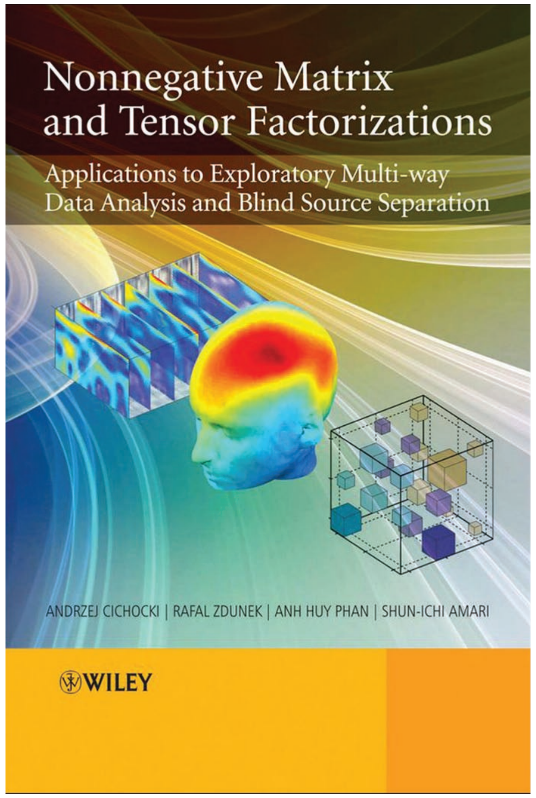
Multilinear Algebra and Tensor Decomposition



Qibin Zhao
Tensor Learning Unit
RIKEN AIP

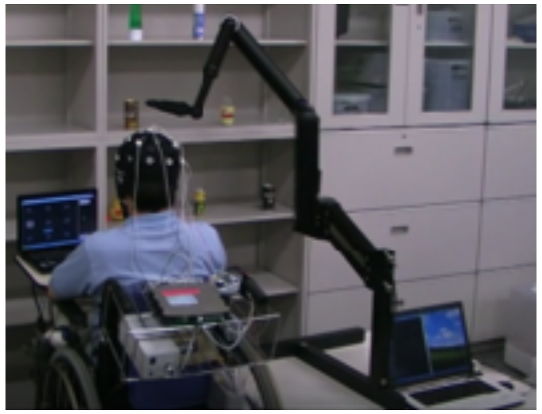
2018-6-2 @ Waseda University

Self-Introduction

- 2009, Ph.D. in Computer Science, Shanghai Jiao Tong University
- 2009 - 2017, RIKEN Brain Science Institute
- 2017 - Now, RIKEN AIP

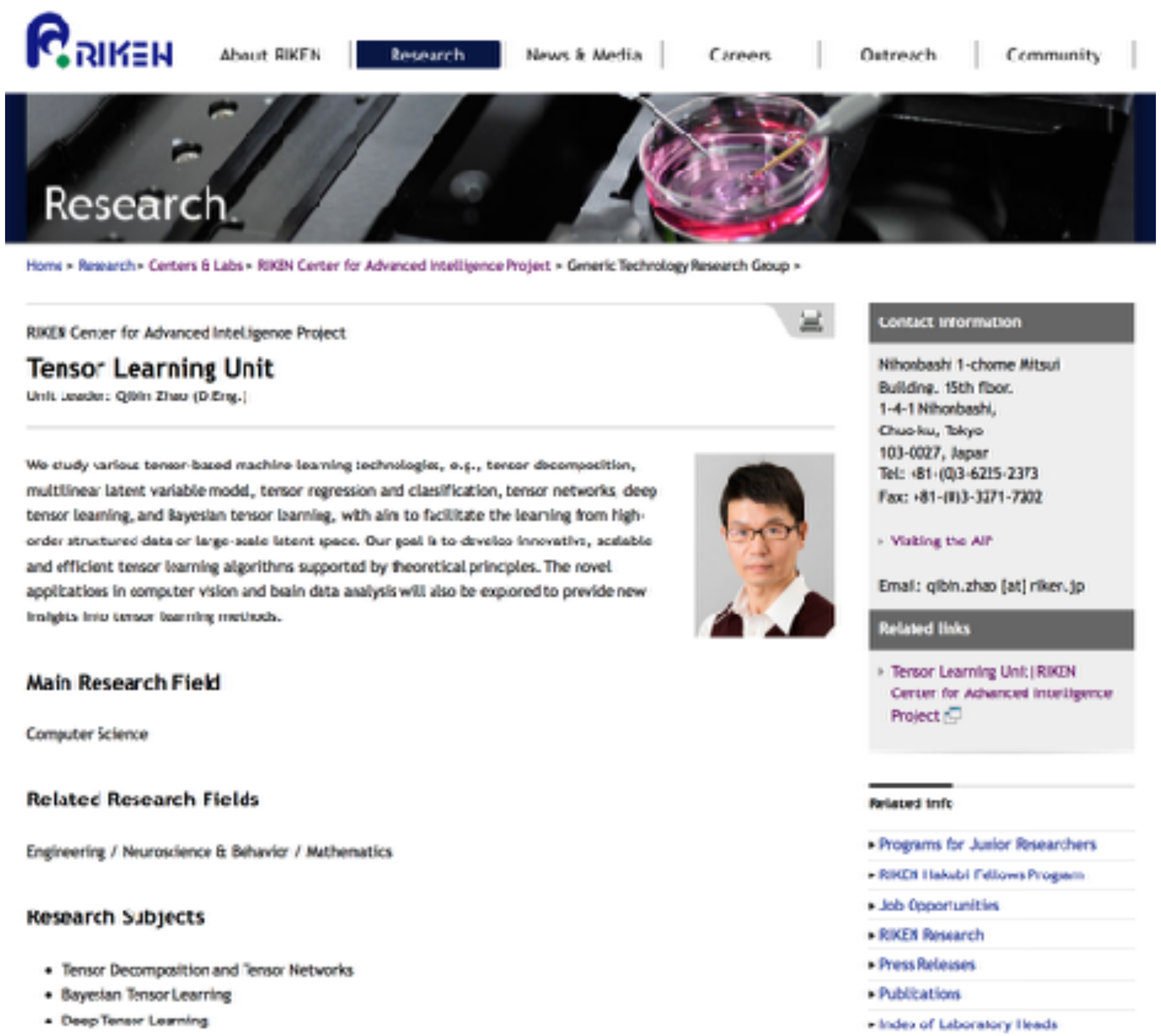
- Research Interests:
  - Brain computer interface, brain signal processing
  - Tensor decomposition and machine learning

Self-Introduction



Brain computer interface

Self-Introduction



About RIKEN | Research | News & Media | Careers | Outreach | Community

Research

Home » Research » Centers & Labs » RIKEN Center for Advanced Intelligence Project » Generic Technology Research Group »

RIKEN Center for Advanced Intelligence Project

Tensor Learning Unit

Unit Leader: Qibin Zhao (D.Eng.)

We study various tensor-based machine learning technologies, e.g., tensor decomposition, multilinear latent variable model, tensor regression and classification, tensor networks, deep tensor learning, and Bayesian tensor learning, with aim to facilitate the learning from high-order structured data or large-scale latent space. Our goal is to develop innovative, scalable and efficient tensor learning algorithms supported by theoretical principles. The novel applications in computer vision and brain data analysis will also be explored to provide new insights into tensor learning methods.

Main Research Field

Computer Science

Related Research Fields

Engineering / Neuroscience & Behavior / Mathematics

Research Subjects

- Tensor Decomposition and Tensor Networks
- Bayesian Tensor Learning
- Deep Tensor Learning

Contact Information

Nihonbashi 1-chome Mitsui Building, 15th floor, 1-4-1 Nihonbashi, Chuo-ku, Tokyo 103-0027, Japan
Tel: +81-(0)3-6215-2373
Fax: +81-(0)3-3271-7302

› Visiting the AIP

Email: qibin.zhao [at] riken.jp

Related links

› Tensor Learning Unit | RIKEN Center for Advanced Intelligence Project

Related Info

- Programs for Junior Researchers
- RIKEN Hakubi Fellows Program
- Job Opportunities
- RIKEN Research
- Press Releases
- Publications
- Index of Laboratory Heads

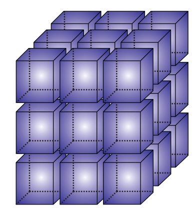
RECRUITMENT INFORMATION — MARCH 15, 2017

RIKEN Center for Advanced Intelligence Project (AIP)

http://www.riken.jp/en/research/labs/aip/

Tensor Learning Unit

Unit Leader: Dr. Qibin Zhao



Tensors are high-dimensional generalizations of vectors and matrices, which can provide a natural and scalable representation for multi-dimensional, multi-relational or multi-aspect data with inherent structure and complex dependence. In our team, we investigate the various tensor-based machine learning models, e.g., tensor decomposition, multilinear latent variable model, tensor regression and classification, tensor networks, deep tensor learning, and Bayesian tensor learning, with aim to facilitate the learning from high-dimensional structured data or large-scale latent parameter space. In addition, we develop the scalable and efficient tensor learning algorithms supported by the theoretical principles, with the goal to advance existing machine learning approaches. The novel applications in computer vision and brain data analysis will also be exploited to provide new insights into tensor learning methods.

Opening Positions

1	2	3
POSTDOCTORAL RESEARCHER	TECHNICAL STAFF	RESEARCH INTERN
Doctoral degree	Technical support for researchers	Ph.D students are preferable

We are seeking talented and creative researchers who are willing to solve the challenging problems in machine learning. For research topics, please refer to the bottom-right side. If you are interested in joining our team, please contact us (see the top-right side).

Contact Information

Mitsui Building, 15th floor, 1-4-1 Nihonbashi, Chuo-ku, Tokyo103-0027, Japan

Email: qibin.zhao@riken.jp

Research Field

Computer Science

Related Fields

Machine Learning
Computer Vision
Neuroscience

Research Subjects

- Tensor Decomposition
- Tensor Networks
- Tensor Regression and Classification
- Deep Tensor Learning
- Bayesian Tensor Learning

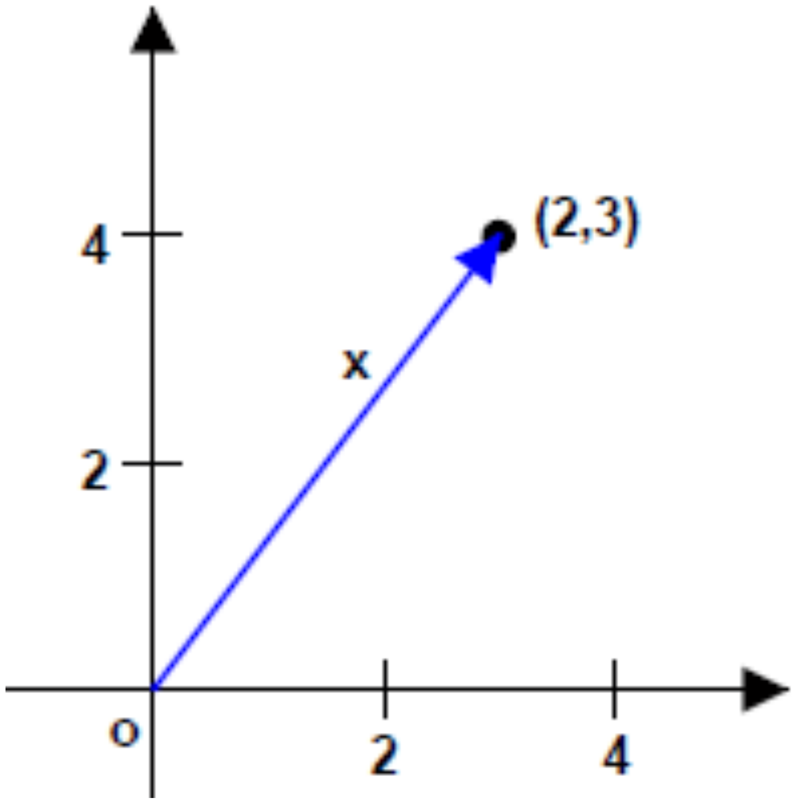
Outline

- Vector and linear algebra
- Matrix and its decomposition
- What is tensor?
- Basic operations in tensor algebra
- Classical tensor decomposition
    - ✦CP Decomposition
    - ✦Tucker Decomposition

Vectors

- We can think of vectors in two ways:
  - Points in a multidimensional space with respect to some coordinate system
  - translation of a point in a multidimensional space

    ex., translation of the origin (0,0)

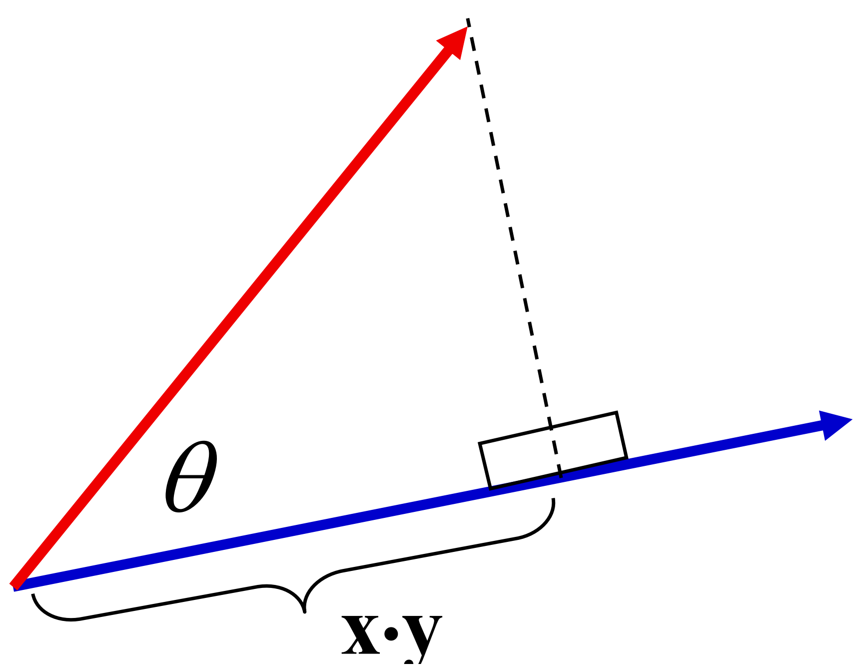


Dot Product or Scalar Product

- Dot product is the product of two vectors
- Example:

$$\mathbf{x}\cdot\mathbf{y}=\begin{bmatrix}x_1\\ x_2\end{bmatrix}\cdot\begin{bmatrix}y_1\\ y_2\end{bmatrix}=x_1y_1+x_2y_2=s$$

- It is the projection of one vector onto another



$$\mathbf{x}\cdot\mathbf{y}=\|\mathbf{x}\|\|\mathbf{y}\|\cos\theta$$

Dot Product or Scalar Product

- Commutative: $$\mathbf{x} \cdot \mathbf{y} = \mathbf{y} \cdot \mathbf{x}$$
- Distributive: $$(\mathbf{x} + \mathbf{y}) \cdot \mathbf{z} = \mathbf{x} \cdot \mathbf{z} + \mathbf{y} \cdot \mathbf{z}$$
- Linearity

$$(c\mathbf{x}) \cdot \mathbf{y} = c(\mathbf{x} \cdot \mathbf{y})$$
$$\mathbf{x} \cdot (c\mathbf{y}) = c(\mathbf{x} \cdot \mathbf{y})$$

$$(c_1\mathbf{x}) \cdot (c_2\mathbf{y}) = (c_1 c_2)(\mathbf{x} \cdot \mathbf{y})$$

Norms

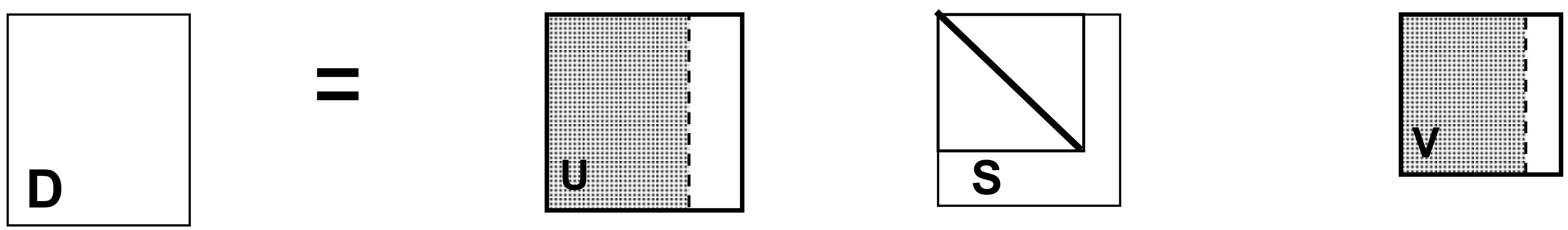
- Euclidean norm (sometimes called 2-norm):

$$\|\mathbf{x}\| = \|\mathbf{x}\|_2 = \sqrt{\mathbf{x}\cdot\mathbf{x}} = \sqrt{x_1^2 + x_2^2 + \cdots + x_n^2} = \sqrt{\sum_{i=1}^{n} x_i^2}$$

- The length of a vector is defined to be its (Euclidean) norm.
- A unit vector is of length 1.

Singular Value Decomposition

$D=USV^T$



- A matrix $\mathbf{D} \in \mathbb{R}^{I_1 \mathrm{x}\, I_2}$ has a column space and a row space
- SVD orthogonalizes these spaces and decomposes $\mathbf{D}$

$$\mathbf{D} = \mathbf{U}\mathbf{S}\mathbf{V}^{T}$$

($\mathbf{U}$ contains the left singular vectors/eigenvectors)

($\mathbf{V}$ contains the right singular vectors/eigenvectors)

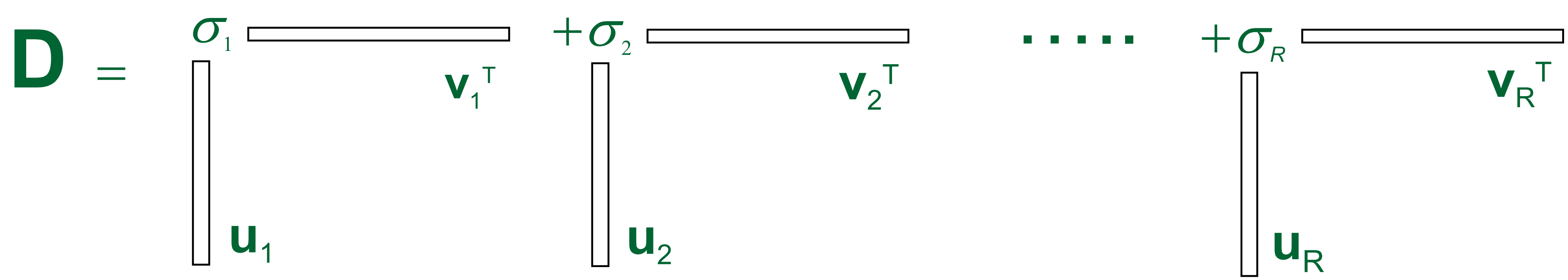
- Rewrite as a sum of a minimum number of rank-1 matrices

$$\mathbf{D} = \sum_{r=1}^{R} \sigma_r \mathbf{u}_r \circ \mathbf{v}_r$$

Matrix SVD Properties

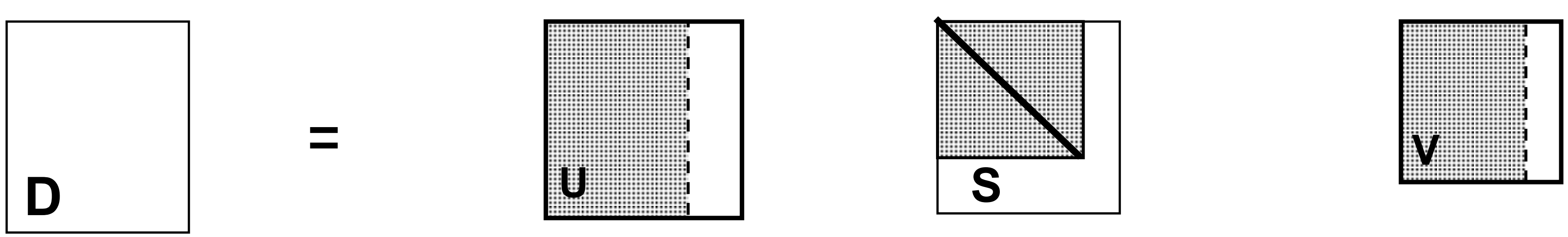
- Rank Decomposition:
  - sum of min. number of rank-1 matrices

$$\mathbf{D} = \sum_{r=1}^{R} \sigma_r \mathbf{u}_r \circ \mathbf{v}_r$$



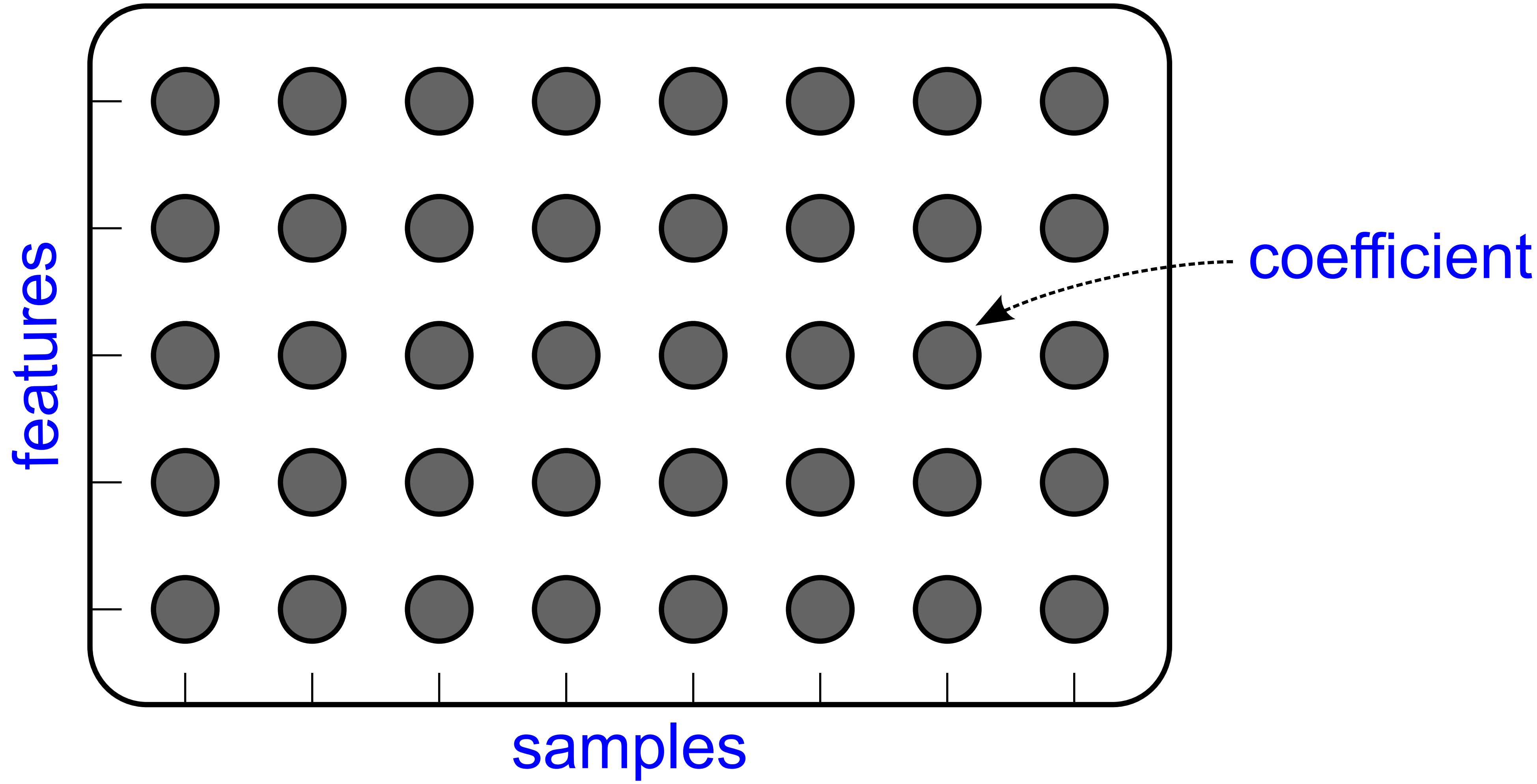
- Multilinear Rank Decomposition:

$$\mathbf{D} = \sum_{r_1=1}^{R_1} \sum_{r_2=1}^{R_2} \sigma_{r_1 r_2} \mathbf{u}_{r_1} \circ \mathbf{v}_{r_2}$$



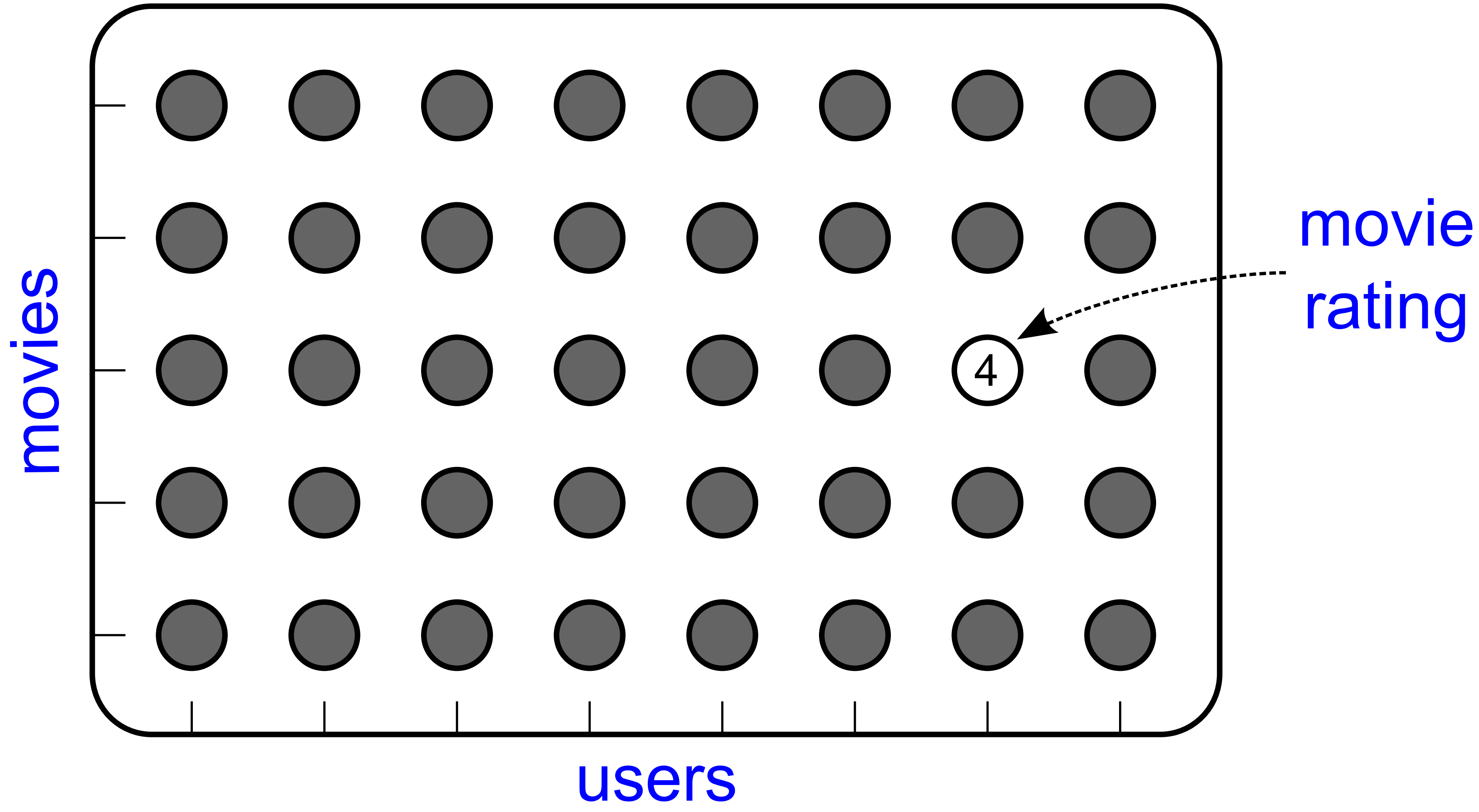
Matrix in Machine Learning

Data often available in matrix form.



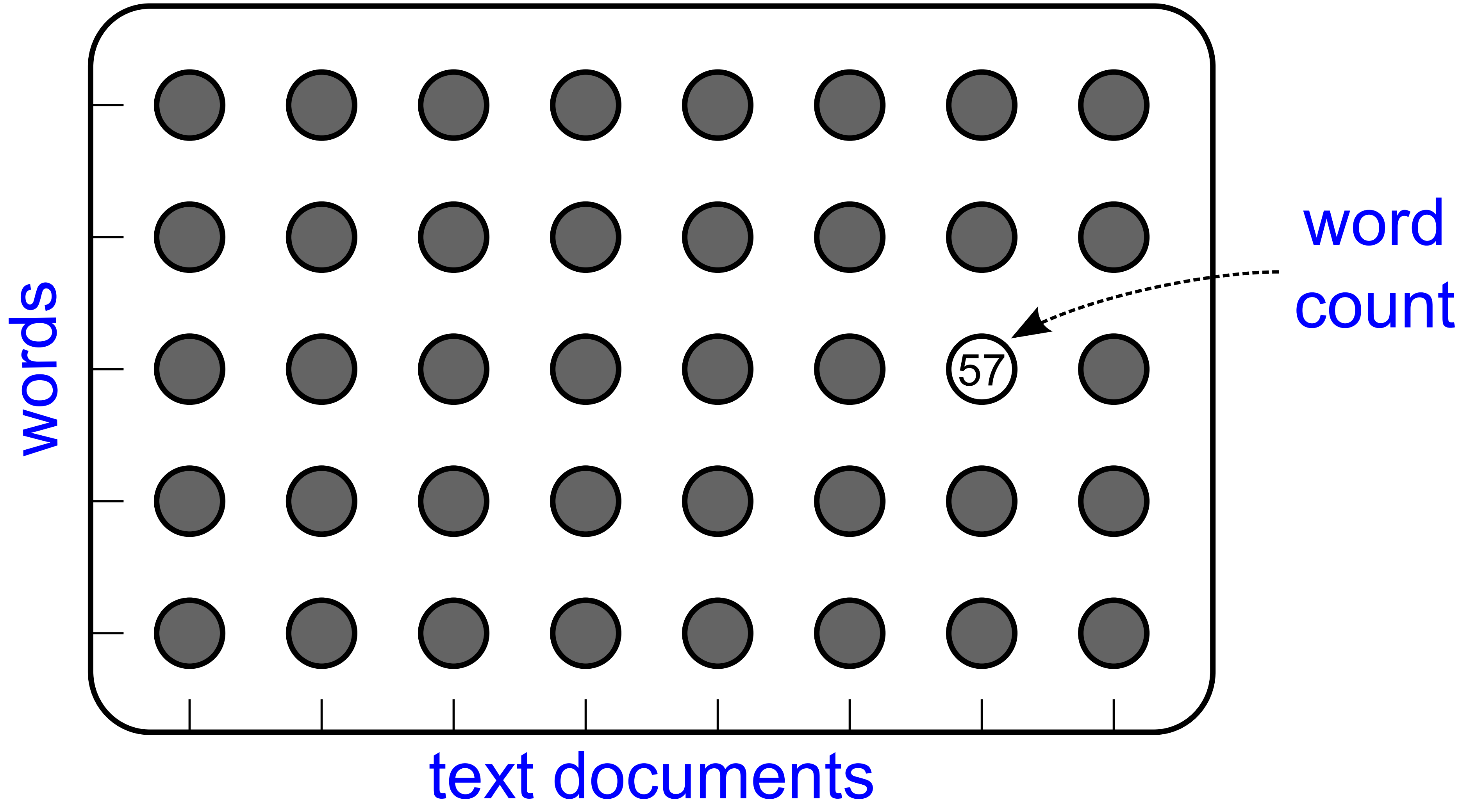
Matrix in Machine Learning

Data often available in matrix form.



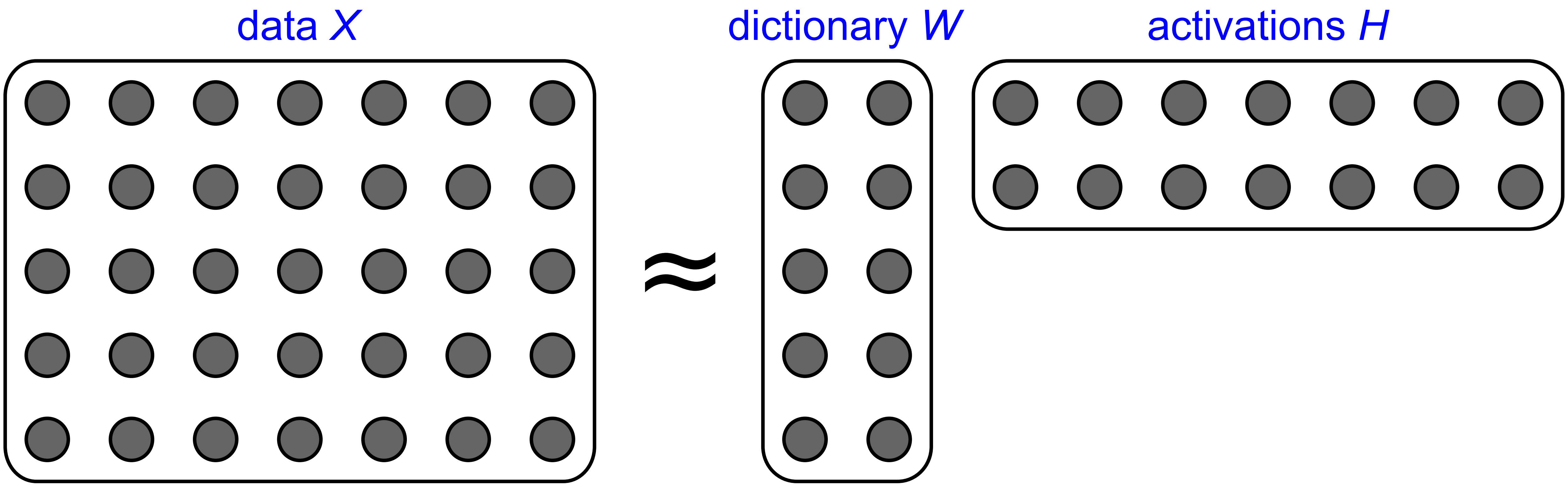
Matrix in Machine Learning

Data often available in matrix form.



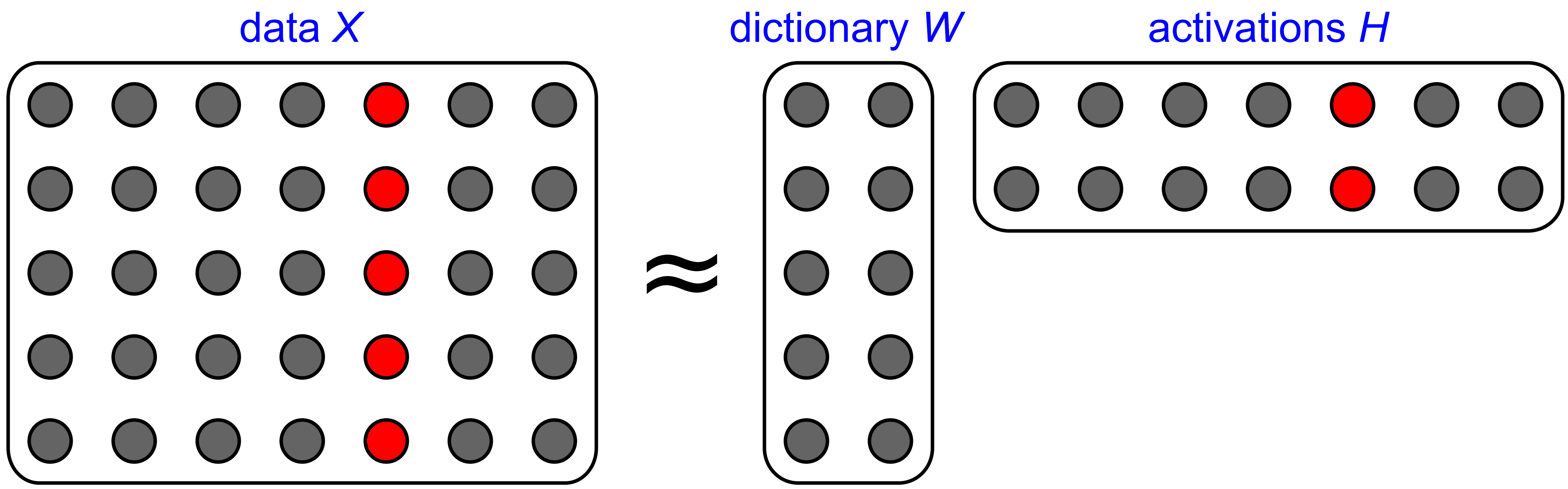
Matrix Decomposition in Machine Learning

$\approx$ **dictionary learning**
low-rank approximation
factor analysis
latent semantic analysis

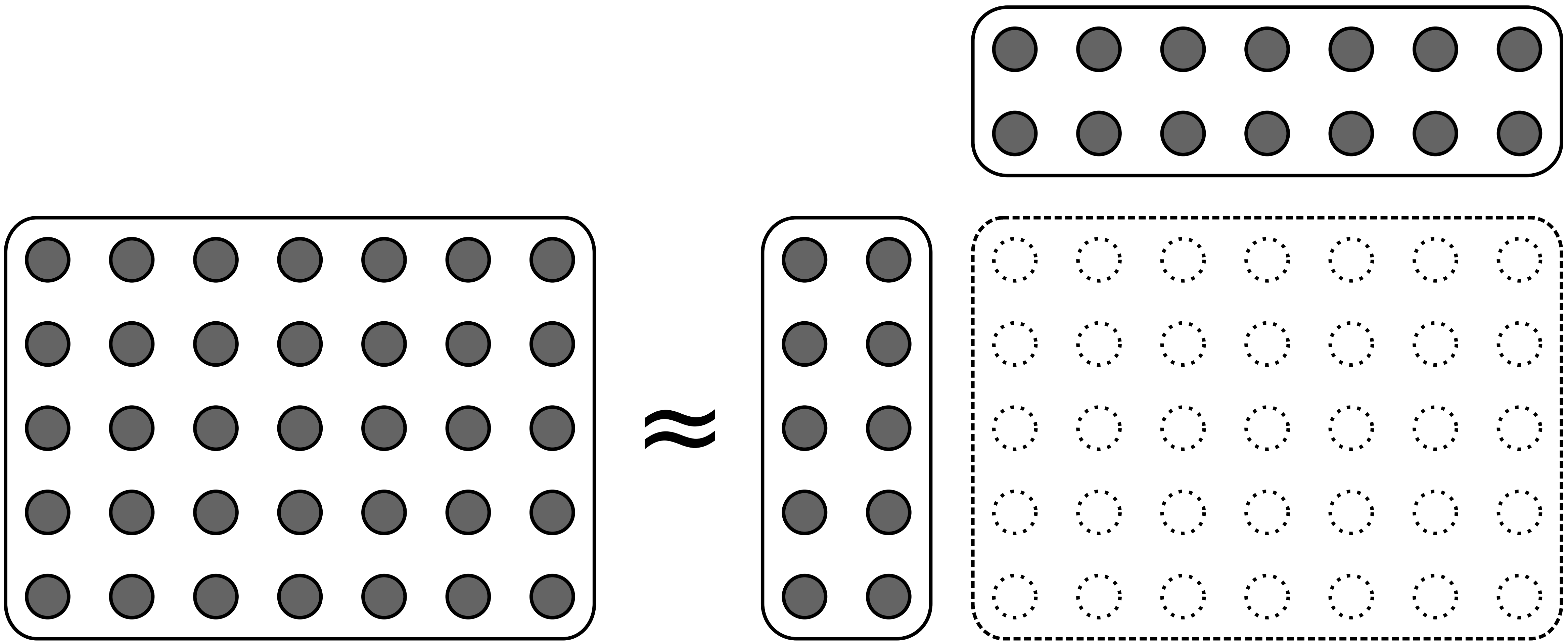


Matrix Decomposition in Machine Learning

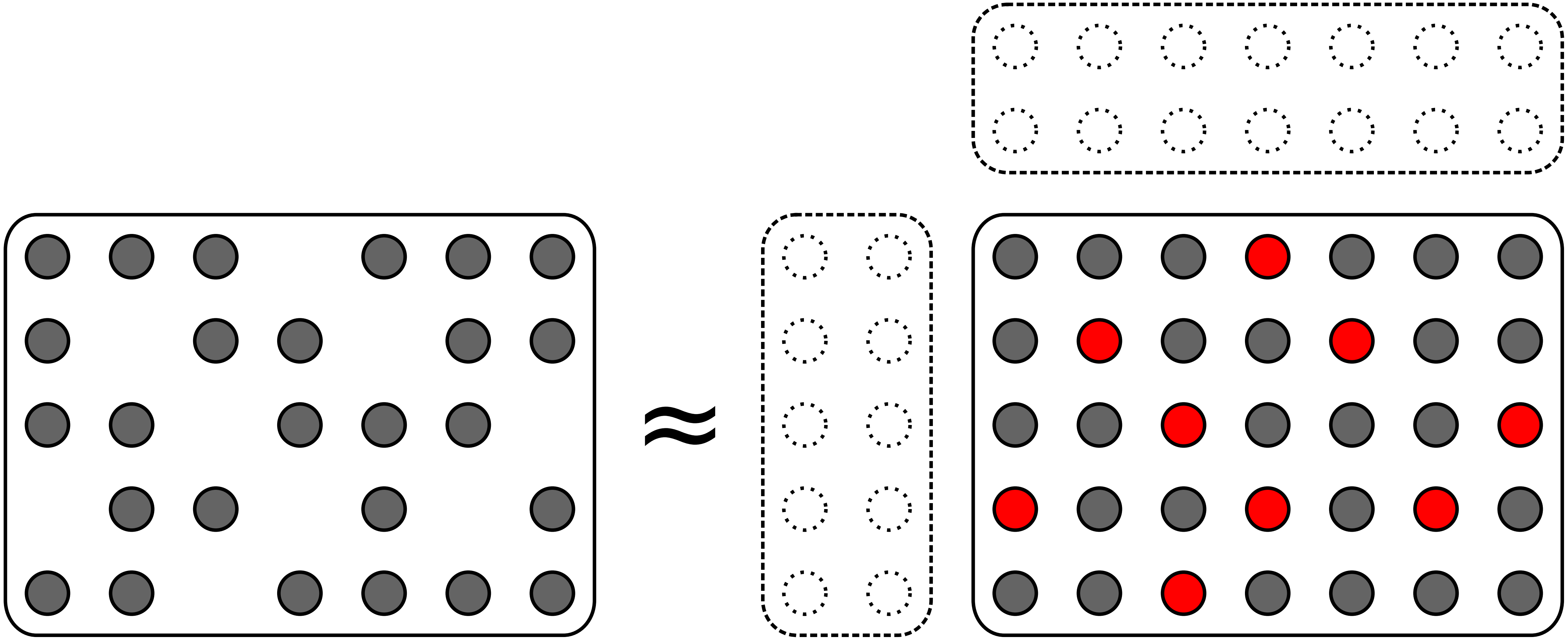
$\approx$ **dictionary learning**
low-rank approximation
factor analysis
latent semantic analysis



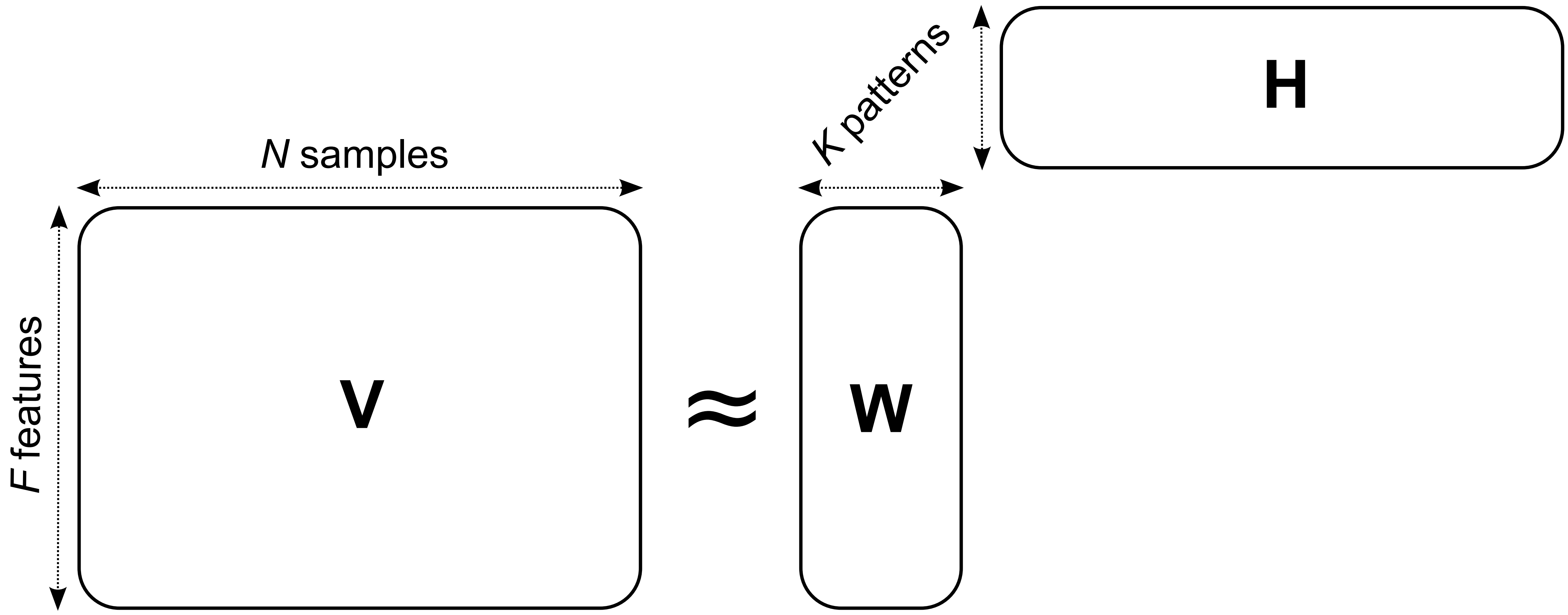
for dimensionality reduction (coding, low-dimensional embedding)



for interpolation (collaborative filtering, image inpainting)



Basic Model of Matrix Decomposition



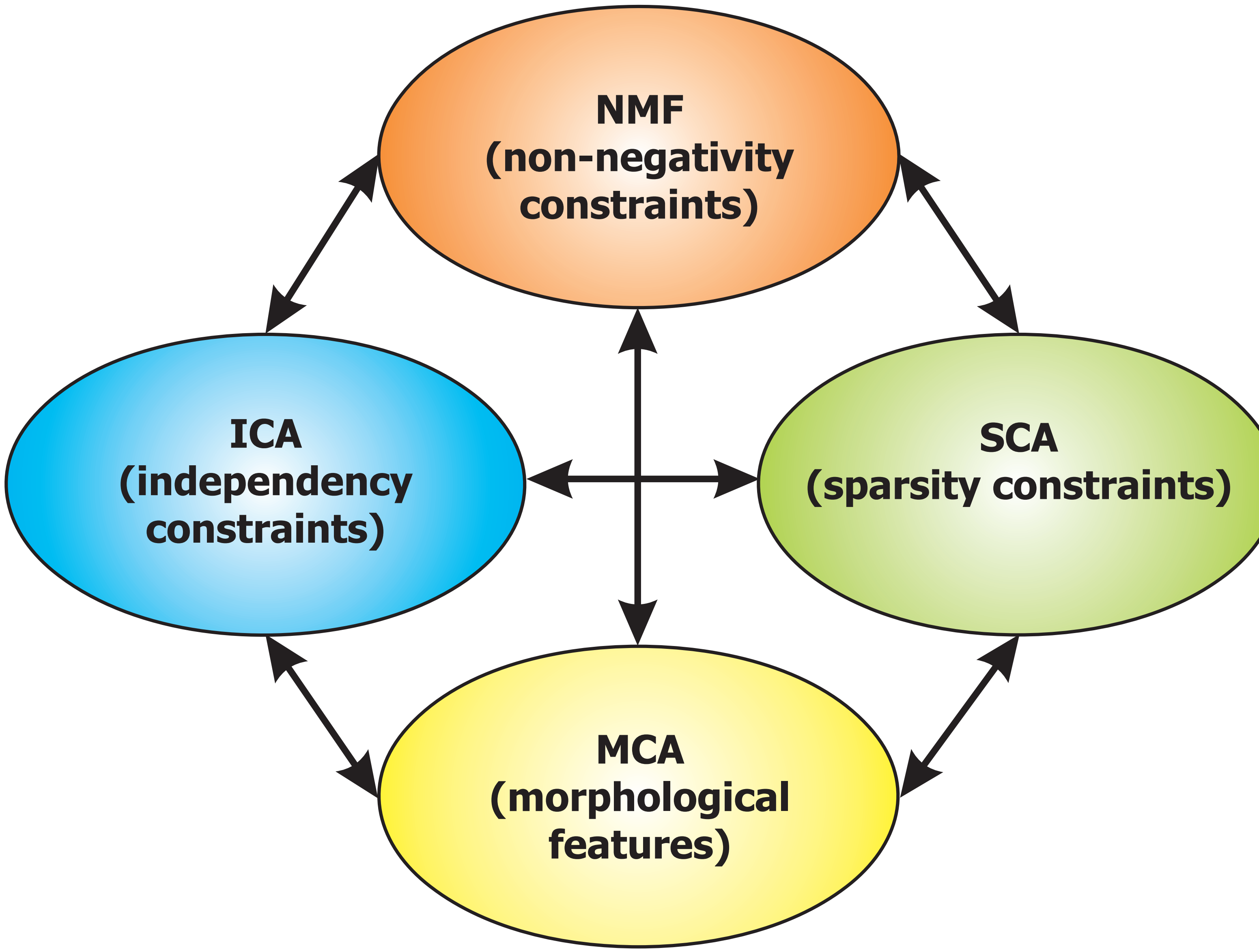
Matrix Decomposition with Constraints

Different types of constraints have been considered in previous works:

- **Sparsity** constraints: either on $\mathbf{W}$ or $\mathbf{H}$ (*e.g.*, Hoyer, 2004; Eggert and Korner, 2004);
- **Shape** constraints on $\mathbf{w}_k$, *e.g.*:
  - **convex NMF**: $\mathbf{w}_k$ are convex combinations of inputs (Ding et al., 2010);
  - **harmonic NMF**: $\mathbf{w}_k$ are mixtures of harmonic spectra (Vincent et al., 2008).
- **Spatial coherence** or **temporal** constraints on $\mathbf{h}_k$: activations are **smooth** (Virtanen, 2007; Jia and Qian, 2009; Essid and Fevotte, 2013);
- **Cross-modal correspondence** constraints: factorisations of related modalities are related, *e.g.*, temporal activations are correlated (Seichepine et al., 2013; Liu et al., 2013; Yilmaz et al., 2011);
- **Geometric** constraints: *e.g.*, select particular cones $\mathcal{C}_{\mathbf{w}}$ (Klingenberg et al., 2009; Essid, 2012).

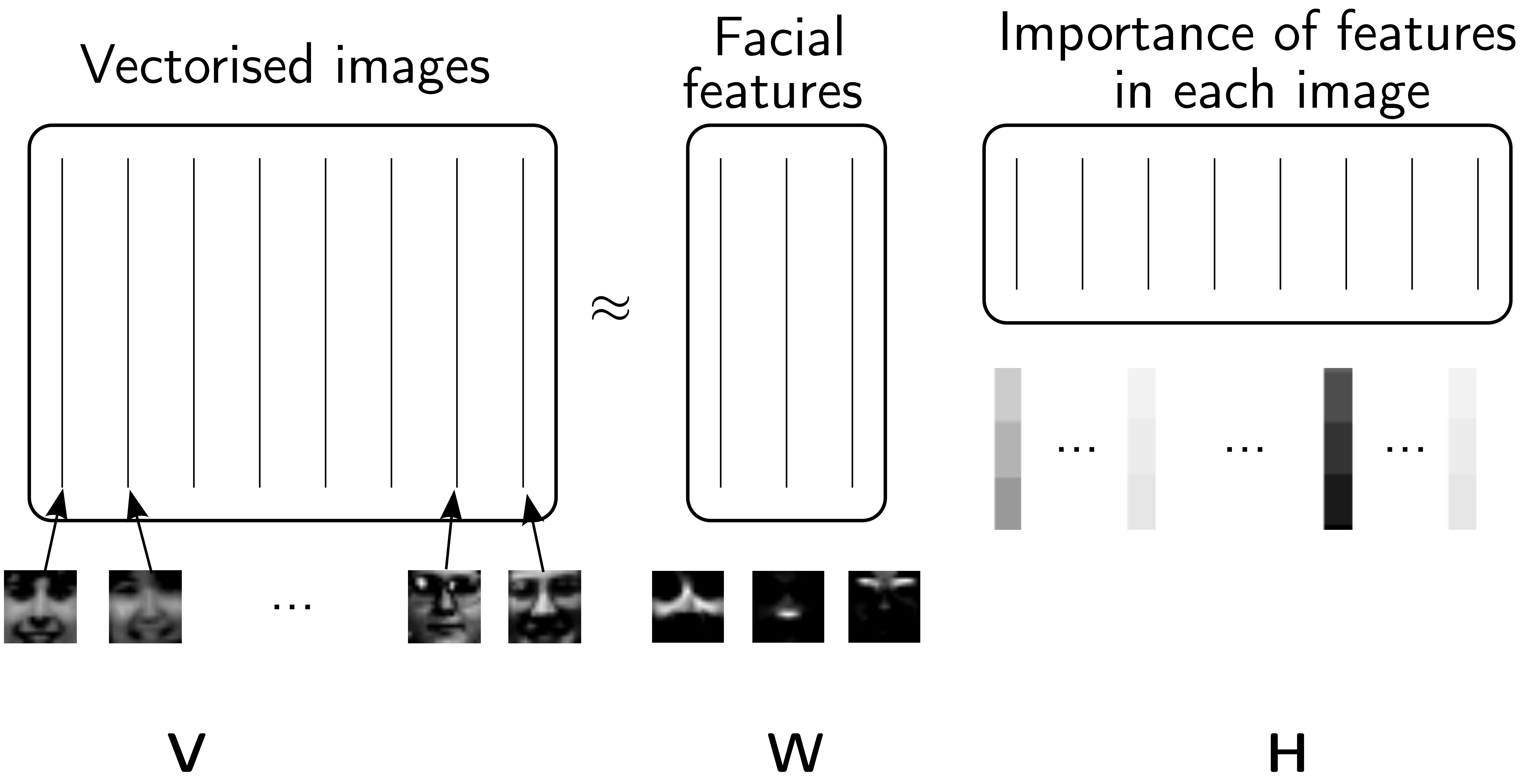
Matrix and Matrix Decomposition

- ICA (Independent Component Analysis)
- SCA (Sparse Component Analysis)
- MCA (Morphological Component Analysis)
- NMF (Non-negative Factorization)

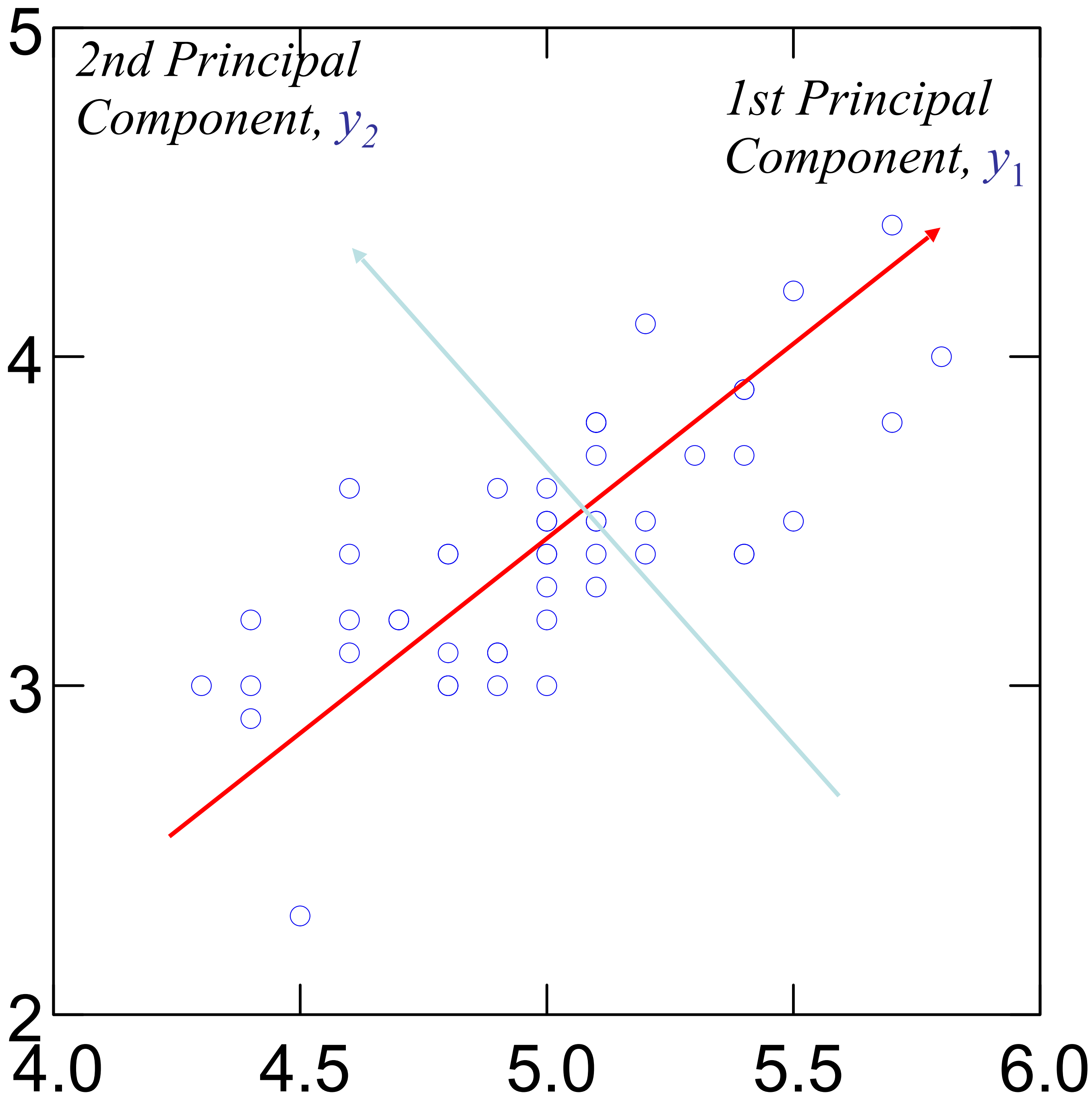


49 images among 2429 from MIT's CBCL face dataset

Principal Component Analysis



Principal Components



Objective Function:

$$\max_{\mathbf{w}} \left(\mathbf{w}^T \mathbf{X} \mathbf{X}^T \mathbf{w} \right)$$

- PCA is to look for a low dimensional projection in which the majority of signal energy is kept.
- Here "Principal" represents "Major" that the projected signal has the largest energy along the first principal direction (red line in the figure).

Principal Component Analysis

Assuming the data is real-valued ($\mathbf{v}_n \in \mathbb{R}^F$) and centered ($\mathbb{E}[\mathbf{v}] = 0$),

- PCA returns a dictionary $\mathbf{W}_{PCA} \in \mathbb{R}^{F \times K}$ such that the **least squares error** is minimized:

$$\mathbf{W}_{PCA} = \min_{\mathbf{W}} \frac{1}{N} \sum_n \|\mathbf{v}_n - \hat{\mathbf{v}}_n\|_2^2 = \frac{1}{N} \|\mathbf{V} - \mathbf{W}\mathbf{W}^T\mathbf{V}\|_F^2$$

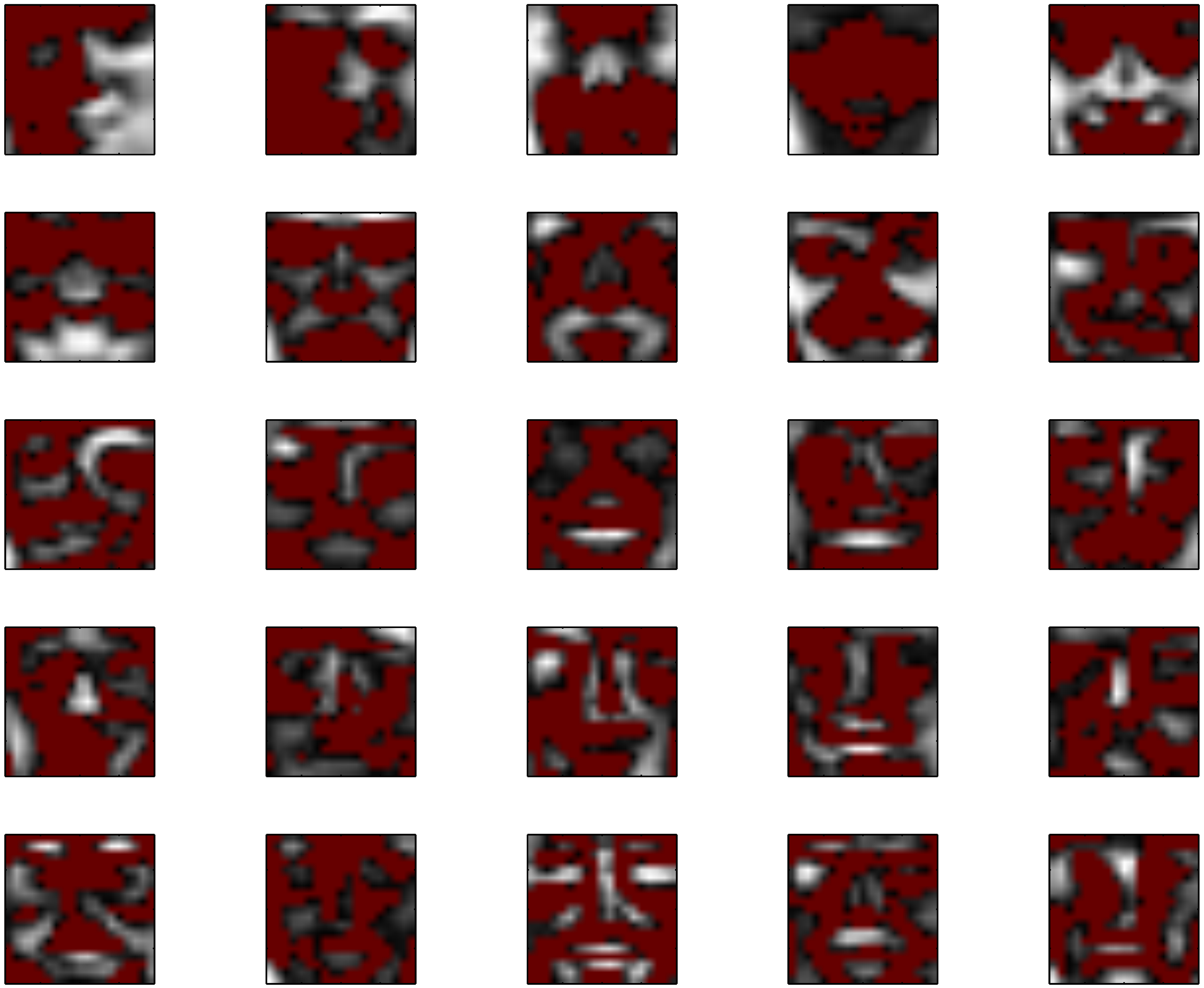
- A solution is given by:

$$\mathbf{W}_{PCA} = \mathbf{E}_{1:K}$$

where $\mathbf{E}_{1:K}$ denotes the K dominant **eigenvectors** of $\mathbf{C}_{\mathbf{v}}$:

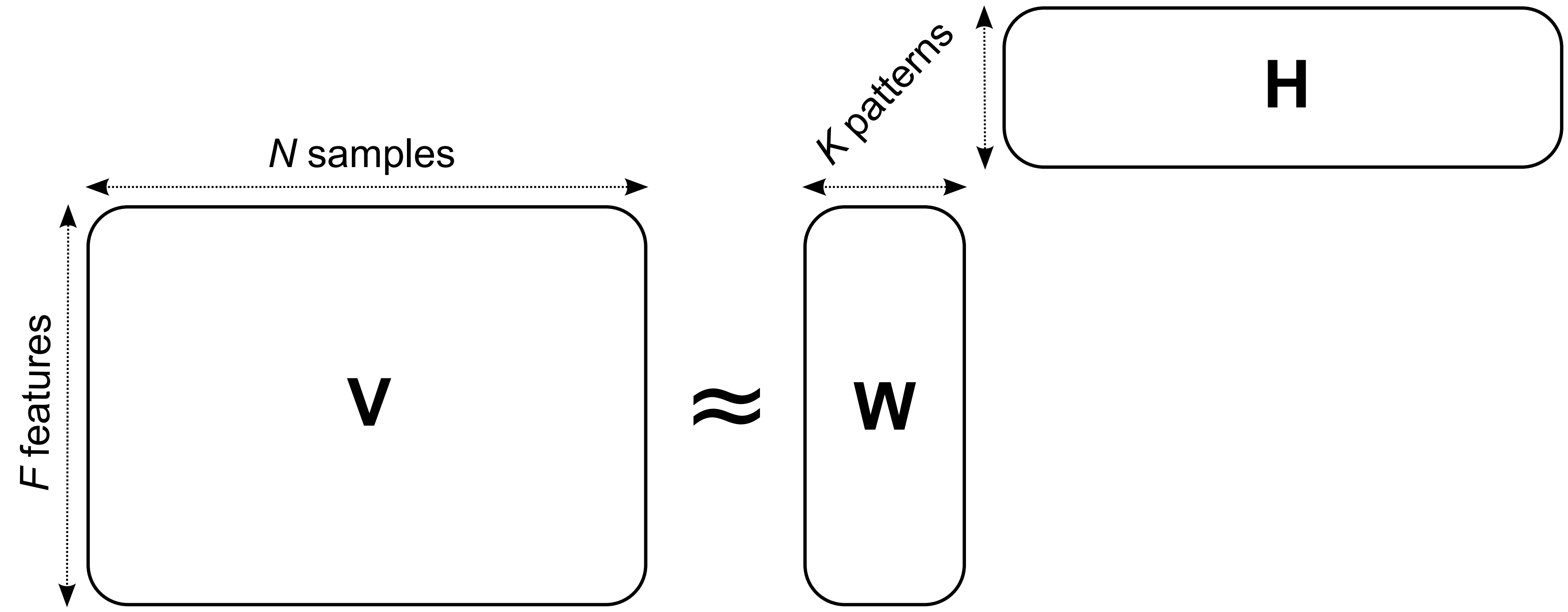
$$\mathbf{C}_{\mathbf{v}} = \mathbb{E}[\mathbf{v}\mathbf{v}^T] \approx \frac{1}{N} \sum_n \mathbf{v}_n \mathbf{v}_n.$$

PCA dictionary with K=25



red pixels indicate negative values

Nonnegative Matrix Decomposition



- data **V** and factors **W**, **H** have nonnegative entries.
- nonnegativity of **W** ensures interpretability of the dictionary, because patterns $\mathbf{w}_k$ and samples $\mathbf{v}_n$ belong to the same space.
- nonnegativity of **H** tends to produce part-based representations, because subtractive combinations are forbidden.

Early work by Paatero and Tapper (1994), landmark *Nature* paper by Lee and Seung (1999)

NMF as a constrained minimization problem

Minimise a measure of fit between $\mathbf{V}$ and $\mathbf{WH}$, subject to nonnegativity:

$$\min_{\mathbf{W},\mathbf{H}\geq \mathbf{0}} D(\mathbf{V}|\mathbf{WH}) = \sum_{fn} d([\mathbf{V}]_{fn}|[\mathbf{WH}]_{fn}),$$

where $d(x|y)$ is a scalar cost function, e.g.,

- squared Euclidean distance (Paatero and Tapper, 1994; Lee and Seung, 2001)
- Kullback-Leibler divergence (Lee and Seung, 1999; Finesso and Spreij, 2006)
- Itakura-Saito divergence (Févotte, Bertin, and Durrieu, 2009)
- α-divergence (Cichocki et al., 2008)
- β-divergence (Cichocki et al., 2006; Févotte and Idier, 2011)
- Bregman divergences (Dhillon and Sra, 2005)
- and more in (Yang and Oja, 2011)

Regularisation terms often added to $D(\mathbf{V}|\mathbf{WH})$ for sparsity, smoothness, dynamics, etc.

Common NMF algorithm design

- Block-coordinate update of $\mathbf{H}$ given $\mathbf{W}^{(i-1)}$ and $\mathbf{W}$ given $\mathbf{H}^{(i)}$.
- Updates of $\mathbf{W}$ and $\mathbf{H}$ equivalent by transposition:

$$\mathbf{V} \approx \mathbf{W}\mathbf{H} \Leftrightarrow \mathbf{V}^T \approx \mathbf{H}^T\mathbf{W}^T$$

- Objective function separable in the columns of $\mathbf{H}$ or the rows of $\mathbf{W}$:

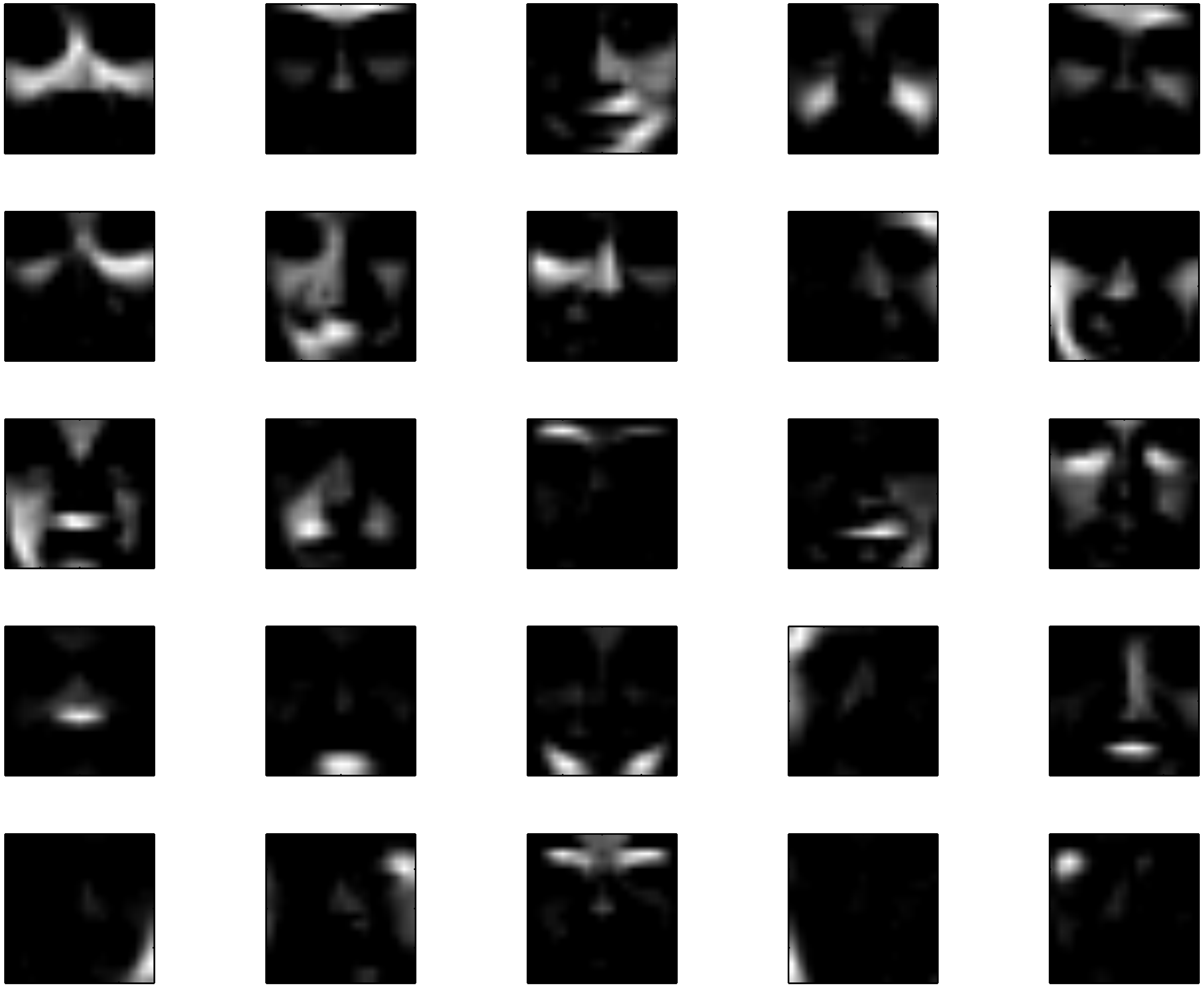
$$D(\mathbf{V}|\mathbf{W}\mathbf{H}) = \sum_n D(\mathbf{v}_n|\mathbf{W}\mathbf{h}_n)$$

- Essentially left with nonnegative linear regression:

$$\min_{\mathbf{h}\geq\mathbf{0}} C(\mathbf{h}) \stackrel{\text{def}}{=} D(\mathbf{v}|\mathbf{W}\mathbf{h})$$

Numerous references in the image restoration literature. e.g., (Richardson, 1972; Lucy, 1974; Daube-Witherspoon and Muehllehner, 1986; De Pierro, 1993)

NMF dictionary with K=25

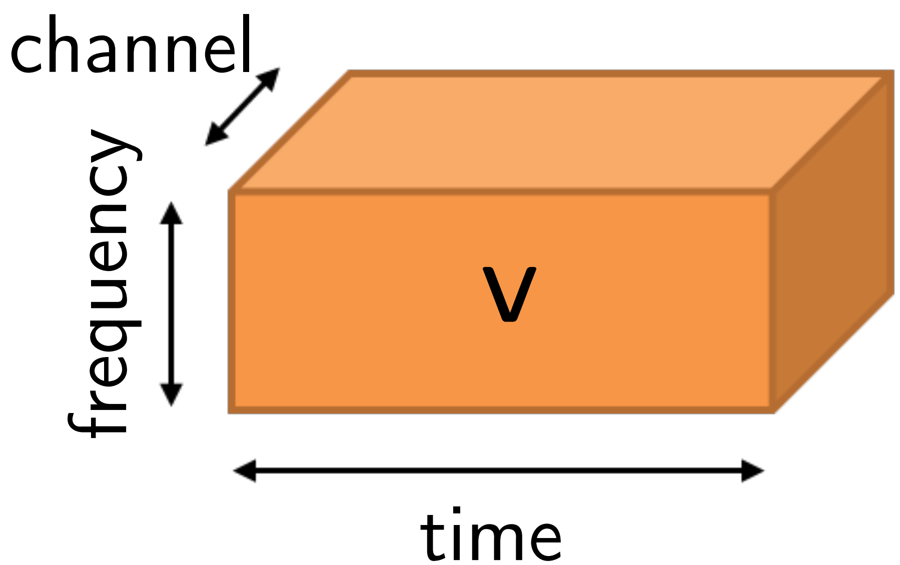


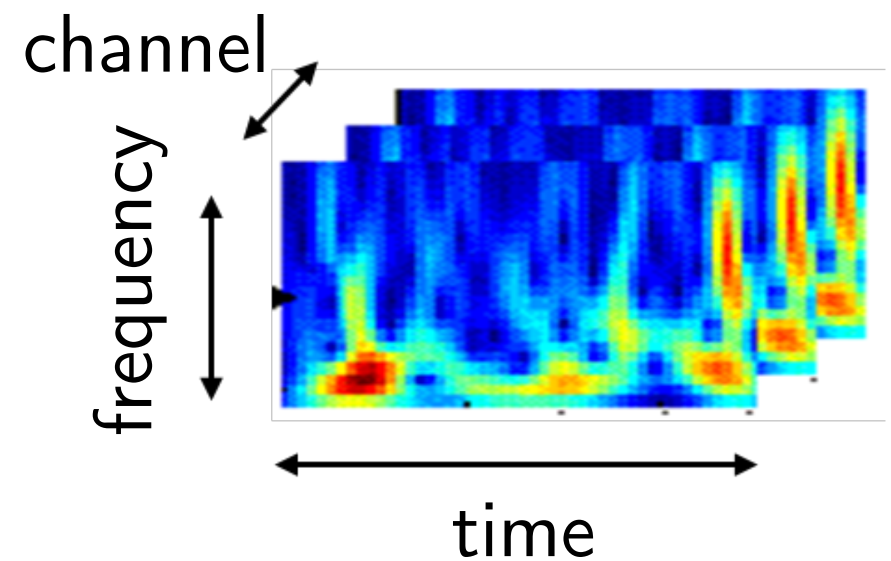
experiment reproduced from (Lee and Seung, 1999)

What is tensor?

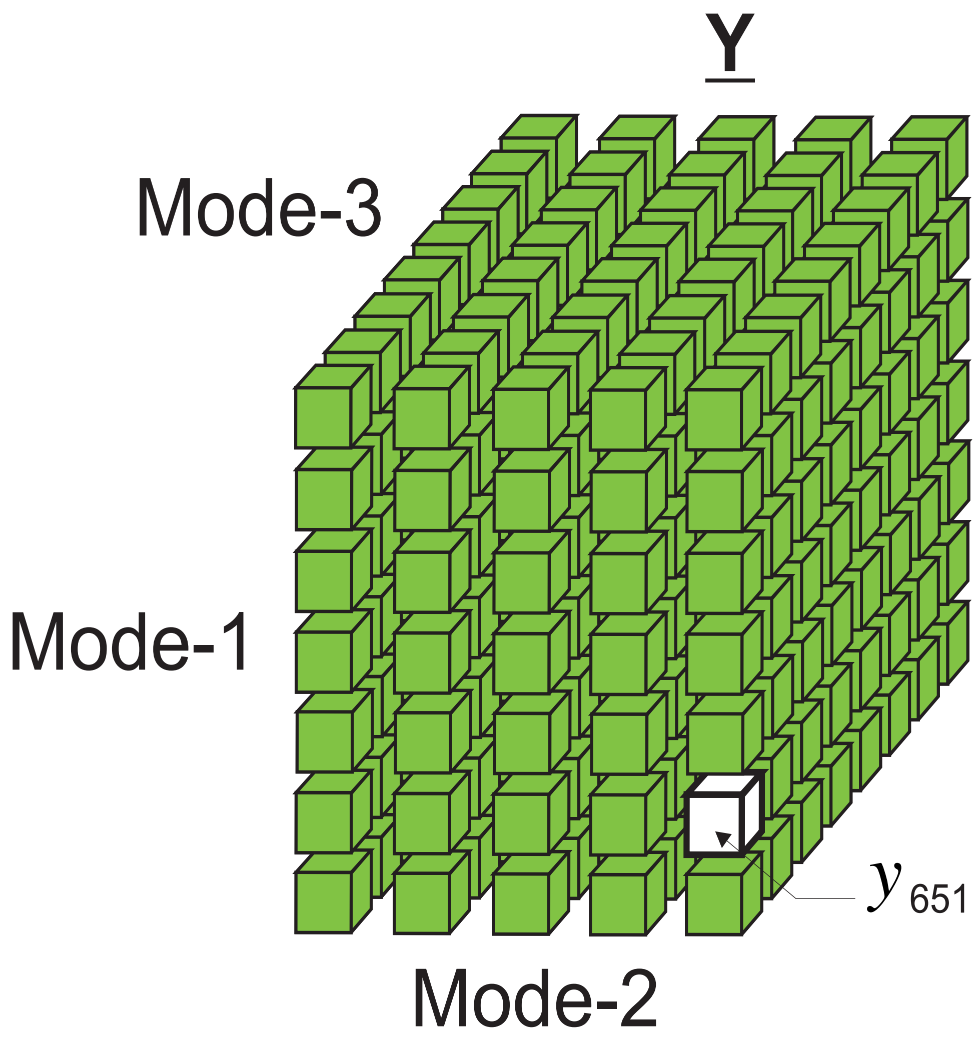
Some data can have more **meaningful representation** using **multi-way arrays** rather than **matrices** (two-way arrays).

Electroencephalography (EEG) data (Lee et al., 2007)

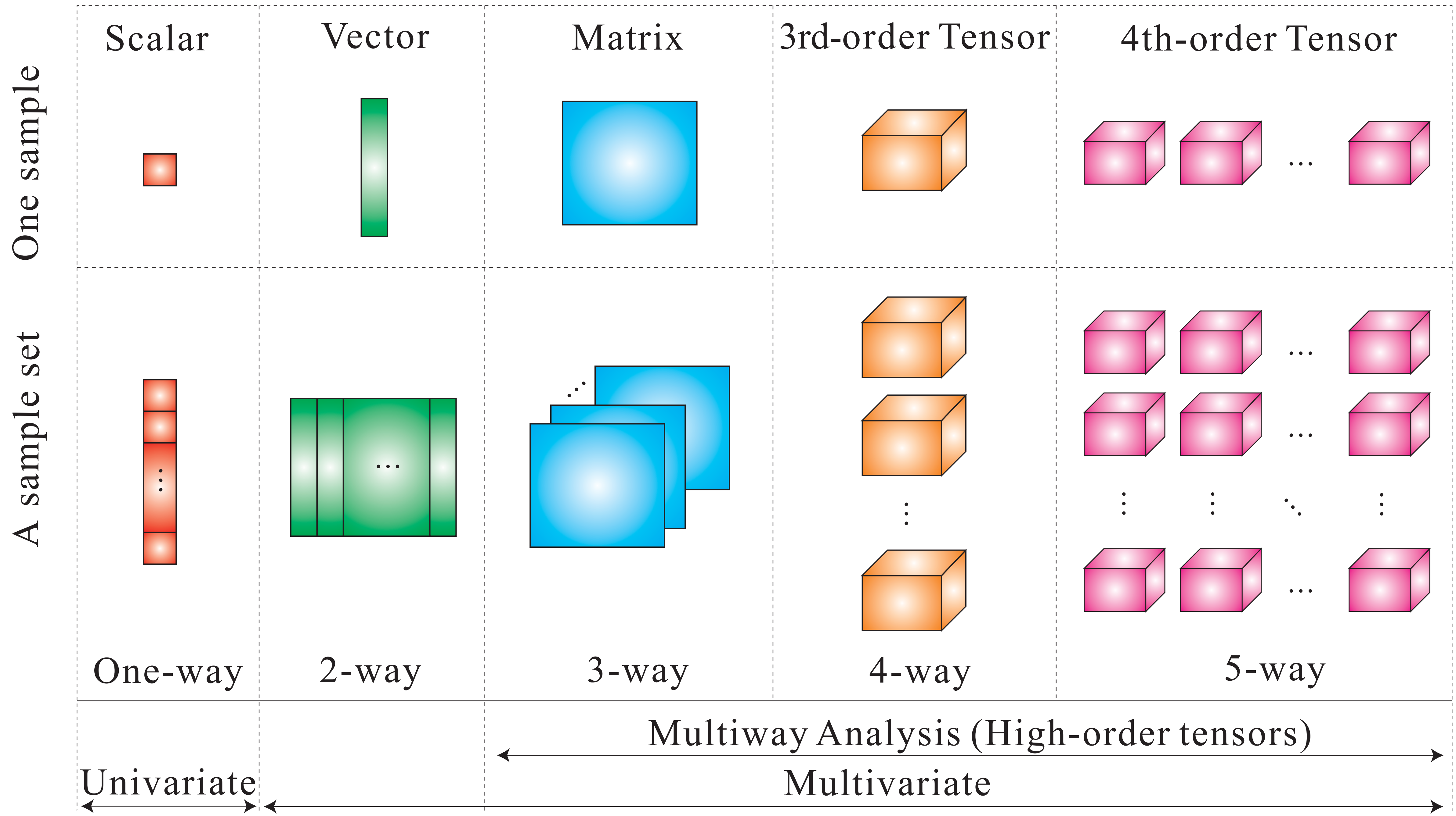




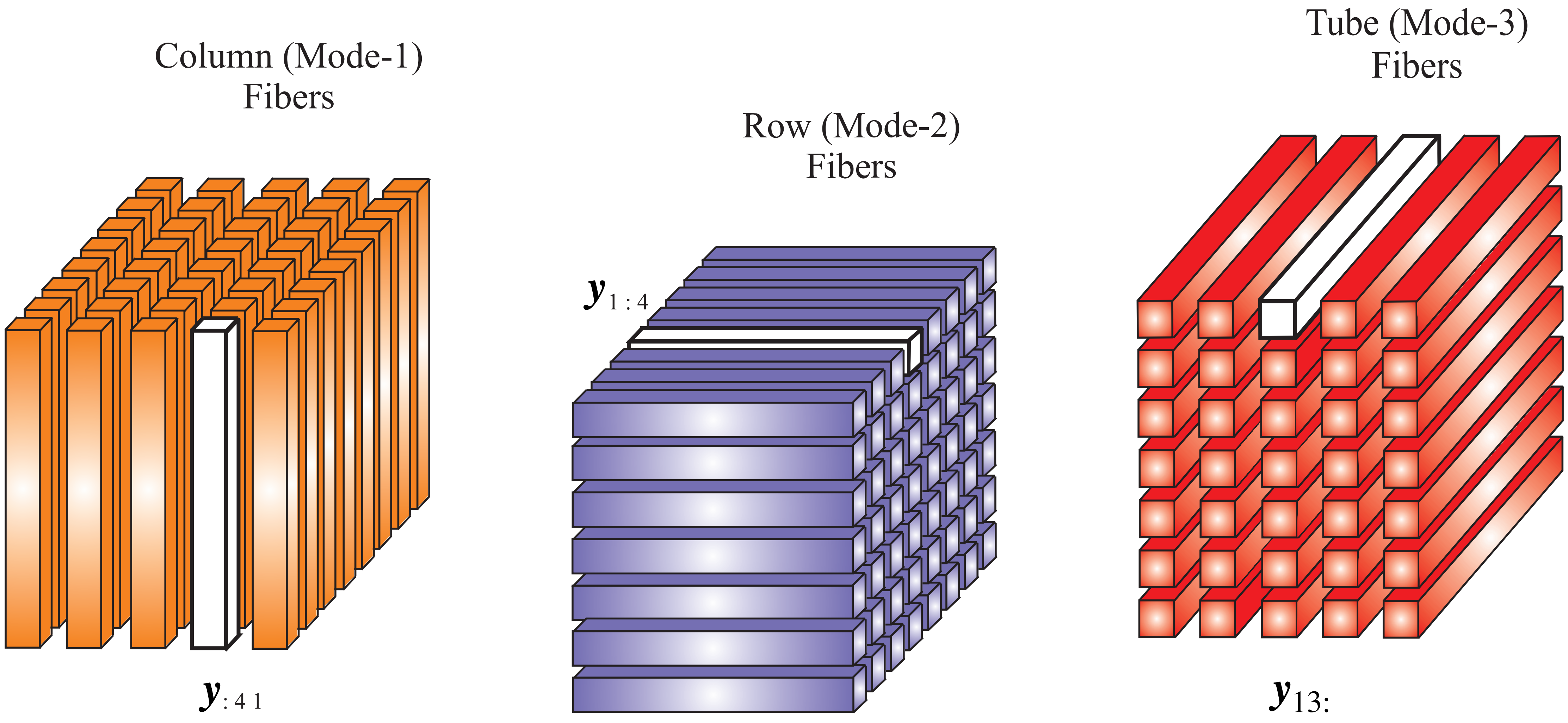
What is tensor?



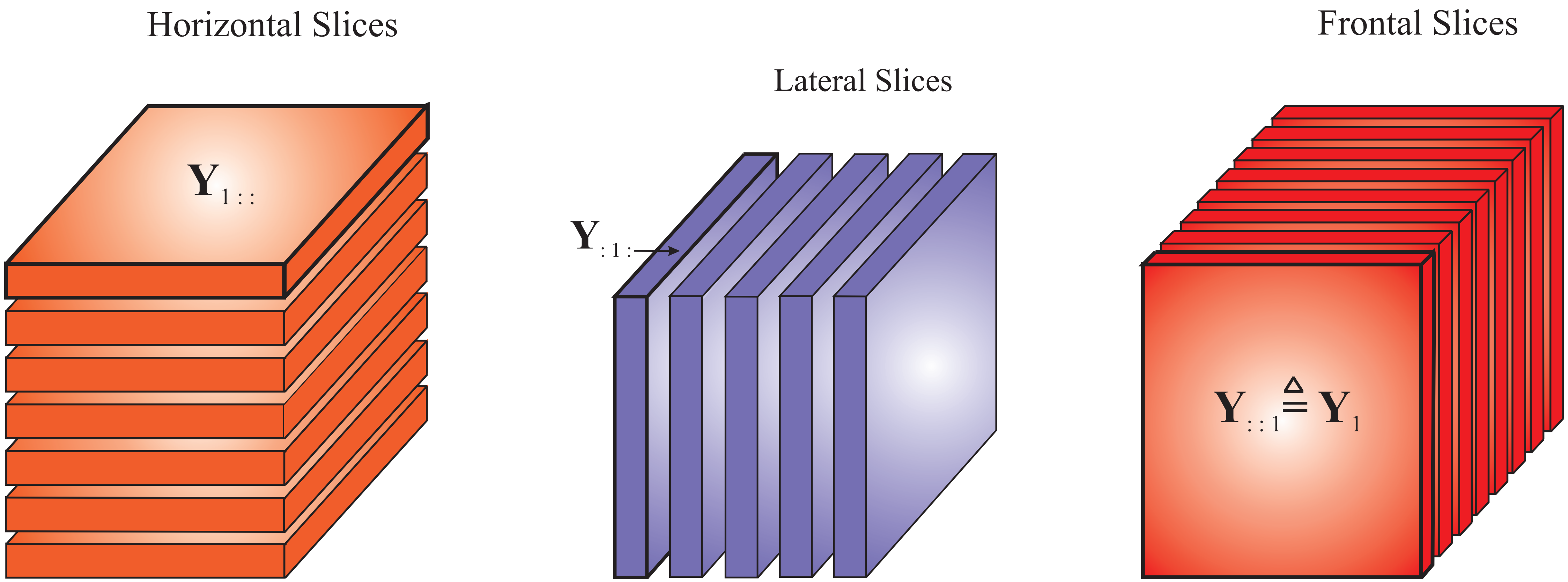
What is tensor?



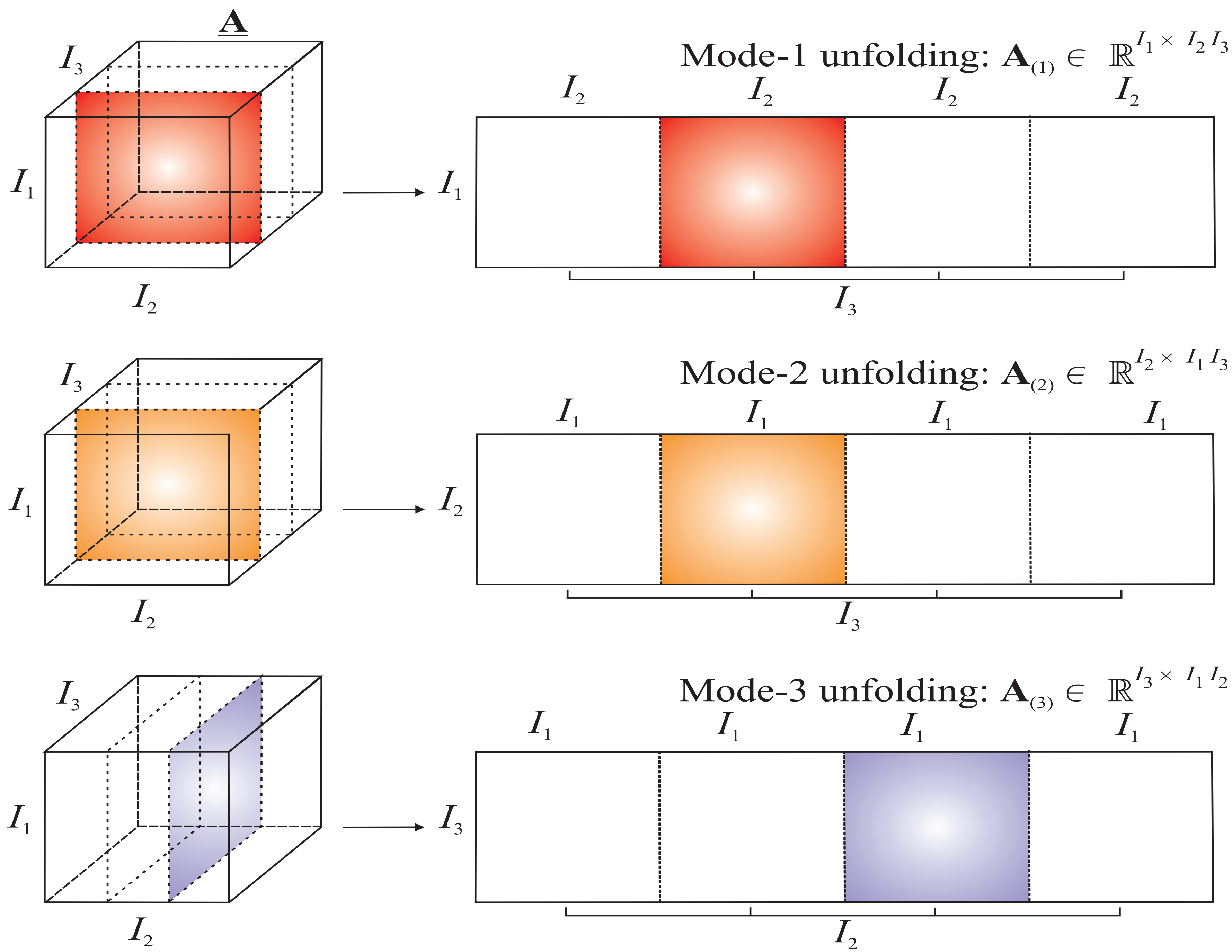
Tensor Fibers



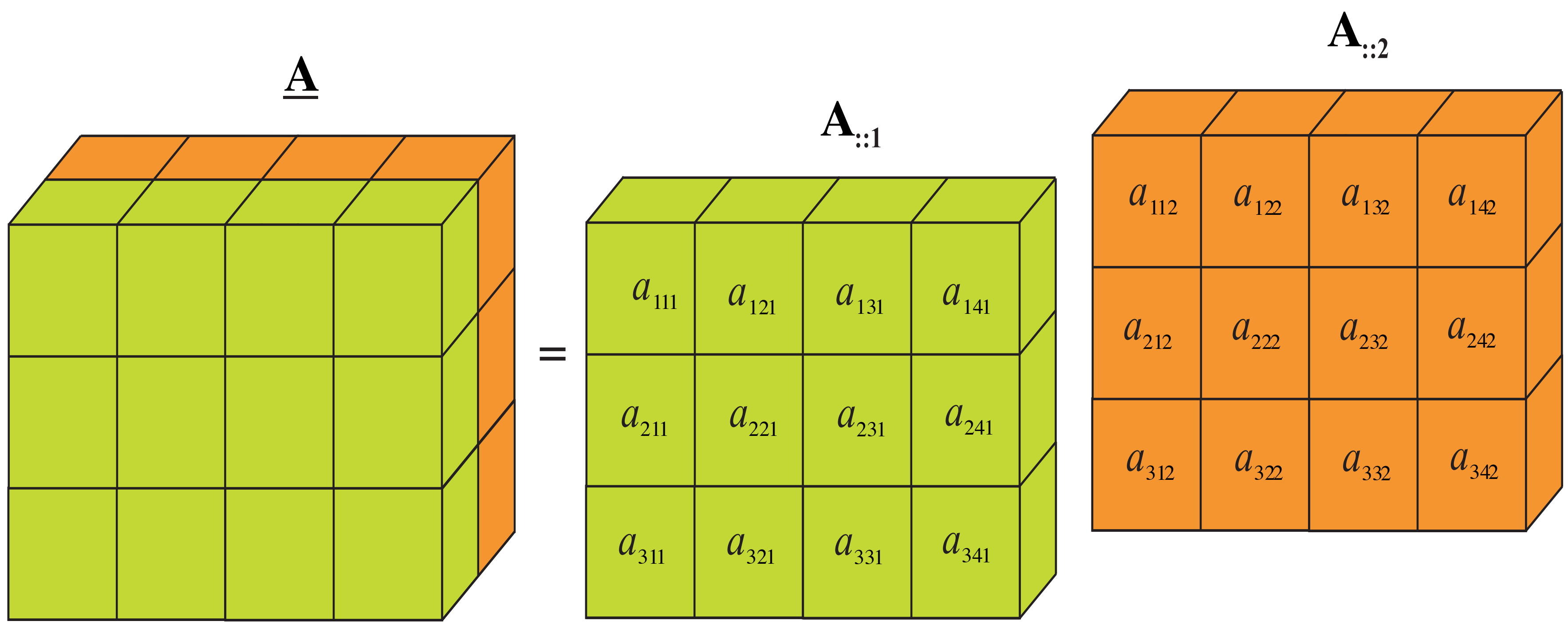
Tensor slices



Tensor Unfolding



An Example of Tensor Unfolding



$$\mathbf{A}_{(1)} = \left[\begin{array}{cccc|cccc} a_{111} & a_{121} & a_{131} & a_{141} & a_{112} & a_{122} & a_{132} & a_{142} \\ a_{211} & a_{221} & a_{231} & a_{241} & a_{212} & a_{222} & a_{232} & a_{242} \\ a_{311} & a_{321} & a_{331} & a_{341} & a_{312} & a_{322} & a_{332} & a_{342} \end{array}\right]$$

$$\mathbf{A}_{(2)} = \left[\begin{array}{ccc|ccc} a_{111} & a_{211} & a_{311} & a_{112} & a_{212} & a_{312} \\ a_{121} & a_{221} & a_{321} & a_{122} & a_{222} & a_{322} \\ a_{131} & a_{231} & a_{331} & a_{132} & a_{232} & a_{332} \\ a_{141} & a_{241} & a_{341} & a_{142} & a_{242} & a_{342} \end{array}\right]$$

$$\mathbf{A}_{(3)} = \left[\begin{array}{ccc|ccc|ccc|ccc} a_{111} & a_{211} & a_{311} & a_{121} & a_{221} & a_{321} & a_{131} & a_{231} & a_{331} & a_{141} & a_{241} & a_{341} \\ a_{112} & a_{212} & a_{312} & a_{122} & a_{222} & a_{322} & a_{132} & a_{232} & a_{332} & a_{142} & a_{242} & a_{342} \end{array}\right]$$

Matrix Products

Matrix Outer Product:

The outer product of the tensors $\underline{\mathbf{Y}} \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_N}$ and $\underline{\mathbf{X}} \in \mathbb{R}^{J_1 \times J_2 \times \cdots \times J_M}$ is given by

$$\underline{\mathbf{Z}} = \mathbf{Y} \circ \mathbf{X} \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_N \times J_1 \times J_2 \times \cdots \times J_M}, \tag{1.75}$$

where

$$z_{i_1, i_2, \ldots, i_N, j_1, j_2, \ldots, j_M} = y_{i_1, i_2, \ldots, i_N} \; x_{j_1, j_2, \ldots, j_M}. \tag{1.76}$$

Matrix Kronecker Product:

The Kronecker product of two matrices $\mathbf{A} \in \mathbb{R}^{I \times J}$ and $\mathbf{B} \in \mathbb{R}^{T \times R}$ is a matrix denoted as $\mathbf{A} \otimes \mathbf{B} \in \mathbb{R}^{IT \times JR}$ and defined as (see the MATLAB function `kron`):

$$\mathbf{A} \otimes \mathbf{B} = \begin{bmatrix} a_{11}\,\mathbf{B} & a_{12}\,\mathbf{B} & \cdots & a_{1J}\,\mathbf{B} \\ a_{21}\,\mathbf{B} & a_{22}\,\mathbf{B} & \cdots & a_{2J}\,\mathbf{B} \\ \vdots & \vdots & \ddots & \vdots \\ a_{I1}\,\mathbf{B} & a_{I2}\,\mathbf{B} & \cdots & a_{IJ}\,\mathbf{B} \end{bmatrix} \tag{1.80}$$

$$= \left[\boldsymbol{a}_1 \otimes \boldsymbol{b}_1 \;\; \boldsymbol{a}_1 \otimes \boldsymbol{b}_2 \;\; \boldsymbol{a}_1 \otimes \boldsymbol{b}_3 \;\; \cdots \;\; \boldsymbol{a}_J \otimes \boldsymbol{b}_{R-1} \;\; \boldsymbol{a}_J \otimes \boldsymbol{b}_R \right]. \tag{1.81}$$

Matrix Products

Matrix Hadamard Product:

The Hadamard product of two equal-size matrices is the element-wise product denoted by $\circledast$ (or $.*$ for MATLAB notation) and defined as

$$\mathbf{A} \circledast \mathbf{B} = \begin{bmatrix} a_{11}\, b_{11} & a_{12}\, b_{12} & \cdots & a_{1J}\, b_{1J} \\ a_{21}\, b_{21} & a_{22}\, b_{22} & \cdots & a_{2J}\, b_{2J} \\ \vdots & \vdots & \ddots & \vdots \\ a_{I1}\, b_{I1} & a_{I2}\, b_{I2} & \cdots & a_{IJ}\, b_{IJ} \end{bmatrix}. \quad (1.88)$$

Matrix Khatri-Rao Product:

For two matrices $\mathbf{A} = [\boldsymbol{a}_1, \boldsymbol{a}_2, \ldots, \boldsymbol{a}_J] \in \mathbb{R}^{I \times J}$ and $\mathbf{B} = [\boldsymbol{b}_1, \boldsymbol{b}_2, \ldots, \boldsymbol{b}_J] \in \mathbb{R}^{T \times J}$ with the same number of columns J, their Khatri-Rao product, denoted by $\odot$, performs the following operation:

$$\mathbf{A} \odot \mathbf{B} = [\boldsymbol{a}_1 \otimes \boldsymbol{b}_1 \;\; \boldsymbol{a}_2 \otimes \boldsymbol{b}_2 \;\; \cdots \;\; \boldsymbol{a}_J \otimes \boldsymbol{b}_J] \quad (1.89)$$

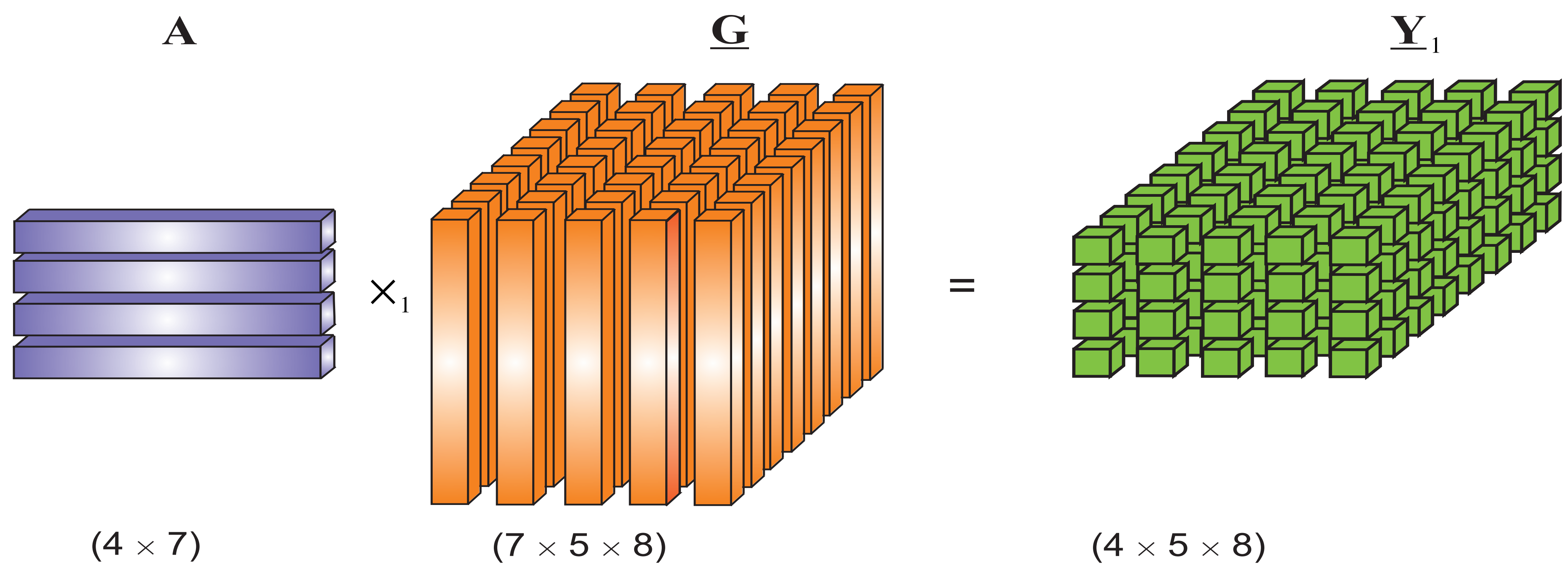
$$= \left[\mathrm{vec}(\boldsymbol{b}_1 \boldsymbol{a}_1^T) \;\; \mathrm{vec}(\boldsymbol{b}_2 \boldsymbol{a}_2^T) \;\; \cdots \;\; \mathrm{vec}(\boldsymbol{b}_J \boldsymbol{a}_J^T)\right] \in \mathbb{R}^{IT \times J}. \quad (1.90)$$

Tensor Matrix Product

Definition 1.5 **(mode-n tensor matrix product)** *The mode-n product* $\underline{\mathbf{Y}} = \underline{\mathbf{G}} \times_n \mathbf{A}$ *of a tensor* $\underline{\mathbf{G}} \in \mathbb{R}^{J_1 \times J_2 \times \cdots \times J_N}$ *and a matrix* $\mathbf{A} \in \mathbb{R}^{I_n \times J_n}$ *is a tensor* $\underline{\mathbf{Y}} \in \mathbb{R}^{J_1 \times \cdots \times J_{n-1} \times I_n \times J_{n+1} \times \cdots \times J_N}$, *with elements*

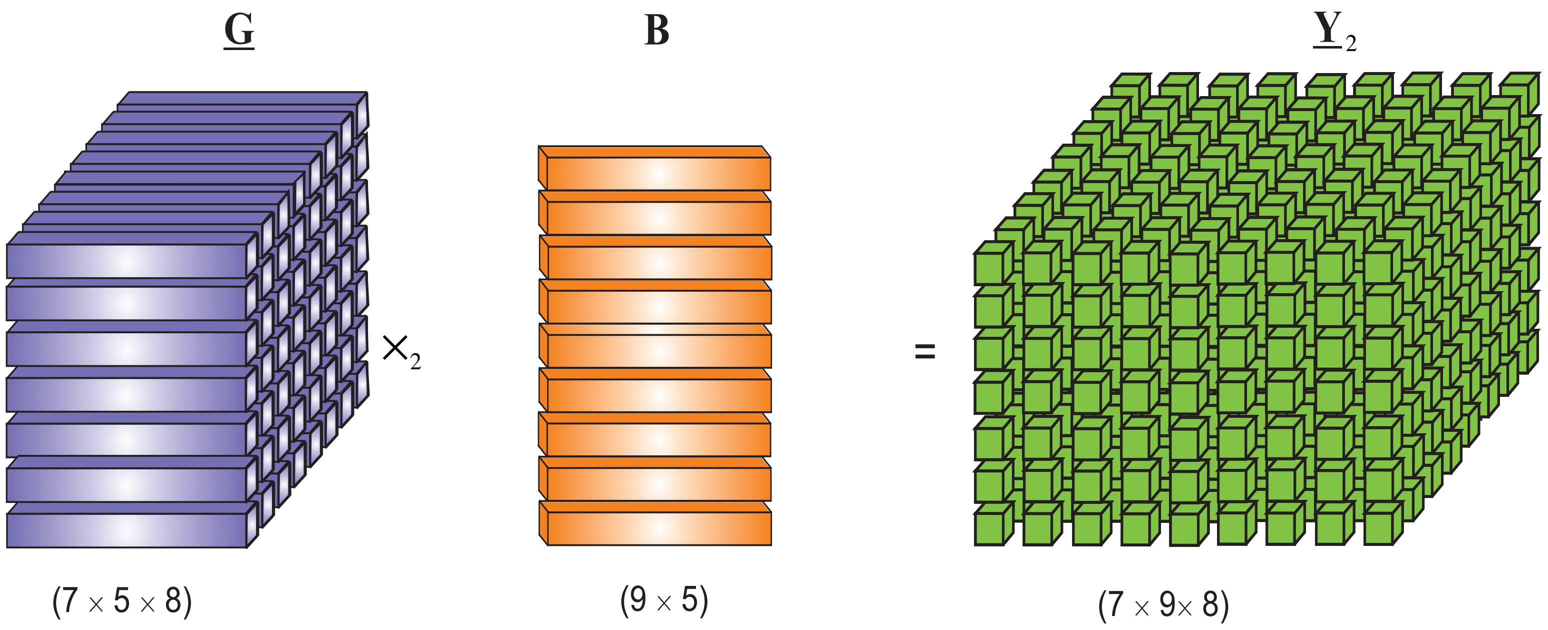
$$y_{j_1, j_2, \ldots, j_{n-1}, i_n, j_{n+1}, \ldots, j_N} = \sum_{j_n=1}^{J_n} g_{j_1, j_2, \ldots, J_N} \, a_{i_n, j_n}. \tag{1.97}$$

(a)

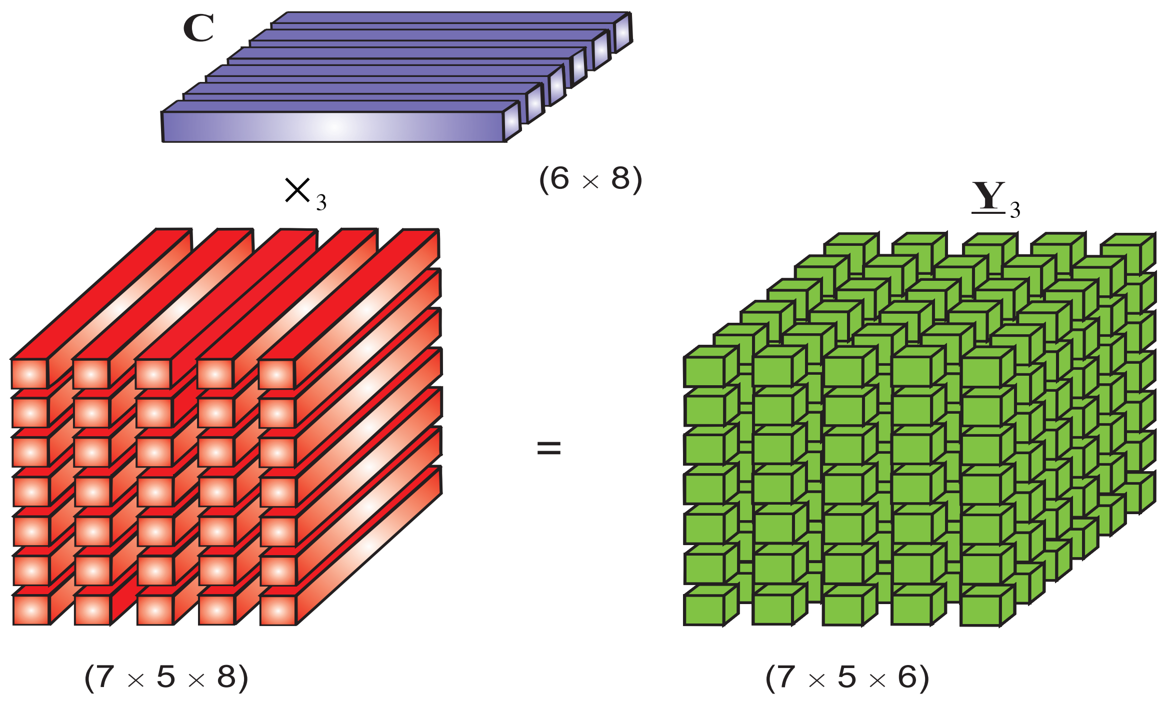


Tensor Matrix Product

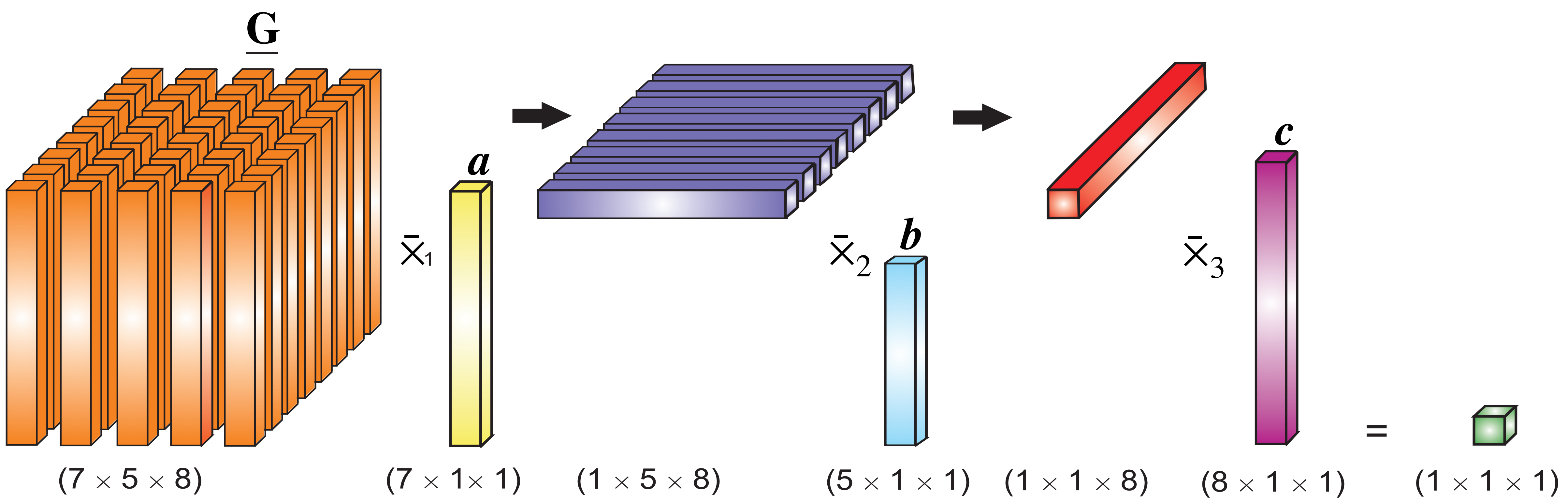
(b)



(c)

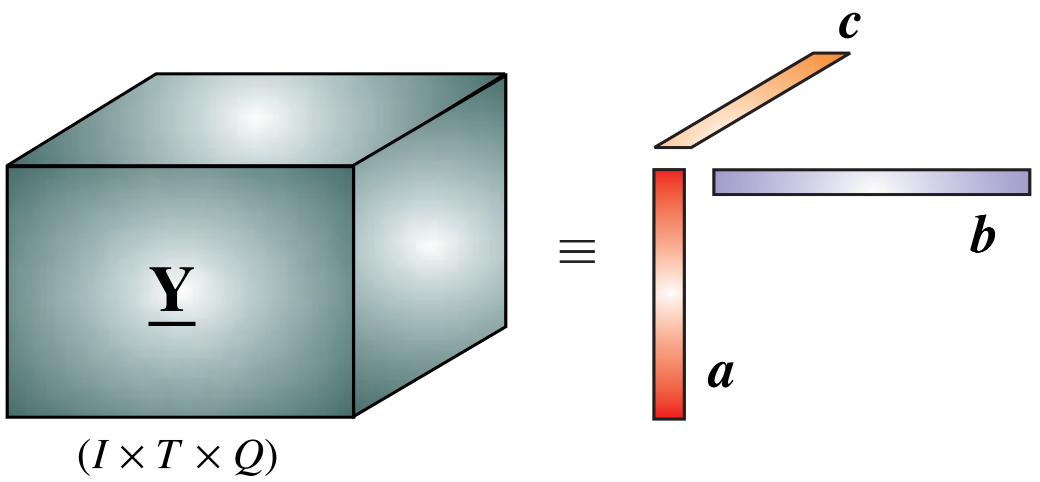


Tensor Vector Contracted Product

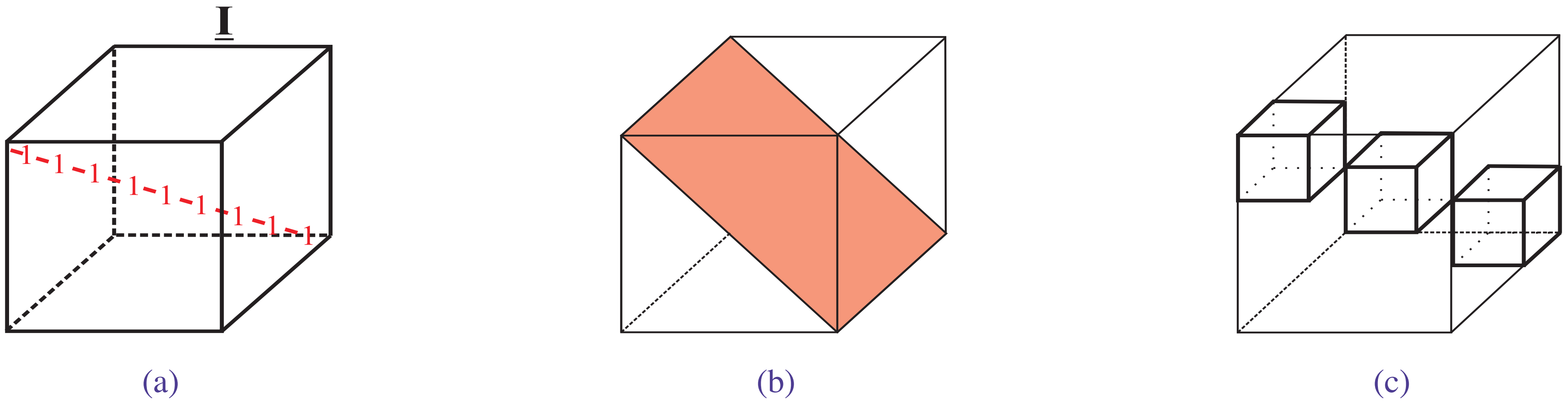


Special Form of Tensors

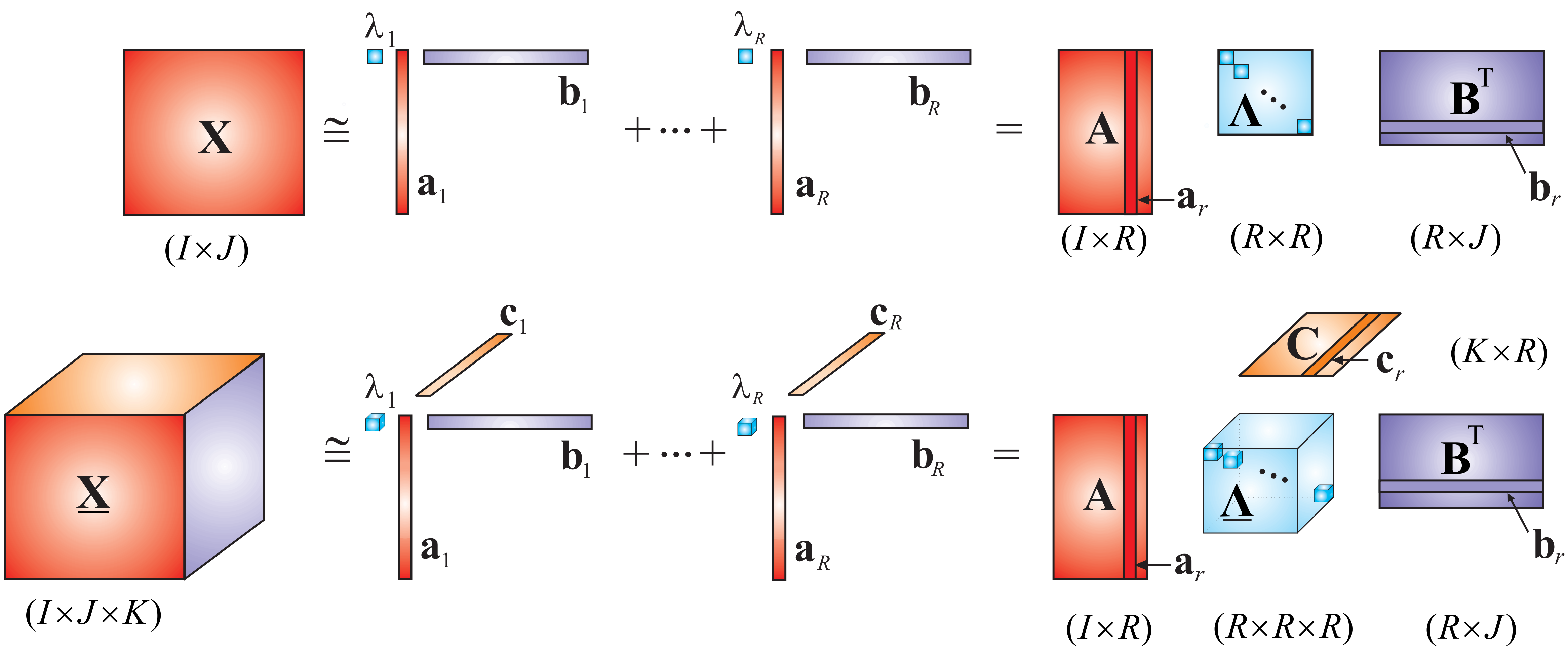
Rank-one tensor:



Examples of tensors with special forms



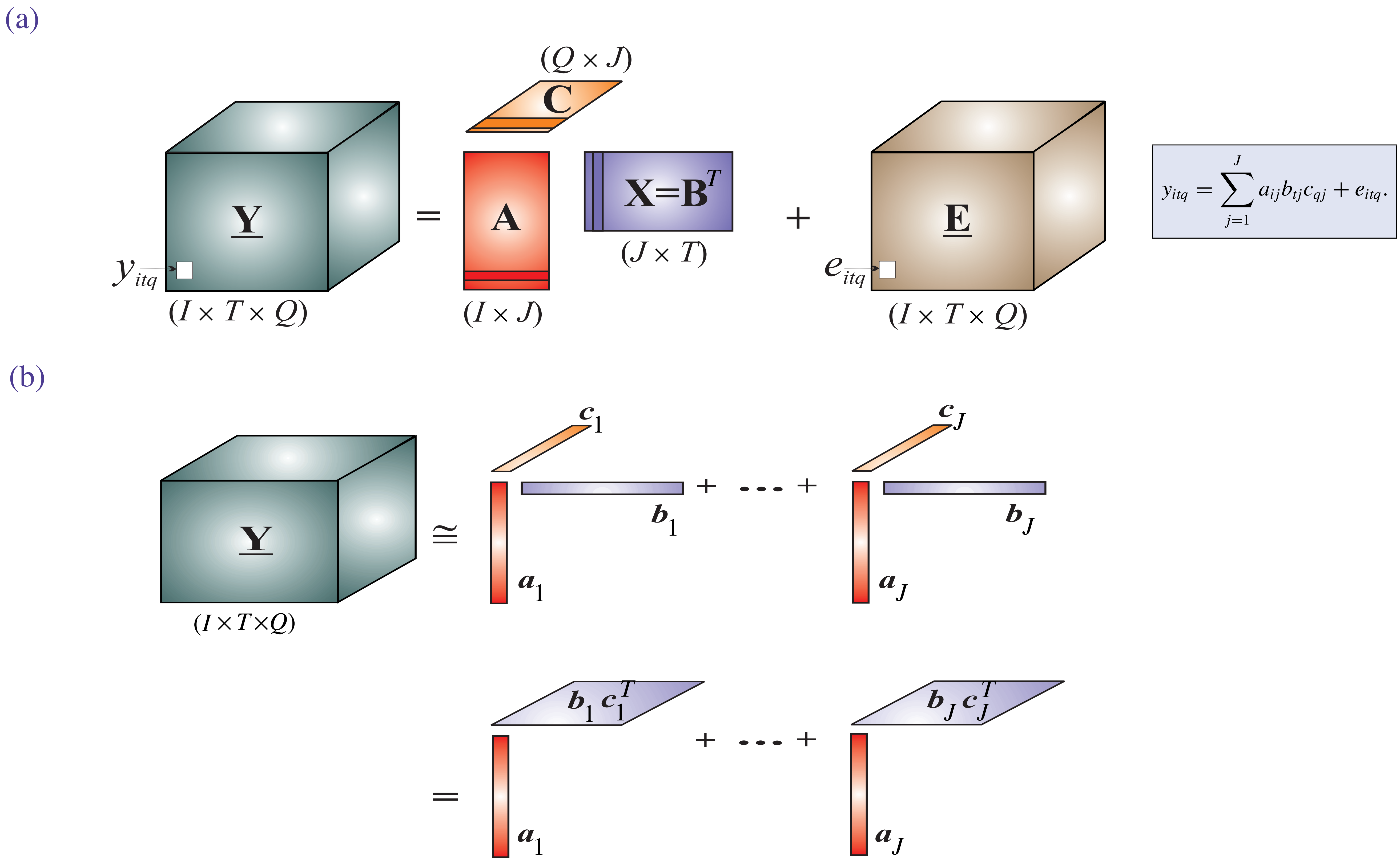
CP Approximation



$$\underline{\mathbf{X}} \cong \sum_{r=1}^{R} \lambda_r \, \mathbf{b}_r^{(1)} \circ \mathbf{b}_r^{(2)} \circ \cdots \circ \mathbf{b}_r^{(N)}$$
$$= \underline{\mathbf{\Lambda}} \times_1 \mathbf{B}^{(1)} \times_2 \mathbf{B}^{(2)} \cdots \times_N \mathbf{B}^{(N)}$$
$$= [\![\underline{\mathbf{\Lambda}}; \mathbf{B}^{(1)}, \mathbf{B}^{(2)}, \ldots, \mathbf{B}^{(N)}]\!],$$

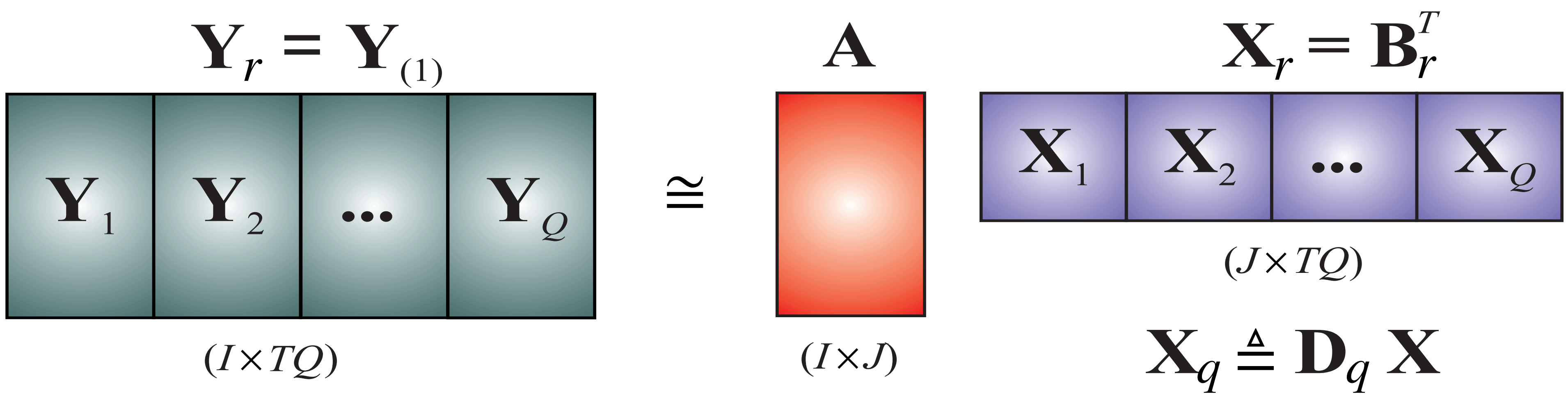
$$\mathbf{X}_{(1)} = \mathbf{A}\mathbf{\Lambda} \left(\mathbf{C} \odot \mathbf{B}\right)^T + \mathbf{E}_{(1)}$$
$$\mathbf{X}_{(2)} = \mathbf{B}\mathbf{\Lambda} \left(\mathbf{C} \odot \mathbf{A}\right)^T + \mathbf{E}_{(2)}$$
$$\mathbf{X}_{(3)} = \mathbf{C}\mathbf{\Lambda} \left(\mathbf{B} \odot \mathbf{A}\right)^T + \mathbf{E}_{(3)}$$

Alternative Representations of CP Decomposition



Alternative Representations of CP Decomposition

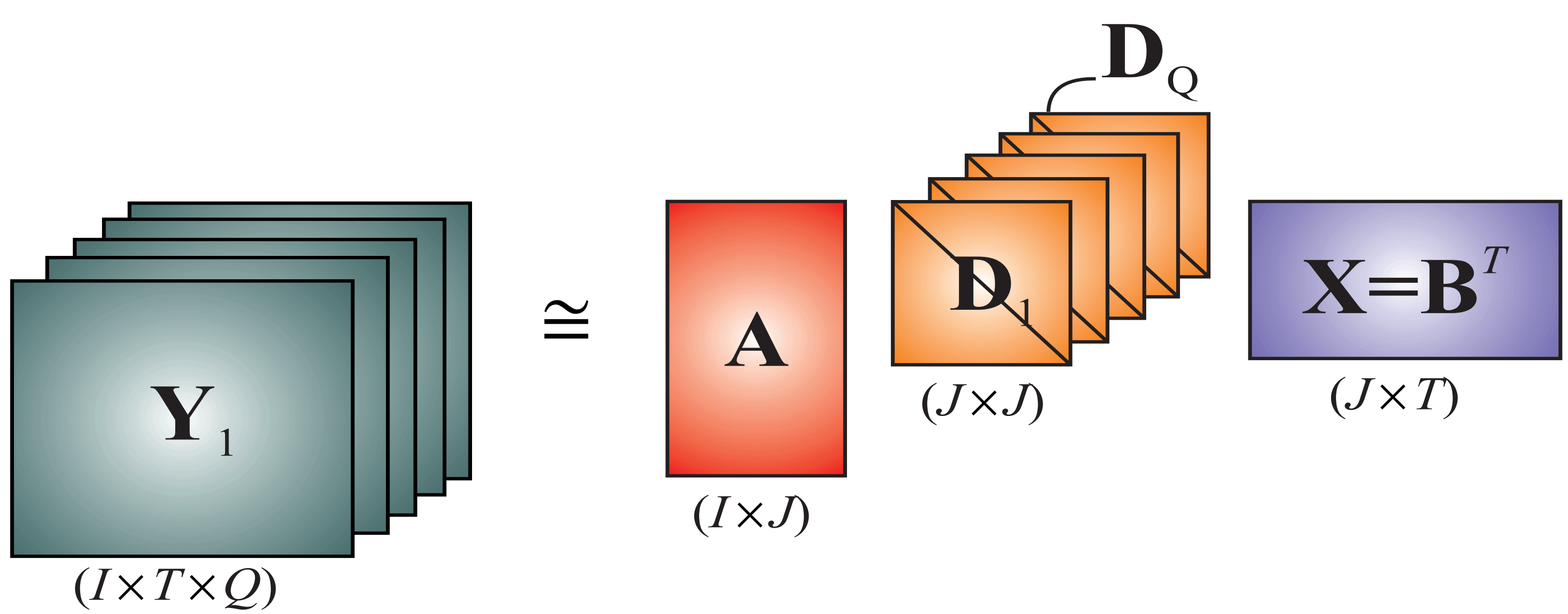
(c)



$$\mathbf{X}_q \triangleq \mathbf{D}_q \, \mathbf{X}$$

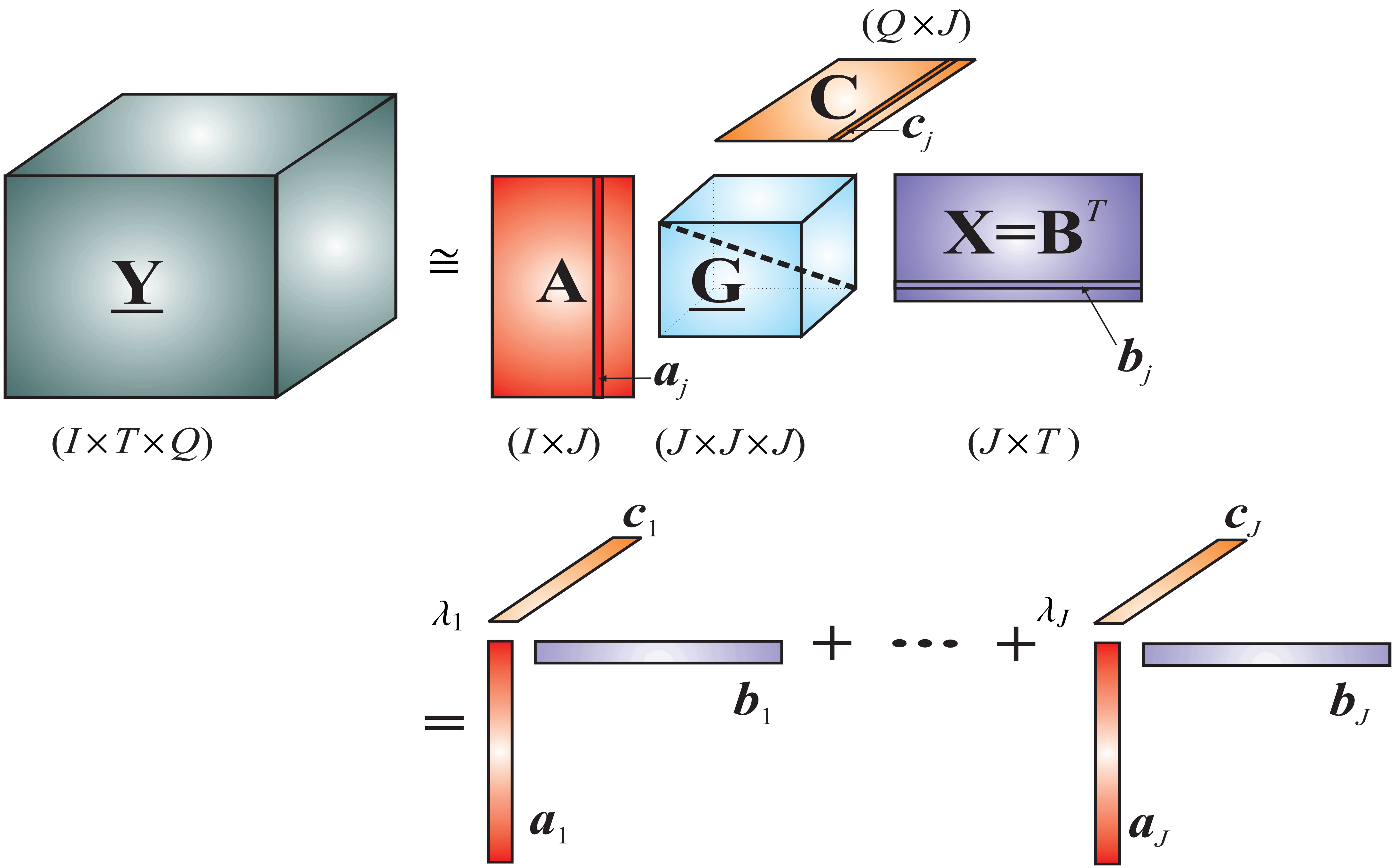
$$(q = 1, 2, \ldots, Q)$$

(d)



$$\mathbf{Y}_q = \mathbf{A}\mathbf{D}_q\mathbf{X}, \quad (q = 1, 2, \ldots, Q)$$

Alternative Representations of CP Decomposition



CP Approximation

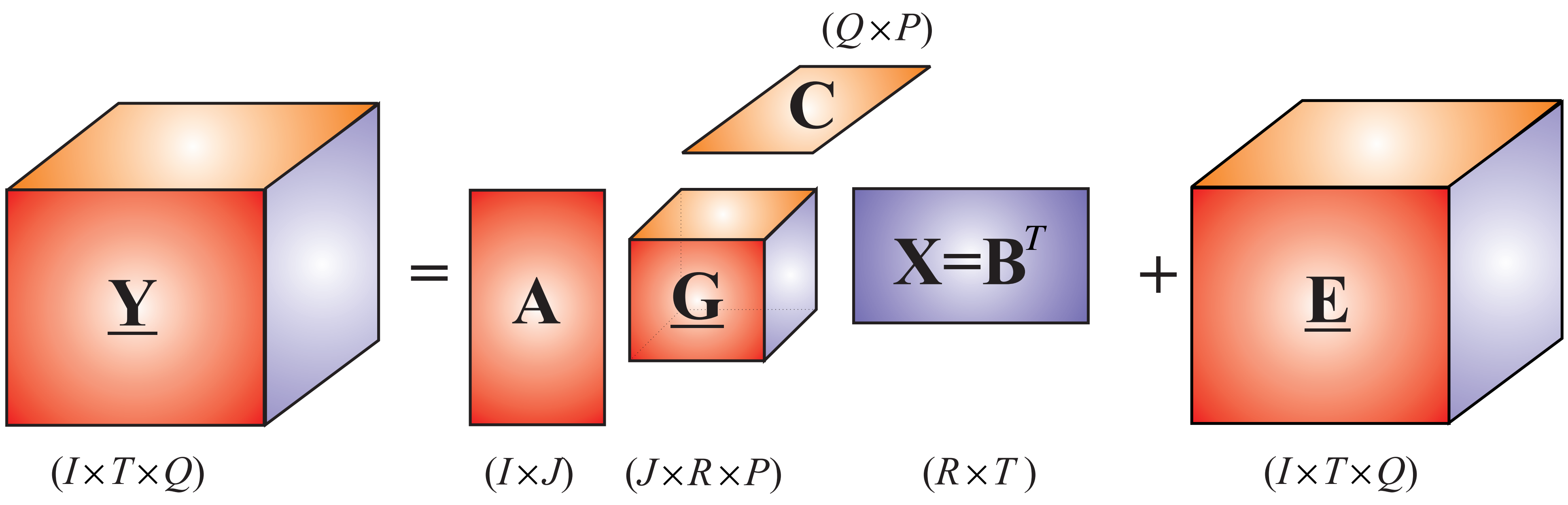
Algorithm 1: Basic ALS for the CP decomposition of a 3rd-order tensor

Input: Data tensor $\underline{\mathbf{X}} \in \mathbb{R}^{I \times J \times K}$ and rank R

Output: Factor matrices $\mathbf{A} \in \mathbb{R}^{I \times R}$, $\mathbf{B} \in \mathbb{R}^{J \times R}$, $\mathbf{C} \in \mathbb{R}^{K \times R}$, and scaling vector $\boldsymbol{\lambda} \in \mathbb{R}^{R}$

1: Initialize $\mathbf{A}, \mathbf{B}, \mathbf{C}$
2: **while** not converged or iteration limit is not reached **do**
3: $\quad \mathbf{A} \leftarrow \mathbf{X}_{(1)}(\mathbf{C} \odot \mathbf{B})(\mathbf{C}^{\mathrm{T}}\mathbf{C} \circledast \mathbf{B}^{\mathrm{T}}\mathbf{B})^{\dagger}$
4: $\quad$ Normalize column vectors of $\mathbf{A}$ to unit length (by computing the norm of each column vector and dividing each element of a vector by its norm)
5: $\quad \mathbf{B} \leftarrow \mathbf{X}_{(2)}(\mathbf{C} \odot \mathbf{A})(\mathbf{C}^{\mathrm{T}}\mathbf{C} \circledast \mathbf{A}^{\mathrm{T}}\mathbf{A})^{\dagger}$
6: $\quad$ Normalize column vectors of $\mathbf{B}$ to unit length
7: $\quad \mathbf{C} \leftarrow \mathbf{X}_{(3)}(\mathbf{B} \odot \mathbf{A})(\mathbf{B}^{\mathrm{T}}\mathbf{B} \circledast \mathbf{C}^{\mathrm{T}}\mathbf{C})^{\dagger}$
8: $\quad$ Normalize column vectors of $\mathbf{C}$ to unit length, store the norms in vector $\boldsymbol{\lambda}$
9: **end while**
10: **return** $\mathbf{A}, \mathbf{B}, \mathbf{C}$ and $\boldsymbol{\lambda}$.

Tucker Approximation



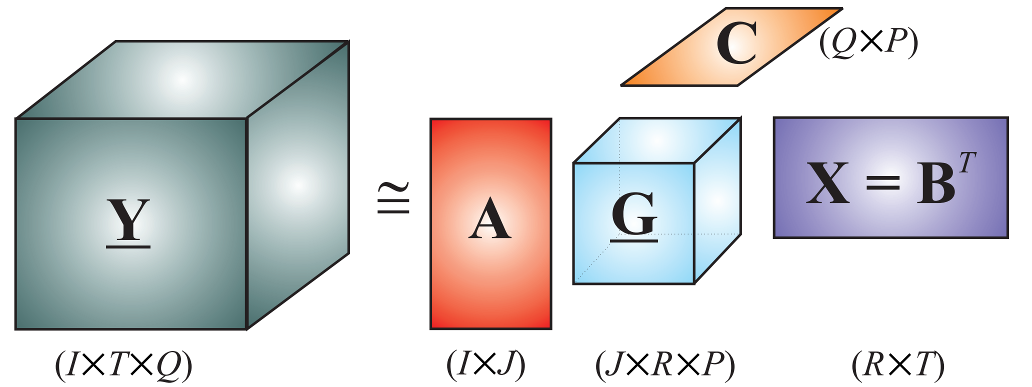
$$\underline{\mathbf{Y}} = \underline{\mathbf{G}} \times_1 \mathbf{A} \times_2 \mathbf{B} \times_3 \mathbf{C} + \underline{\mathbf{E}} = [\![\underline{\mathbf{G}}; \mathbf{A}, \mathbf{B}, \mathbf{C}]\!] + \underline{\mathbf{E}},$$

Matrix Form of Tucker Decomposition:

$$\mathbf{X}_{(1)} \approx \mathbf{A}\mathbf{G}_{(1)}(\mathbf{C} \otimes \mathbf{B})^{\mathsf{T}},$$
$$\mathbf{X}_{(2)} \approx \mathbf{B}\mathbf{G}_{(2)}(\mathbf{C} \otimes \mathbf{A})^{\mathsf{T}},$$
$$\mathbf{X}_{(3)} \approx \mathbf{C}\mathbf{G}_{(3)}(\mathbf{B} \otimes \mathbf{A})^{\mathsf{T}}.$$

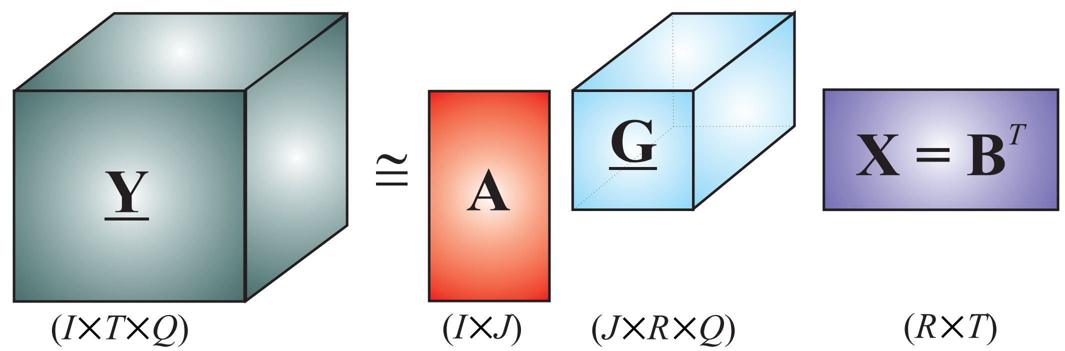
Different Types of Tucker Decomposition

$$y_{itq} = \sum_{j=1}^{J} \sum_{r=1}^{R} \sum_{p=1}^{P} g_{jrp}\, a_{ij}\, b_{tr}\, c_{qp}$$



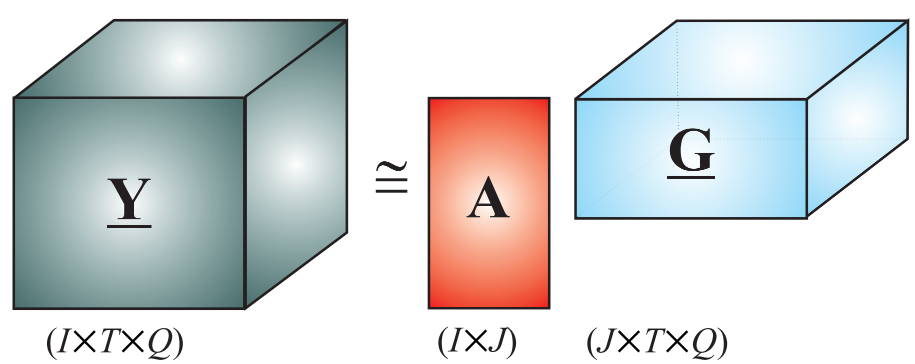
(a) Tucker3

$$y_{itq} = \sum_{j=1}^{J} \sum_{r=1}^{R} g_{jrq}\, a_{ij}\, b_{tr}$$



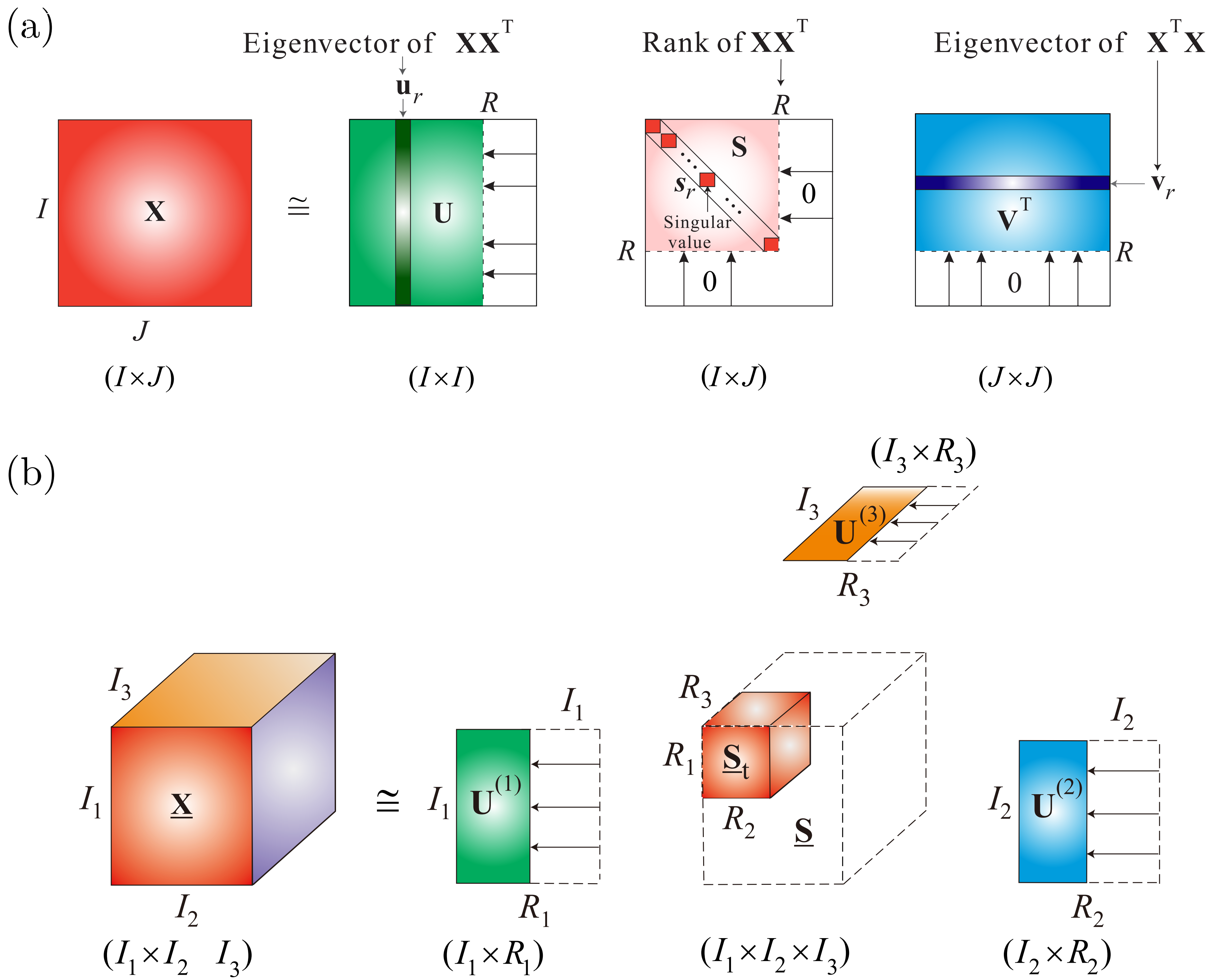
(b) Tucker2

$$y_{itq} = \sum_{j=1}^{J} g_{jtq}\, a_{ij}$$



(c) Tucker1

From Matrix SVD to Higher-order Case



HOSVD

Algorithm 2: Sequentially Truncated HOSVD (Vannieuwenhoven *et al.*, 2012)

Input: Nth-order tensor $\underline{\mathbf{X}} \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_N}$ and approximation accuracy ε

Output: HOSVD in the Tucker format $\hat{\underline{\mathbf{X}}} = [\![\underline{\mathbf{S}}; \mathbf{U}^{(1)}, \ldots, \underline{\mathbf{U}}^{(N)}]\!]$, such that $\|\underline{\mathbf{X}} - \hat{\underline{\mathbf{X}}}\|_F \leqslant \varepsilon$

```
S ← X
for n = 1 to N do
    [U^(n), S, V] = truncated_svd(S_(n), ε/√N)
    S ← VS
end for
S ← reshape(S, [R_1, ..., R_N])
return Core tensor S and orthogonal factor matrices U^(n) ∈ R^(I_n × R_n).
```

$\underline{\mathbf{S}} \leftarrow \underline{\mathbf{X}}$

for $n = 1$ to N **do**

$[\mathbf{U}^{(n)}, \mathbf{S}, \mathbf{V}] = \texttt{truncated\_svd}(\mathbf{S}_{(n)}, \frac{\varepsilon}{\sqrt{N}})$

$\underline{\mathbf{S}} \leftarrow \mathbf{VS}$

end for

$\underline{\mathbf{S}} \leftarrow \texttt{reshape}(\underline{\mathbf{S}}, [R_1, \ldots, R_N])$

return Core tensor $\underline{\mathbf{S}}$ and orthogonal factor matrices $\mathbf{U}^{(n)} \in \mathbb{R}^{I_n \times R_n}$.

rSVD

Algorithm 3: Randomized SVD (rSVD) for large-scale and low-rank matrices with single sketch (Halko *et al.*, 2011)

Input: A matrix $\mathbf{X} \in \mathbb{R}^{I \times J}$, desired or estimated rank R, and oversampling parameter P or overestimated rank $\tilde{R} = R + P$, exponent of the power method q ($q = 0$ or $q = 1$)

Output: An approximate rank-$\tilde{R}$ SVD, $\mathbf{X} \cong \mathbf{U}\mathbf{S}\mathbf{V}^{\mathrm{T}}$, i.e., orthogonal matrices $\mathbf{U} \in \mathbb{R}^{I \times \tilde{R}}$, $\mathbf{V} \in \mathbb{R}^{J \times \tilde{R}}$ and diagonal matrix $\mathbf{S} \in \mathbb{R}^{\tilde{R} \times \tilde{R}}$ with singular values

1: Draw a random Gaussian matrix $\boldsymbol{\Omega} \in \mathbb{R}^{J \times \tilde{R}}$,
2: Form the sample matrix $\mathbf{Y} = (\mathbf{X}\mathbf{X}^{\mathrm{T}})^q \, \mathbf{X}\boldsymbol{\Omega} \in \mathbb{R}^{I \times \tilde{R}}$
3: Compute a QR decomposition $\mathbf{Y} = \mathbf{Q}\mathbf{R}$
4: Form the matrix $\mathbf{A} = \mathbf{Q}^{\mathrm{T}}\mathbf{X} \in \mathbb{R}^{\tilde{R} \times J}$
5: Compute the SVD of the small matrix $\mathbf{A}$ as $\mathbf{A} = \hat{\mathbf{U}}\mathbf{S}\mathbf{V}^{\mathrm{T}}$
6: Form the matrix $\mathbf{U} = \mathbf{Q}\hat{\mathbf{U}}$.

HOOI

Algorithm 4: Higher Order Orthogonal Iteration (HOOI) (De Lathauwer *et al.*, 2000b; Austin *et al.*, 2015)

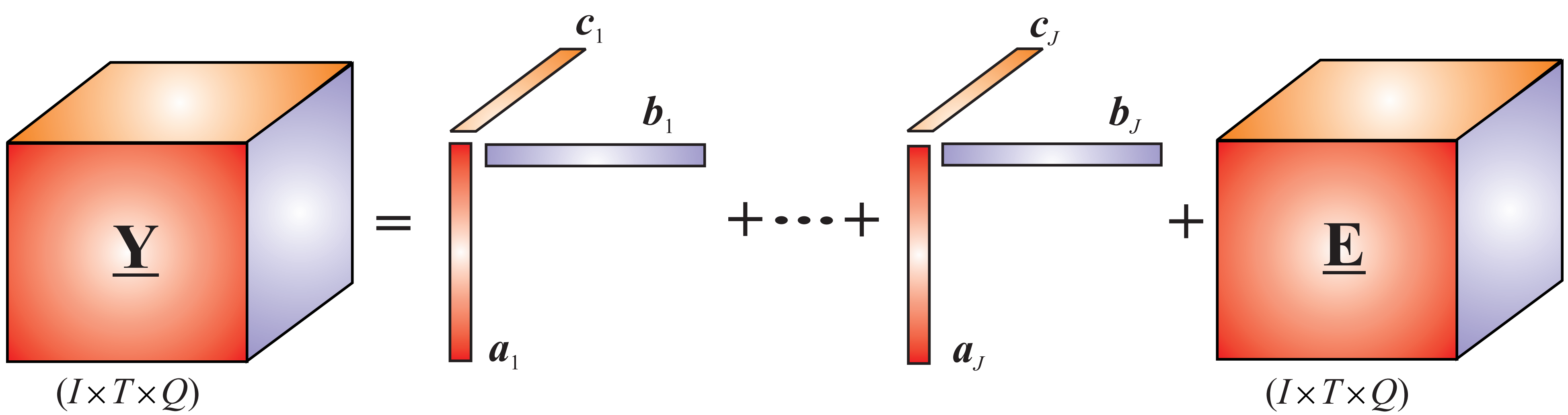
Input: Nth-order tensor $\underline{\mathbf{X}} \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_N}$ (usually in Tucker/HOSVD format)

Output: Improved Tucker approximation using ALS approach, with orthogonal factor matrices $\mathbf{U}^{(n)}$

```
Initialization via the standard HOSVD (see Algorithm 2)
repeat
    for n = 1 to N do
        Z ← X ×_{p≠n} {U^{(p) T}}
        C ← Z_(n) Z_(n)^T ∈ R^{R×R}
        U^(n) ← leading R_n eigenvectors of C
    end for
    G ← Z ×_N U^{(N) T}
until the cost function (‖X‖_F^2 − ‖G‖_F^2) ceases to decrease
return ⟦G; U^(1), U^(2), ..., U^(N)⟧
```

- Initialization via the standard HOSVD (see Algorithm 2)
- **repeat**
  - **for** $n = 1$ to N **do**
    - $\underline{\mathbf{Z}} \leftarrow \underline{\mathbf{X}} \times_{p \neq n} \{\mathbf{U}^{(p)\,\mathrm{T}}\}$
    - $\mathbf{C} \leftarrow \mathbf{Z}_{(n)} \mathbf{Z}_{(n)}^{\mathrm{T}} \in \mathbb{R}^{R \times R}$
    - $\mathbf{U}^{(n)} \leftarrow$ leading R_n eigenvectors of $\mathbf{C}$
  - **end for**
  - $\underline{\mathbf{G}} \leftarrow \underline{\mathbf{Z}} \times_N \mathbf{U}^{(N)\,\mathrm{T}}$
- **until** the cost function $(\|\underline{\mathbf{X}}\|_F^2 - \|\underline{\mathbf{G}}\|_F^2)$ ceases to decrease
- **return** $[\![\underline{\mathbf{G}}; \mathbf{U}^{(1)}, \mathbf{U}^{(2)}, \ldots, \mathbf{U}^{(N)}]\!]$

Nonnegative Tensor Factorization



Definition (NTF). *Given an N-th order tensor* $\underline{\mathbf{Y}} \in \mathbb{R}^{I_1 \times I_2 \times \ldots \times I_N}$ *and a positive integer* J, *factorize* $\underline{\mathbf{Y}}$ *into a set of* N *nonnegative component matrices* $\mathbf{A}^{(n)} = [\mathbf{a}_1^{(n)}, \mathbf{a}_2^{(n)}, \ldots, \mathbf{a}_J^{(n)}] \in \mathbb{R}^{I_n \times J}$, $(n = 1, 2, \ldots, N)$ *representing the common (loading) factors, that is,*

$$\underline{\mathbf{Y}} = \hat{\underline{\mathbf{Y}}} + \underline{\mathbf{E}} = \sum_{j=1}^{J} \mathbf{a}_j^{(1)} \circ \mathbf{a}_j^{(2)} \circ \ldots \circ \mathbf{a}_j^{(N)} + \underline{\mathbf{E}} =$$

$$\underline{\mathbf{I}} \times_1 \mathbf{A}^{(1)} \times_2 \mathbf{A}^{(2)} \cdots \times_N \mathbf{A}^{(N)} + \underline{\mathbf{E}} = [\![\mathbf{A}^{(1)}, \mathbf{A}^{(2)}, \ldots, \mathbf{A}^{(N)}]\!] + \underline{\mathbf{E}}$$

with $||\mathbf{a}_j^{(n)}||_2 = 1$ *for* $n = 1, 2, \ldots N-1$ *and* $j = 1, 2, \ldots, J$.

Matrix Nonnegative Least-Squares (MNLS)

Algorithm 2: Nesterov-type algorithm for MNLS

Input: $\mathbf{X} \in \mathbb{R}^{m \times k}$, $\mathbf{B} \in \mathbb{R}^{k \times n}$, $\mathbf{A}_0 \in \mathbb{R}^{m \times n}$, tol > 0.

Compute $\mathbf{W} = -\mathbf{X}\mathbf{B}$, $\mathbf{Z} = \mathbf{B}^T\mathbf{B}$.

Compute $L = \max(\text{eig}(\mathbf{Z}))$ $\mu = \min(\text{eig}(\mathbf{Z}))$.

Set $\mathbf{Y}_0 = \mathbf{A}_0$, $\beta = \frac{\sqrt{L}-\sqrt{\mu}}{\sqrt{L}+\sqrt{\mu}}$, $k = 0$.

while (1) **do**

- $\nabla f(\mathbf{Y}_k) = \mathbf{W} + \mathbf{A}_k\mathbf{Z}$;
- **if** $(\max(|\nabla f(\mathbf{Y}_k) \circledast \mathbf{Y}_k|) < \text{tol})$ **then**
  - break;
- **else**
  - $\mathbf{A}_{k+1} = \left[\mathbf{Y}_k - \frac{1}{L}\nabla f(\mathbf{Y}_k)\right]_+$;
  - $\mathbf{Y}_{k+1} = \mathbf{A}_{k+1} + \beta\,(\mathbf{A}_{k+1} - \mathbf{A}_k)$;
  - $k = k + 1$;

return $\mathbf{A}_k$.

The Algorithm

The objective function:
$$f_{\mathcal{X}}(\mathbf{A},\mathbf{B},\mathbf{C}) = \frac{1}{2}\left\|\mathbf{X}_{\mathbf{A}} - \mathbf{A}\left(\mathbf{C}\odot\mathbf{B}\right)^T\right\|_F^2 = \frac{1}{2}\left\|\mathbf{X}_{\mathbf{B}} - \mathbf{B}\left(\mathbf{C}\odot\mathbf{A}\right)^T\right\|_F^2 = \frac{1}{2}\left\|\mathbf{X}_{\mathbf{C}} - \mathbf{C}\left(\mathbf{B}\odot\mathbf{A}\right)^T\right\|_F^2.$$

Algorithm 4: Nesterov-based AO NTF

Input: $\boldsymbol{\mathcal{X}}$, $\mathbf{A}_0 \geq \mathbf{0}$, $\mathbf{B}_0 \geq \mathbf{0}$, $\mathbf{C}_0 \geq \mathbf{0}$, λ, tol.

```
Set k = 0
while (terminating condition is FALSE) do
    W_A = -X_A(C_k ⊙ B_k) - λA_k, Z_A = (C_k ⊙ B_k)^T (C_k ⊙ B_k) + λI
    A_{k+1} = Nesterov_MNLS(W_A, Z_A, A_k, λ, tol)
    W_B = -X_B(C_k ⊙ A_{k+1}) - λB_k, Z_B = (C_k ⊙ A_{k+1})^T (C_k ⊙ A_{k+1}) + λI
    B_{k+1} = Nesterov_MNLS(W_B, Z_B, B_k, λ, tol)
    W_C = -X_C(A_{k+1} ⊙ B_{k+1}) - λC_k, Z_C = (A_{k+1} ⊙ B_{k+1})^T (A_{k+1} ⊙ B_{k+1}) + λI
    C_{k+1} = Nesterov_MNLS(W_C, Z_C, C_k, λ, tol)
    (A_{k+1}, B_{k+1}, C_{k+1}) = Normalize(A_{k+1}, B_{k+1}, C_{k+1})
    (A_{k+1}, B_{k+1}, C_{k+1}) = Accelerate(A_{k+1}, A_k, B_{k+1}, B_k, C_{k+1}, C_k, k)
    k = k + 1
return A_k, B_k, C_k.
```