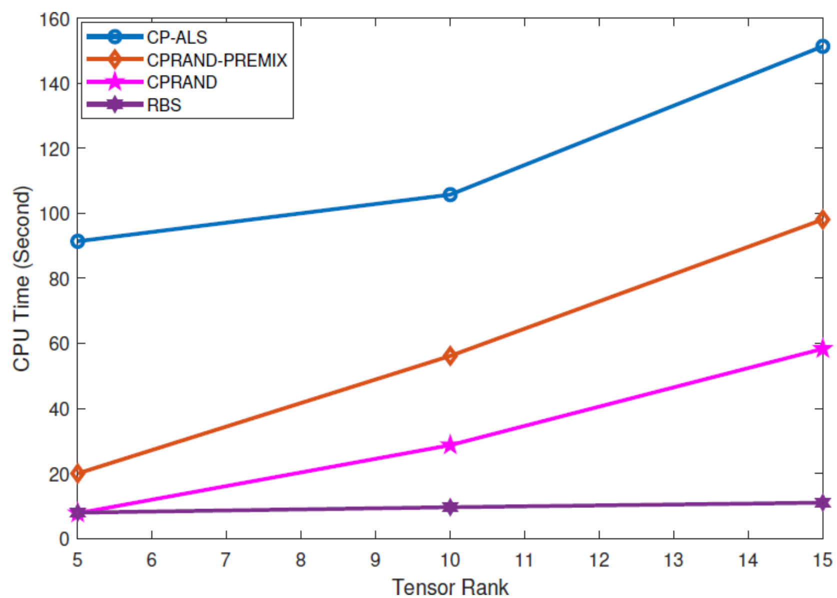
$$\mathbf{A}^{(3)} = \mathrm{randn}\,(500, R)\,.$$

$$\eta = 0.01$$

$$\underline{\mathbf{X}} = [[\mathbf{A}^{(1)}, \mathbf{A}^{(2)}, \mathbf{A}^{(3)}]] \qquad \widehat{\underline{\mathbf{X}}} = \underline{\mathbf{X}} + \eta \left(\frac{\|\underline{\mathbf{X}}\|}{\|\underline{\mathbf{Y}}\|} \right) \underline{\mathbf{Y}},$$

$$f = 1 - \frac{\left\| \widehat{\underline{\mathbf{X}}} - \widetilde{\underline{\mathbf{X}}} \right\|}{\left\| \widehat{\underline{\mathbf{X}}} \right\|}$$

Simulations



Algorithm	$R = 5$	$R = 10$	$R = 15$
CP-ALS	0.9900	0.9900	0.9900
CPRAND-PREMIX	0.9897	0.9898	0.9898
CPRAND	0.9893	0.9895	0.9896
RBS	0.9881	0.9897	0.9898

Simulations

Coil-100 dataset

S. Nene, S. Nayar, and H. Murase, Columbia Object Image Library (COIL-100), Tech. Report CUCS-006-96, Columbia University, 1996.

100 different object classes

72 different angles

The size of the data: $128 \times 128 \times 3 \times 7200$

Simulations

For the CP-rank $R = 20$ the following results were achieved

	Cpu Time (Second)	Fit
CP-ALS	220	0.686
CPRAND-PREMIX	77.46	0.684

Simulations

Video compression

Video link: https://www.youtube.com/watch?v=1qeWugmiGt4

The size of this tensors : $360 \times 480 \times 3 \times 2200$

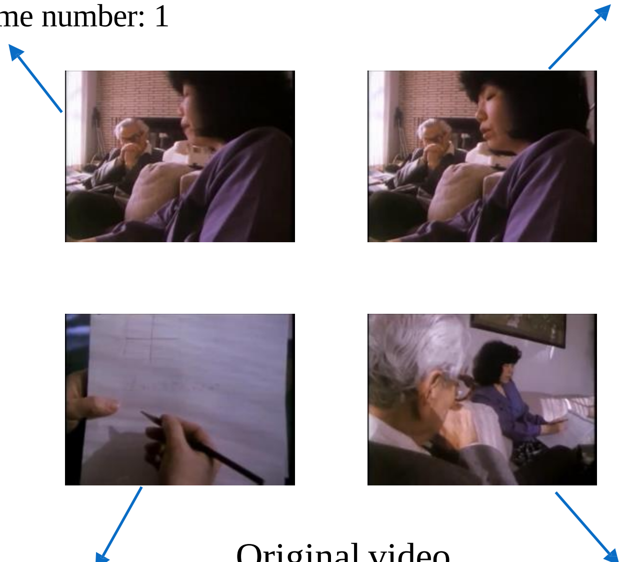
We consider only one and 29 second of that video which is a tensor of size

$$360 \times 480 \times 3 \times 300$$

Simulations

Frame number: 20

Frame number: 1



Frame number: 60

Original video

Frame number: 100

Compressed video by CP rank

$R = 100$

Here again the CP-PREMIX was 2 times faster than the CP-ALS.

Thanks for your attention!