Tensor Methods for Signal Processing and Machine Learning

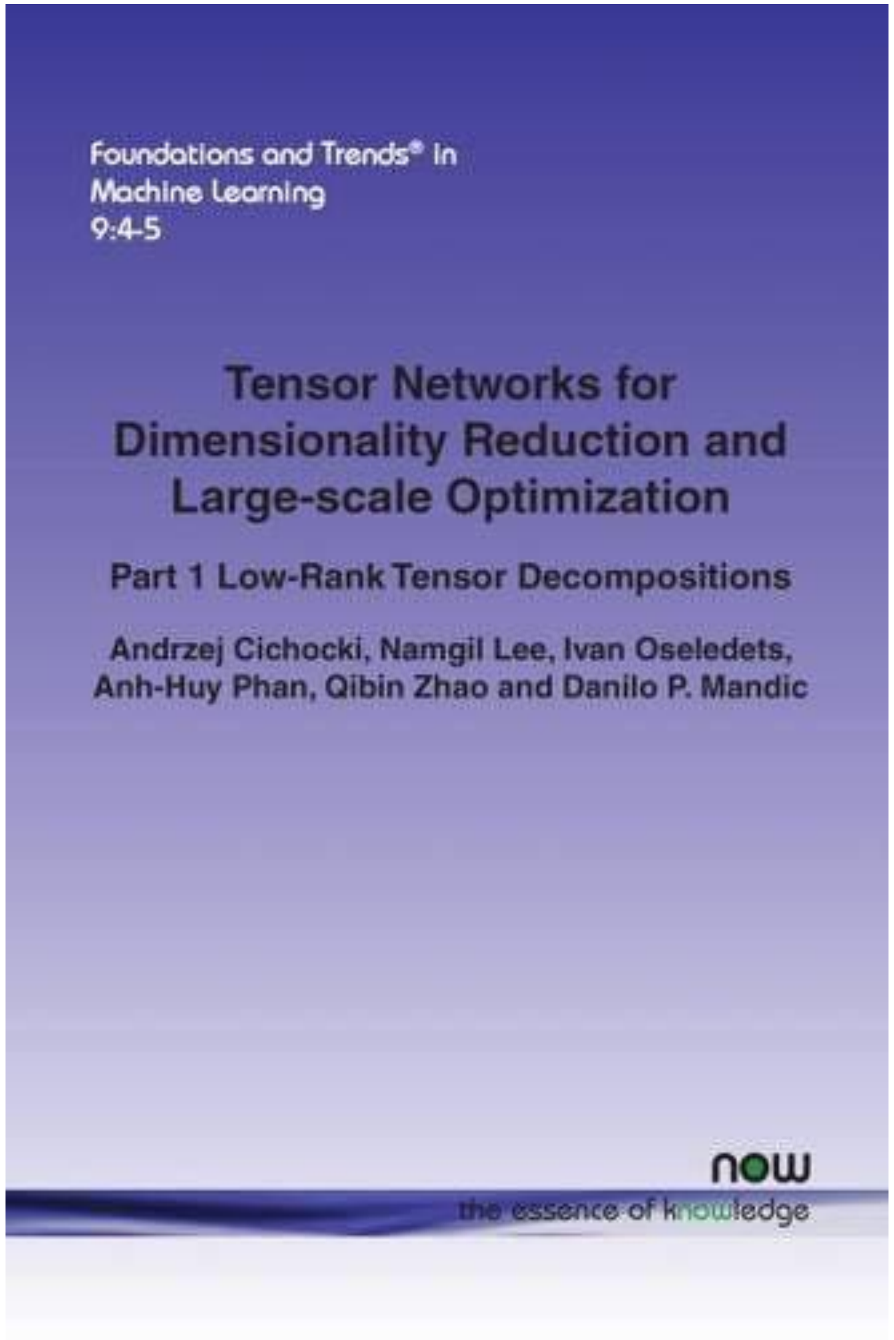
Qibin Zhao
Tensor Learning Unit
RIKEN AIP

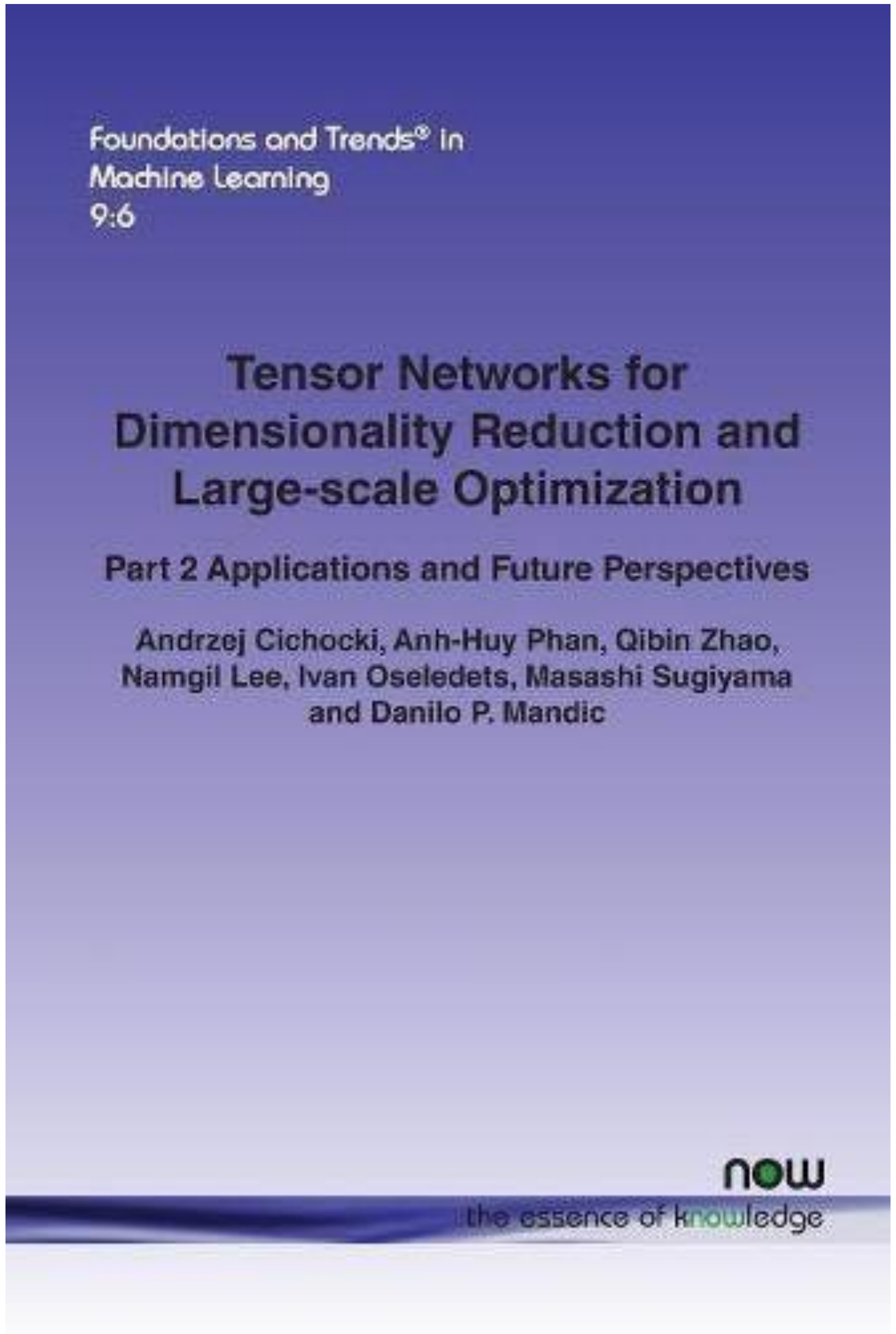
2018-6-9 @ Waseda University

Monographs

Tensor networks for dimensionality reduction and large optimization

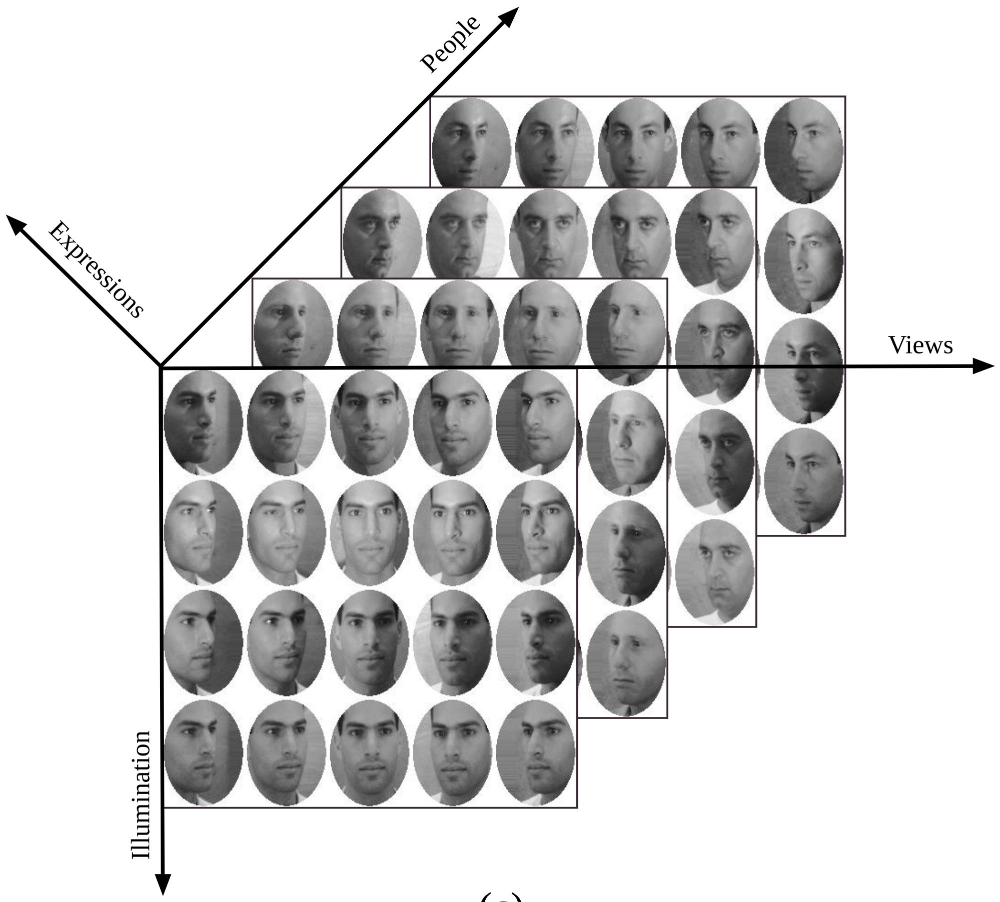
Andrzej Cichocki, Namgil Lee, Ivan Oseledets, Anh-Huy Phan, **Qibin Zhao** and Danilo P.Mandic



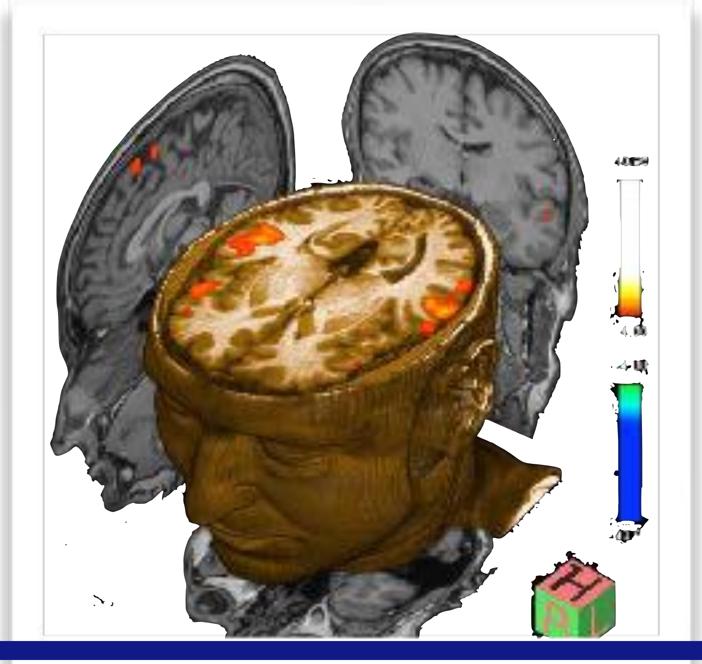


Multidimensional structured data

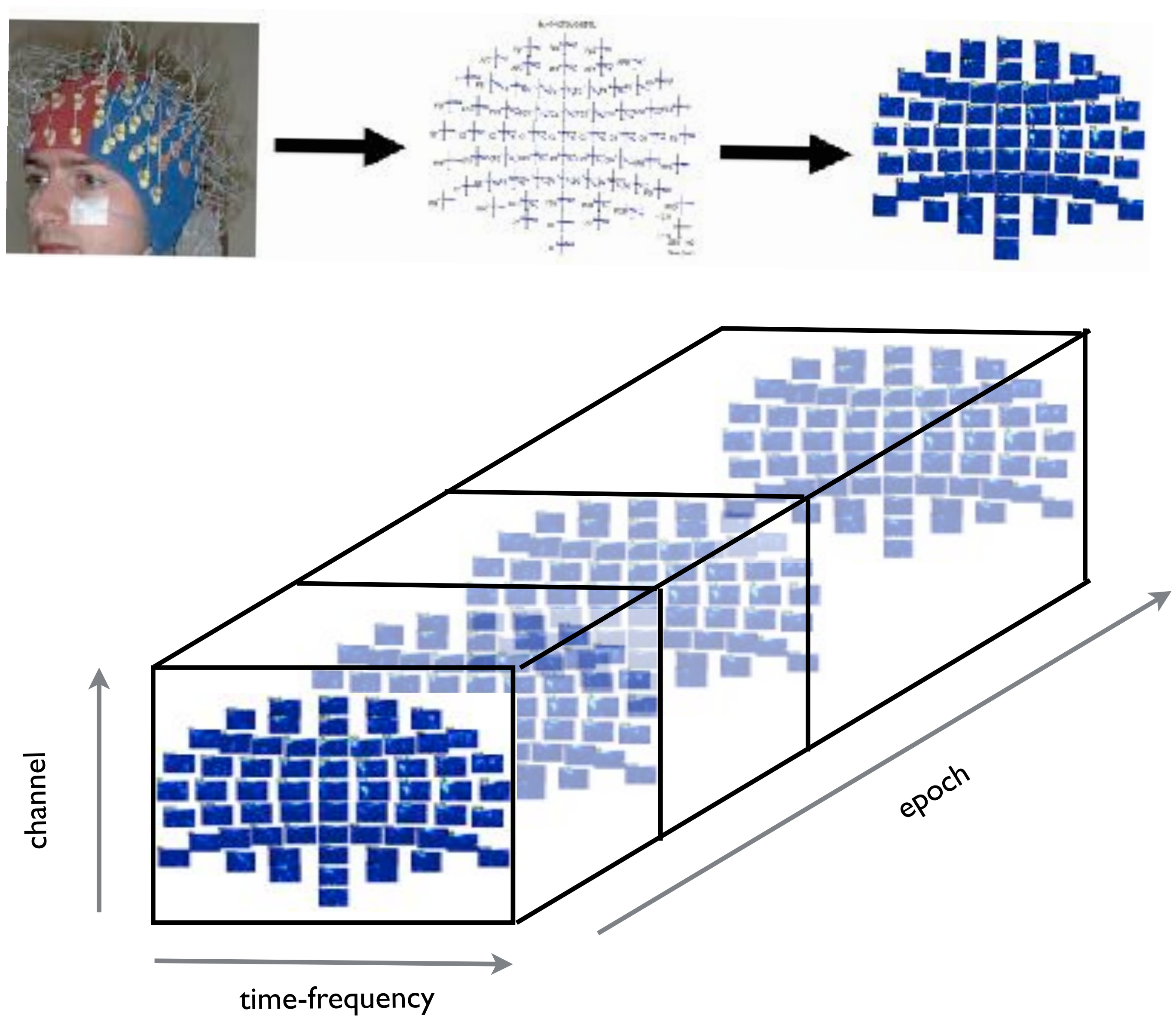
- Data ensemble affected by multiple factors
  - Facial images (expression x people x illumination x views)
  - Collaborative filtering (user x item x time)
- Multidimensional structured data, e.g.,
  - EEG, ECoG (channel x time x frequency)
  - fMRI (3D volume indexed by cartesian coordinate)
  - Video sequences (width x height x frame)

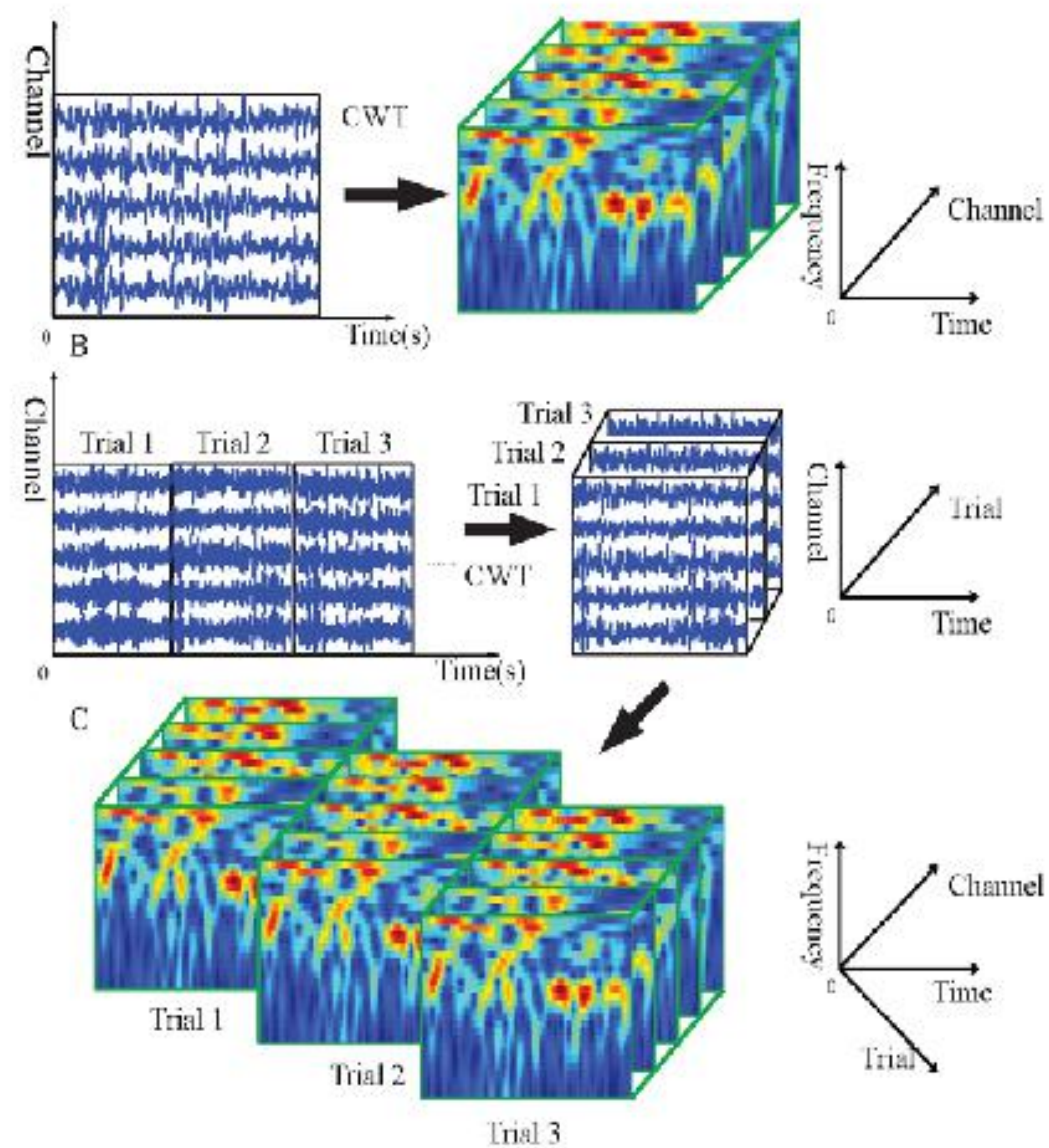


(c)



Tensor Representation of EEG Signals





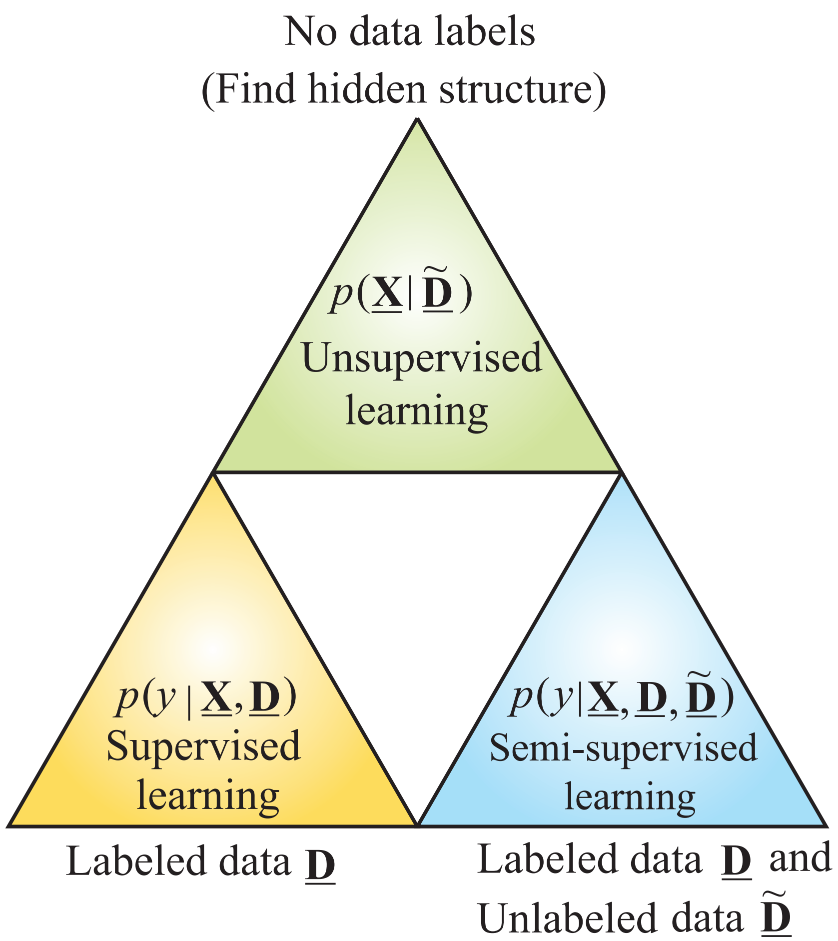
- Matricization causes loss of useful multiway information.
- It is favorable to analyze multi-dimensional data in their own domain.

Outline

- Tensor Regression and Classification
- TensorNets for Deep Neural Networks Compression
- (Multi-)Tensor Completion
- Tensor Denoising

Machine Learning Tasks

- Supervised (and semi-supervised) learning predict a target y from an input x
  - ✓ classification target y represents a category or class
  - ✓ regression target y is real-value number
- Unsupervised learning no explicit prediction target y
  - ✓ density estimation model the probability distribution of input x
  - ✓ clustering, dimensionality reduction discover underlying structure in input x



Classical Regression Models

- Regression models
  - ✓ predict one or more responses (dependent variables, outputs) from a set of predictors (independent variables, inputs)
  - ✓ identify the key predictors (independent variables, inputs)
- Linear and nonlinear regression models
  - ✓ linear model: simple regression, multiple regression, multivariate regression, generalized linear model, partial least squares (PLS)
  - ✓ nonlinear model: Gaussian process (GP), artificial neural networks (ANN), support vector regression (SVR)

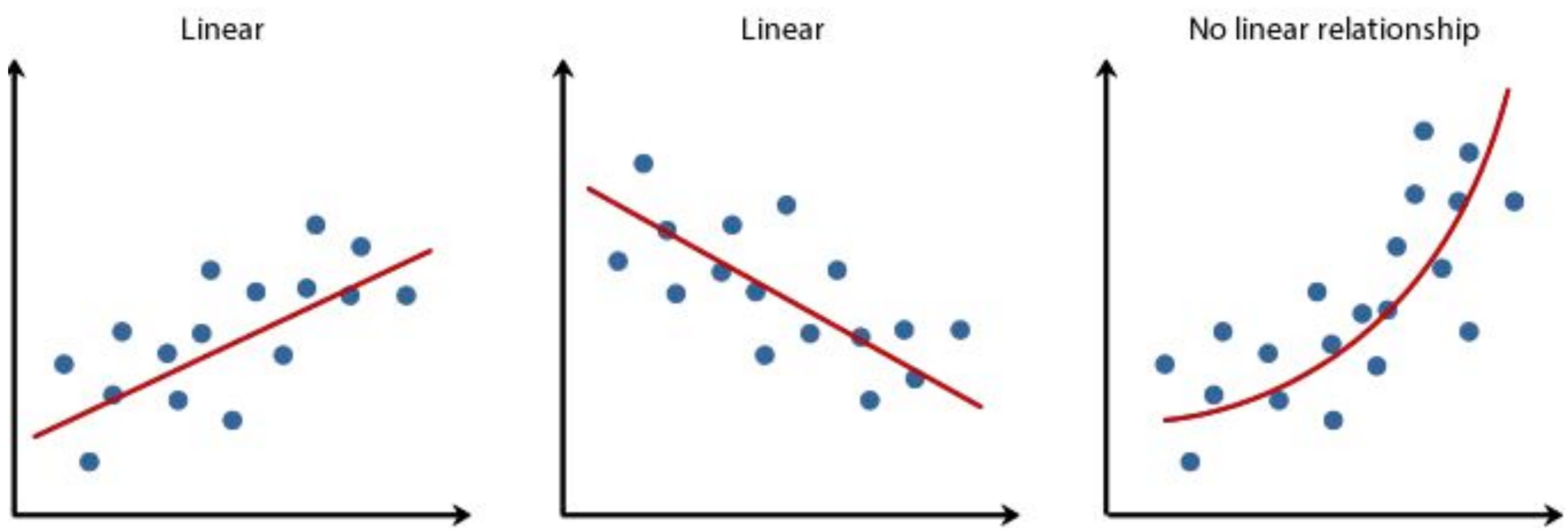


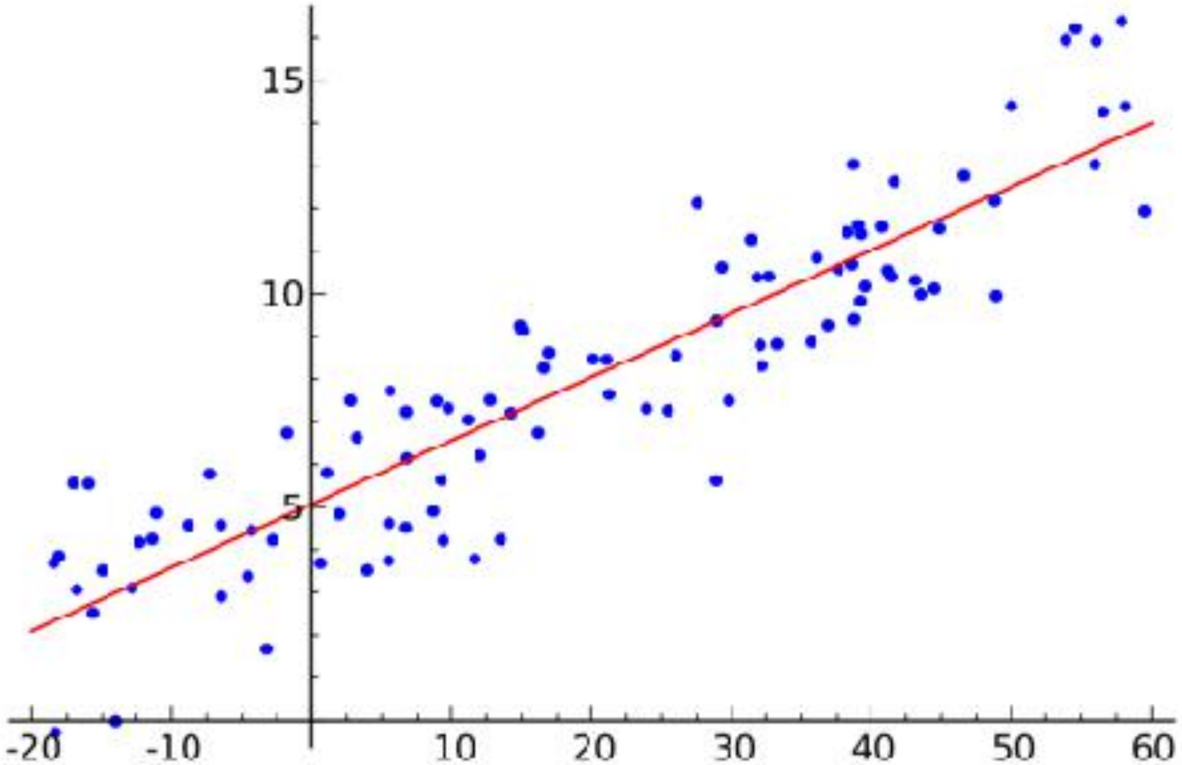
image credit Leard statistics

Basic Linear Regression Model

- A basic linear regression model in vector form is defined as

$$y = f(\mathbf{x}; \mathbf{w}, b) = \langle \mathbf{x}, \mathbf{w} \rangle + b = \mathbf{w}^{\mathrm{T}}\mathbf{x} + b$$

  - ✓ $\mathbf{x} \in \mathbb{R}^I$ is the input vector of independent variables
  - ✓ $\mathbf{w} \in \mathbb{R}^I$ is the vector of regression coefficients
  - ✓ b is the bias
  - ✓ y is the regression output or dependent/target variable

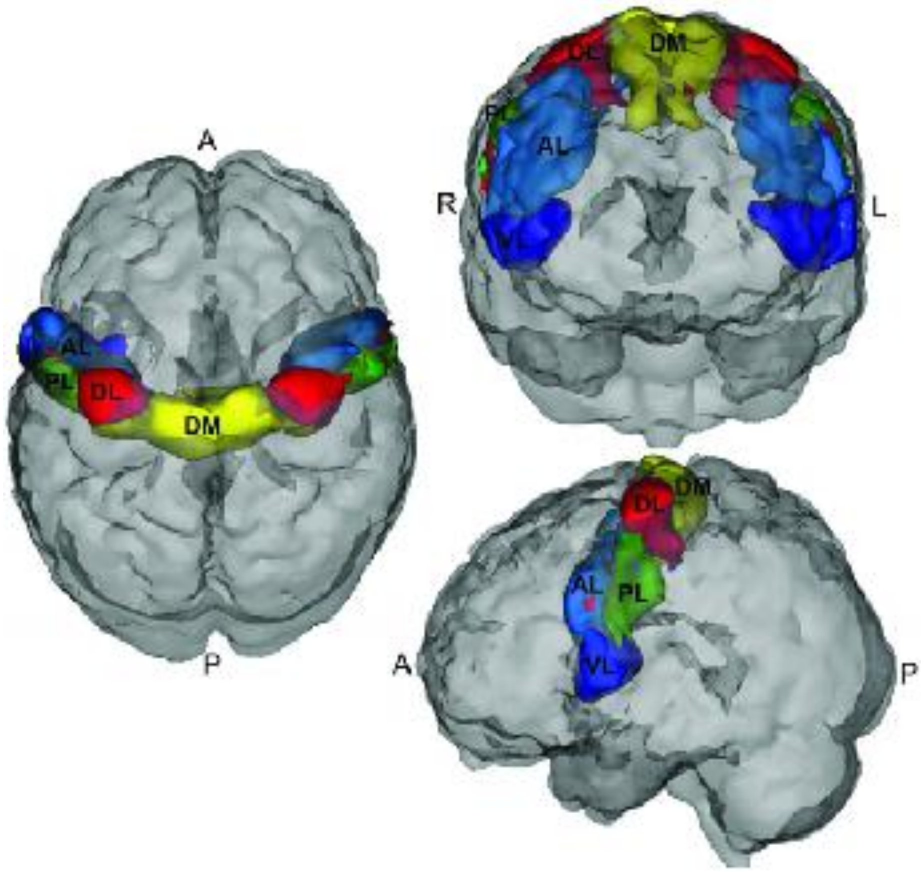


Tensor Data in Real-world Applications

- Medical imaging data analysis
  - ✓ MRI data x-coordinate × y-coordinate × z-coordinate
  - ✓ fMRI data time × x-coordinate × y-coordinate × z-coordinate
- Neural signal processing
  - ✓ EEG data time × frequency × channel
- Computer vision
  - ✓ video data frame × x-coordinate × y-coordinate
  - ✓ face image data pixel × illumination × expression × viewpoint × identity
- Climate data analysis
  - ✓ climate forecast data month × location × variable
- Chemistry
  - ✓ fluorescence excitation-emission data sample × excitation × emission

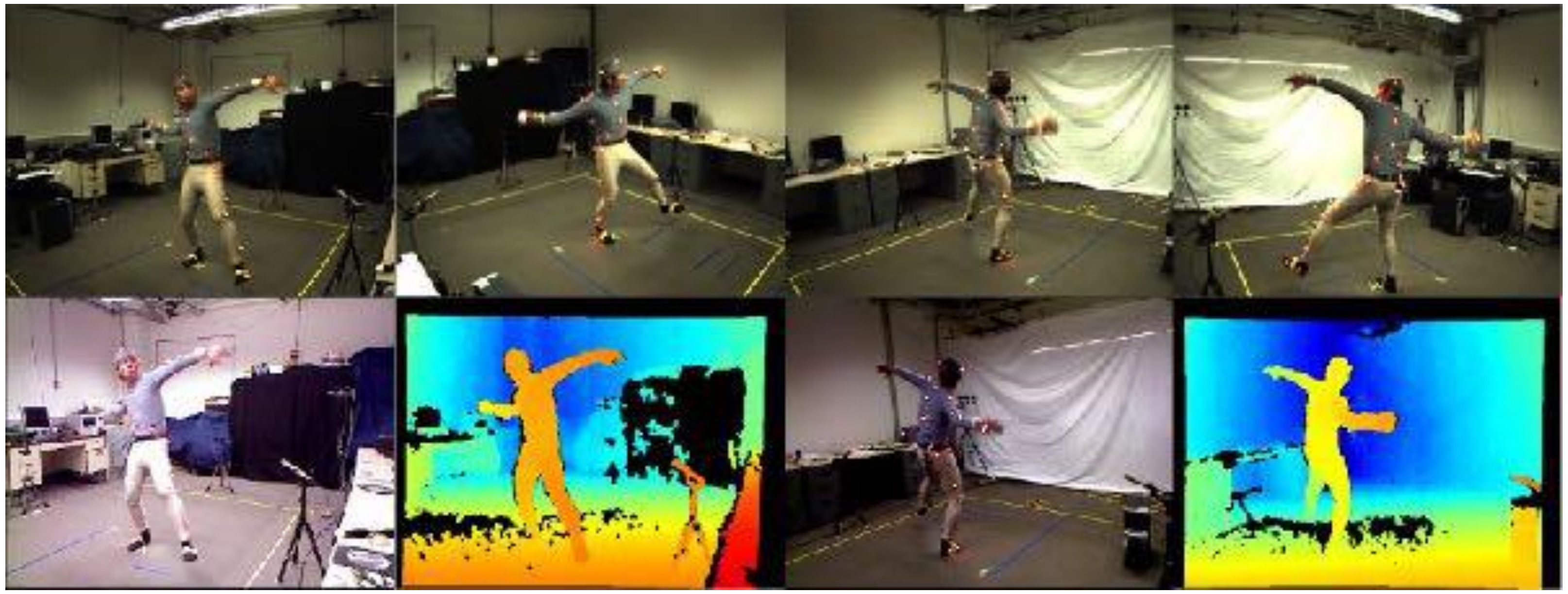
Real-world Regression Tasks with Tensors

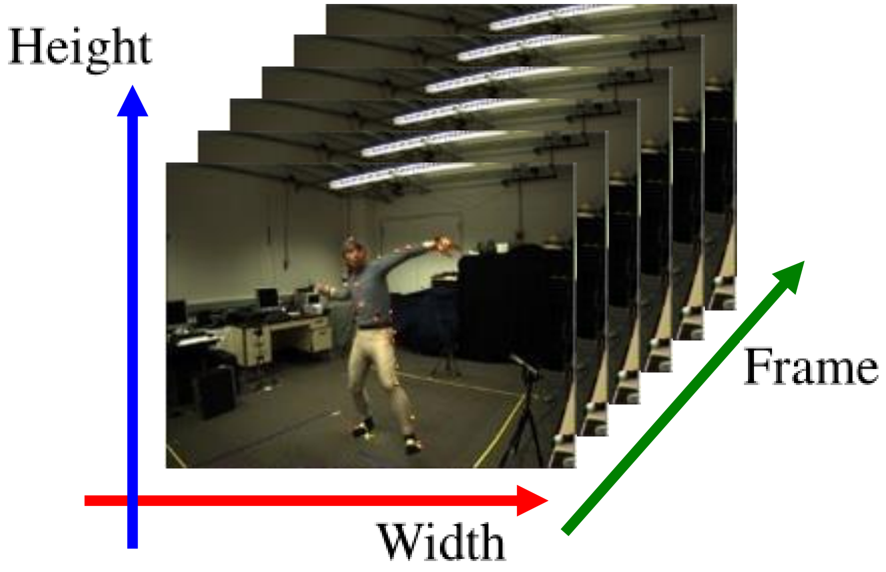
- Goal is to find association between brain images and clinical outcomes
  - ✓ predictor 3rd-order tensor MRI images
  - ✓ response scaler clinical diagnosis indicating one has some disease or not



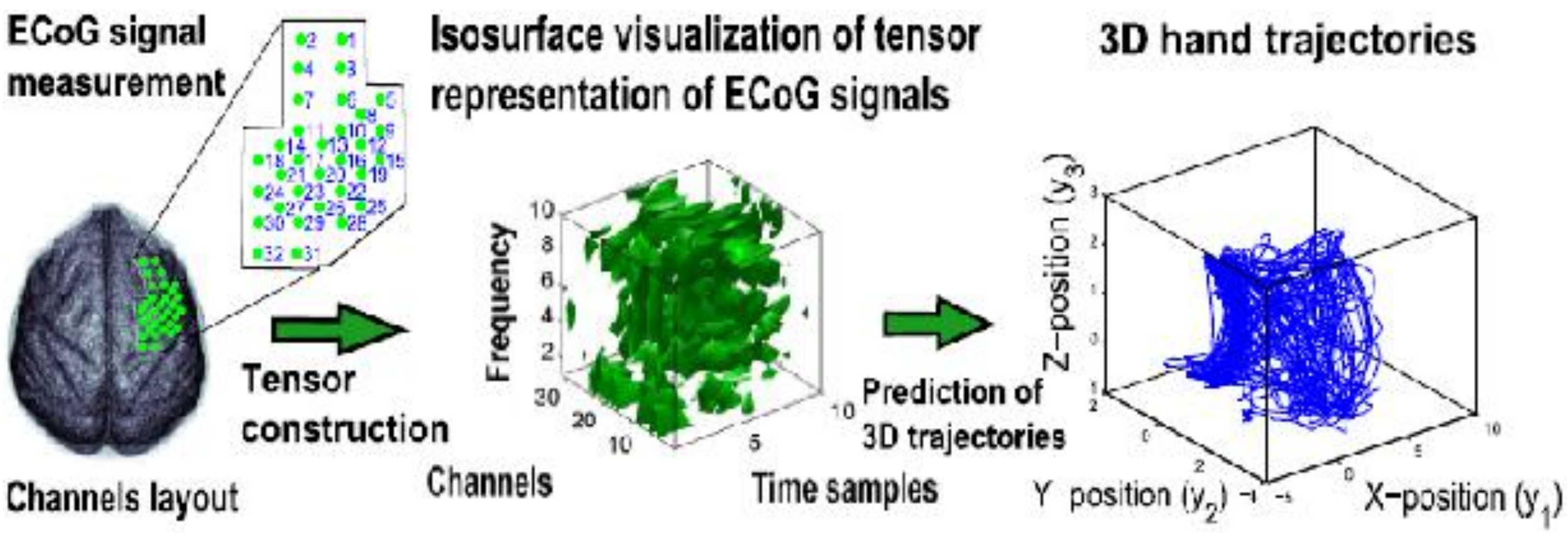
Real-world Regression Tasks with Tensors Cont

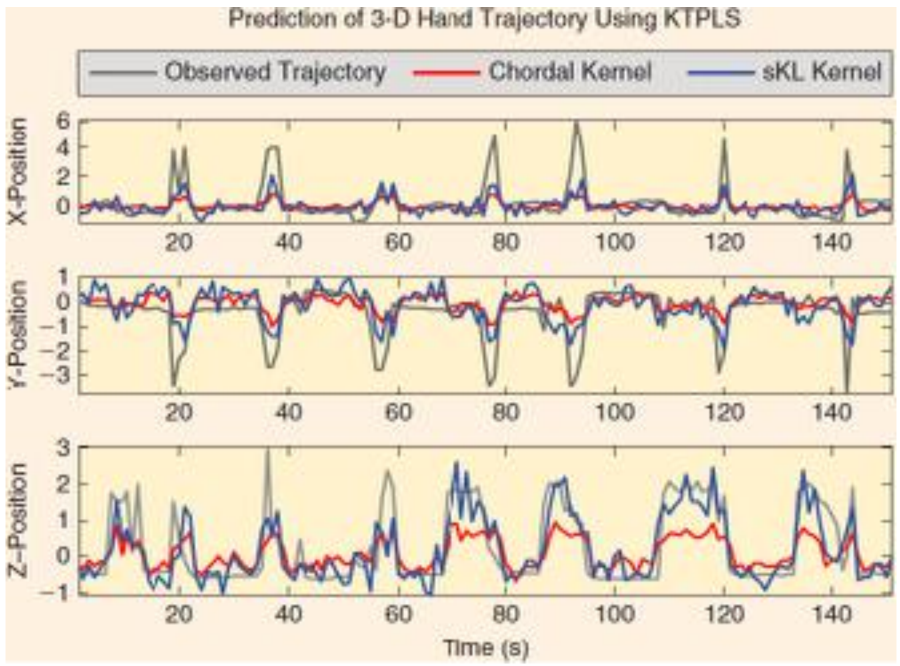
- Goal is to estimate 3D human pose positions from video sequences
  - ✓ predictor 4th-order tensor RGB video (or depth video)
  - ✓ response 3rd-order tensor human motion capture data





- Goal is to reconstruct motion trajectories from brain signals
  - ✓ predictor 4th-order tensor ECoG signals of monkey
  - ✓ response 3rd-order tensor limb movement trajectories





Motivations from New Regression Challenges

- Classical regression models transform tensors into vectors via vectorization operations, then feed them to two-way data analysis techniques for solutions
  - ✓ vectorizing operations destroy underlying multiway structures

    i.e. spatial and temporal correlations are ignored among pixels in a fMRI
  - ✓ ultrahigh tensor dimensionality produces huge parameters

    i.e. a fMRI of size $100 \times 256 \times 256 \times 256$ yields 167 millions!
  - ✓ difficulty of interpretation, sensitivity to noise, absence of uniqueness

- Tensor-based regression models directly model tensors using multiway factor models and multiway analysis techniques
  - ✓ naturally preserve multiway structural knowledge which is useful in mitigating small sample size problem
  - ✓ compactly represent regression coefficients using only a few parameters
  - ✓ ease of interpretation, robust to noise, uniqueness property

Basic Tensor Regression Model

- A basic linear tensor regression model can be formulated as

$$y = f(\underline{\mathbf{X}}; \underline{\mathbf{W}}, b) = \langle \underline{\mathbf{X}}, \underline{\mathbf{W}} \rangle + b$$

  - ✓ $\underline{\mathbf{X}} \in \mathbb{R}^{I_1 \times \cdots \times I_N}$ is the input tensor predictor or tensor regressor
  - ✓ $\underline{\mathbf{W}} \in \mathbb{R}^{I_1 \times \cdots \times I_N}$ is the regression coefficients tensor
  - ✓ b is the bias
  - ✓ y is the regression output or dependent/target variable
  - ✓ $\langle \underline{\mathbf{X}}, \underline{\mathbf{W}} \rangle = \mathrm{vec}(\underline{\mathbf{X}})^{\mathrm{T}} \mathrm{vec}(\underline{\mathbf{W}})$ is the inner product of two tensors
  - ✓ sparse regularization like lasso penalty on $\underline{\mathbf{W}}$ further improves the performance

- The learning of the tensor regression model is typically formulated as the minimization of following squared cost function

$$J(\underline{\mathbf{X}}, y \mid \underline{\mathbf{W}}, b) = \sum_{m=1}^{M} \left(y_m - (\langle \underline{\mathbf{W}}, \underline{\mathbf{X}}_m \rangle + b) \right)^2$$

  - ✓ $\{\underline{\mathbf{X}}_m, y_m\}$ $m = 1, \ldots, M$ are the M pairs of training samples

CP Regression Model

- The linear CP tensor regression [Zhou et. al 2013] model given by

$$y = f(\underline{\mathbf{X}}; \underline{\mathbf{W}}, b) = \langle \underline{\mathbf{X}}, \underline{\mathbf{W}} \rangle + b$$

where the coefficient tensor $\underline{\mathbf{W}}$ is assumed to follow a CP decomposition

$$\begin{aligned} \underline{\mathbf{W}} &= \sum_{r=1}^{R} \mathbf{u}_r^{(1)} \circ \mathbf{u}_r^{(2)} \circ \cdots \circ \mathbf{u}_r^{(N)} \\ &= \underline{\mathbf{I}} \times_1 \mathbf{U}^{(1)} \times_2 \mathbf{U}^{(2)} \cdots \times_N \mathbf{U}^{(N)} \end{aligned}$$

- The advantages of CP regression
  - ✓ substantial reduction in dimensionality

    i.e. a 128×128×128 MRI image, the parameters reduce from 2,097,157 to 1157 via rank-3 decomposition
  - ✓ low rank CP model could provide a sound recovery of many low rank signals

Tucker Regression Model

- The linear Tucker tensor regression [Li et. al 2013] model given by

$$y = f(\underline{\mathbf{X}}; \underline{\mathbf{W}}, b) = \langle \underline{\mathbf{X}}, \underline{\mathbf{W}} \rangle + b$$

where the coefficient tensor $\underline{\mathbf{W}}$ is assumed to follow a Tucker decomposition

$$\underline{\mathbf{W}} = \underline{\mathbf{G}} \times_1 \mathbf{U}^{(1)} \times_2 \mathbf{U}^{(2)} \cdots \times_N \mathbf{U}^{(N)}$$

- The shared advantages of Tucker regression with CP regression
  - ✓ substantially reduce the dimensionality
  - ✓ provide a sound low rank approximation to potentially high rank signal

- The advantages of Tucker regression over CP regression
  - ✓ offer freedom in choice of different ranks when tensor data is skewed in dimensions
  - ✓ explicitly model the interactions between factor matrices

General Linear Tensor Regression Model

- A general tensor regression model can be obtained when regression coefficient tensor $\underline{\mathbf{W}}$ is high-order than the input tensors $\underline{\mathbf{X}}_m$, leading to

$$\underline{\mathbf{Y}}_m = \langle \underline{\mathbf{X}}_m | \underline{\mathbf{W}} \rangle + \underline{\mathbf{E}}_m, \qquad m = 1, \ldots, M$$

  - ✓ $\underline{\mathbf{X}}_m \in \mathbb{R}^{I_1 \times \cdots \times I_N}$ is the Nth-order predictor tensor
  - ✓ $\underline{\mathbf{W}} \in \mathbb{R}^{I_1 \times \cdots \times I_P}$ is the Pth-order regression coefficient tensor with $P > N$
  - ✓ $\underline{\mathbf{Y}} \in \mathbb{R}^{I_{P+1} \times \cdots \times I_P}$ is the (P-N)th-order response tensor
  - ✓ $\langle \underline{\mathbf{X}}_m | \underline{\mathbf{W}} \rangle$ denotes a tensor contraction along the first N modes

- This model allows response to be a high-order tensor
- This model includes many linear tensor regression models as special cases i.e., CP regression, Tucker regression, etc

PLS for Matrix Regression

- Goal of partial least squares (PLS) regression is to predict the response matrix Y from the predictor matrix X, and describe their common latent structure

- The PLS regression consists of two steps

  i) extract a set of latent variables of X and Y by performing a simultaneous decomposition of X and Y, such that maximum pairwise covariance is between the latent variables of X and the latent variables of Y

  ii) use the extracted latent variables to predict Y

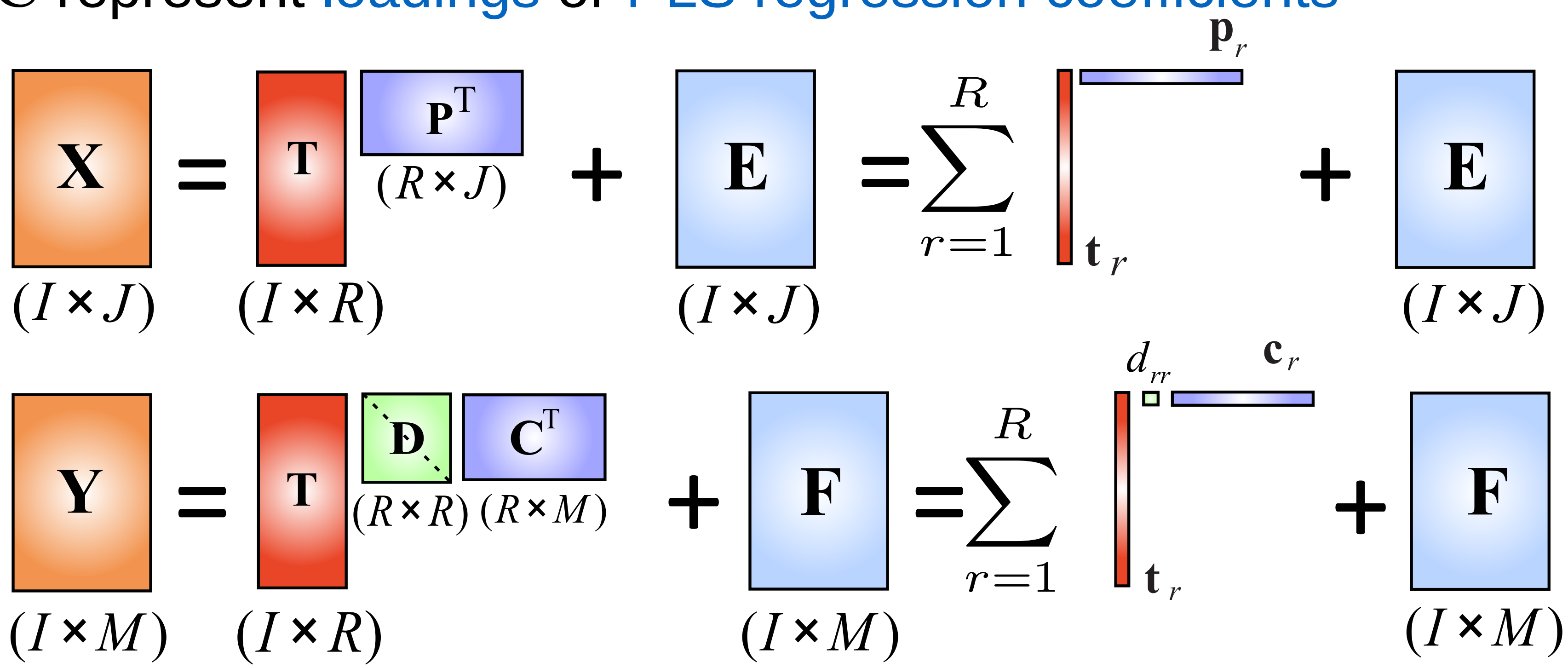
PLS for Matrix Regression Cont

- The standard PLS regression takes the form of

$$\mathbf{X} = \mathbf{T}\mathbf{P}^{\mathrm{T}} + \mathbf{E} = \sum_{r=1}^{R} \mathbf{t}_r \mathbf{p}_r^{\mathrm{T}} + \mathbf{E},$$

$$\mathbf{Y} = \mathbf{T}\mathbf{D}\mathbf{C}^{\mathrm{T}} + \mathbf{F} = \sum_{r=1}^{R} d_{rr} \mathbf{t}_r \mathbf{c}_r^{\mathrm{T}} + \mathbf{F}$$

- ✓ $\mathbf{X} \in \mathbb{R}^{I \times J}$ is the matrix predictor and $\mathbf{Y} \in \mathbb{R}^{I \times M}$ is the matrix response
- ✓ $\mathbf{T} = [\mathbf{t}_1, \mathbf{t}_2, \ldots, \mathbf{t}_R] \in \mathbb{R}^{I \times R}$ contains R latent variables from $\mathbf{X}$
- ✓ $\mathbf{U} = \mathbf{T}\mathbf{D} = [\mathbf{u}_1, \mathbf{u}_2, \ldots, \mathbf{u}_R] \in \mathbb{R}^{I \times R}$ represents R latent variables from $\mathbf{Y}$
- ✓ $\mathbf{P}$ and $\mathbf{C}$ represent loadings or PLS regression coefficients



PLS for Matrix Regression Cont

- The PLS typically applies a deflation strategy to extract the latent variables $\mathbf{T} = [\mathbf{t}_1, \mathbf{t}_2, \ldots, \mathbf{t}_R] \in \mathbb{R}^{I \times R}$ and $\mathbf{U} = \mathbf{TD} = [\mathbf{u}_1, \mathbf{u}_2, \ldots, \mathbf{u}_R] \in \mathbb{R}^{I \times R}$ as well as all the loadings
- A classical algorithm for the extraction process is called nonlinear iterative partial least squares PLS regression algorithm (NIPALS-PLS) [Wold, 1984]
- Having extracted all the factors, the prediction for the new test point $\mathbf{X}^*$ can be performed by

$$\mathbf{Y}^* \approx \mathbf{X}^* \mathbf{W} \mathbf{D} \mathbf{C}^{\mathrm{T}}$$

here $\mathbf{W}$ is some weight matrix obtained from NIPALS-PLS algorithm

High-order PLS for Tensor Regression

- Goal of high-order partial least squares (HOPLS) [Zhao et. al 2011] regression allows to predict the response tensor Y from the predictor tensor X and describe their common latent structure
- HOPLS extends PLS by projecting tensorial data onto a common latent subspace but using block Tucker decomposition [De Lathauwer, 2008]

- Similarly, HOPLS regression consists of two steps

  i) extract a set of latent variables of tensor X and tensor Y by performing a simultaneous block Tucker decomposition of both tensor X and tensor Y, such that maximum pairwise covariance is between the latent variables of X and the latent variables of Y

  ii) use the extracted latent variables to predict tensor Y

HOPLS Framework

- The standard HOPLS performs joint block Tucker decomposition of both predictor tensor and response tensor by

$$\underline{\mathbf{X}} = \sum_{r=1}^{R} \underline{\mathbf{G}}_{xr} \times_1 \mathbf{t}_r \times_2 \mathbf{P}_r^{(1)} \cdots \times_{N+1} \mathbf{P}_r^{(N)} + \underline{\mathbf{E}}_R$$
$$\underline{\mathbf{Y}} = \sum_{r=1}^{R} \underline{\mathbf{G}}_{yr} \times_1 \mathbf{t}_r \times_2 \mathbf{Q}_r^{(1)} \cdots \times_{N+1} \mathbf{Q}_r^{(N)} + \underline{\mathbf{F}}_R$$

✓ $\underline{\mathbf{X}} \in \mathbb{R}^{M \times I_1 \times \cdots \times I_N}$ is the (N+1)th-order predictor tensor by concatenating M samples

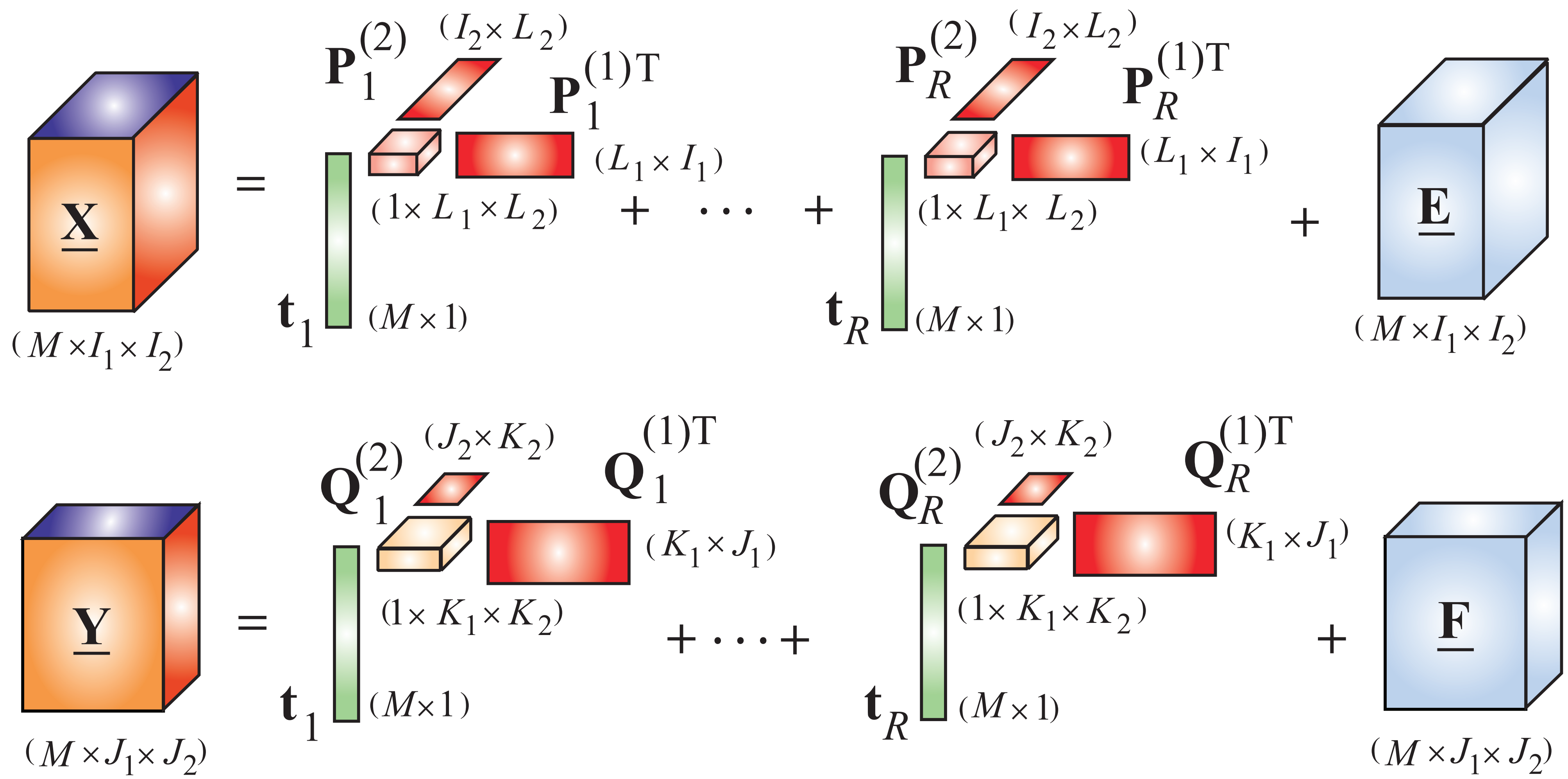
✓ $\underline{\mathbf{Y}} \in \mathbb{R}^{M \times J_1 \times \cdots \times J_N}$ is the (N+1)th-order response tensor having the same size M

✓ $\mathbf{t}_r \in \mathbb{R}^M$ is the latent variable for the r-th component

✓ $\left\{\mathbf{P}_r^{(n)}\right\}_{n=1}^{N} \in \mathbb{R}^{I_n \times L_n}$ and $\left\{\mathbf{Q}_r^{(n)}\right\}_{n=1}^{N} \in \mathbb{R}^{J_n \times K_n}$ are the loadings for r-th component

✓ $\underline{\mathbf{G}}_{xr} \in \mathbb{R}^{1 \times L_1 \times \cdots \times L_N}$ and $\underline{\mathbf{G}}_{yr} \in \mathbb{R}^{1 \times K_1 \times \cdots \times K_N}$ are the core tensors for r-th component

HOPLS Framework Cont



$$\underline{\mathbf{X}} = \sum_{r=1}^{R} \underline{\mathbf{G}}_{xr} \times_1 \mathbf{t}_r \times_2 \mathbf{P}_r^{(1)} \cdots \times_{N+1} \mathbf{P}_r^{(N)} + \underline{\mathbf{E}}_R$$

$$\underline{\mathbf{Y}} = \sum_{r=1}^{R} \underline{\mathbf{G}}_{yr} \times_1 \mathbf{t}_r \times_2 \mathbf{Q}_r^{(1)} \cdots \times_{N+1} \mathbf{Q}_r^{(N)} + \underline{\mathbf{F}}_R$$

HOPLS Framework A Compact Formulation

- The standard HOPLS can be rewritten in a more compact form

$$\underline{\mathbf{X}} = \underline{\mathbf{G}}_x \times_1 \mathbf{T} \times_2 \overline{\mathbf{P}}^{(1)} \cdots \times_{N+1} \overline{\mathbf{P}}^{(N)} + \underline{\mathbf{E}}_R$$

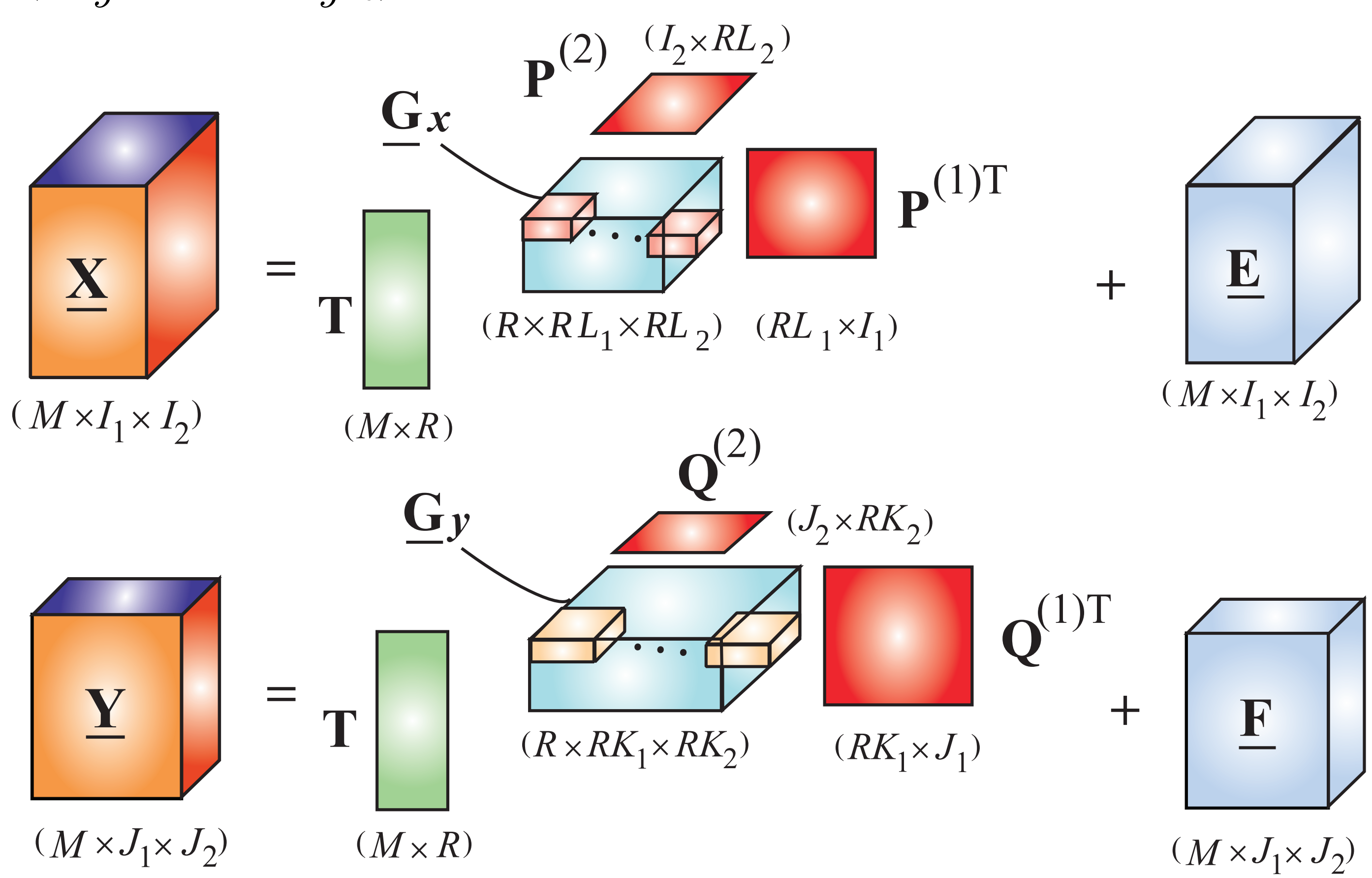
$$\underline{\mathbf{Y}} = \underline{\mathbf{G}}_y \times_1 \mathbf{T} \times_2 \overline{\mathbf{Q}}^{(1)} \cdots \times_{N+1} \overline{\mathbf{Q}}^{(N)} + \underline{\mathbf{F}}_R$$

✓ $\mathbf{T} = [\mathbf{t}_1, \ldots, \mathbf{t}_R]$ is the latent matrix

✓ $\overline{\mathbf{P}}^{(n)} = [\mathbf{P}_1^{(n)}, \ldots, \mathbf{P}_R^{(n)}]$ and $\overline{\mathbf{Q}}^{(n)} = [\mathbf{Q}_1^{(n)}, \ldots, \mathbf{Q}_R^{(n)}]$ are the loading matrix

✓ $\underline{\mathbf{G}}_x = \mathrm{blockdiag}(\underline{\mathbf{G}}_{x1}, \ldots, \underline{\mathbf{G}}_{xR}) \in \mathbb{R}^{R \times RL_1 \times \cdots \times RL_N}$ is the core tensor for input

✓ $\underline{\mathbf{G}}_y = \mathrm{blockdiag}(\underline{\mathbf{G}}_{y1}, \ldots, \underline{\mathbf{G}}_{yR}) \in \mathbb{R}^{R \times RK_1 \times \cdots \times RK_N}$ is the core tensor for output



HOPLS Experimental Results

- Goal to decode limb movement trajectories based on ECoG signals of monkey
  - ✓ dataset ECoG food tracking data
  - ✓ predictor 4th-order tensor sample × time × frequency × channel
  - ✓ response 3rd-order tensor sample × time × 3D positions × marker

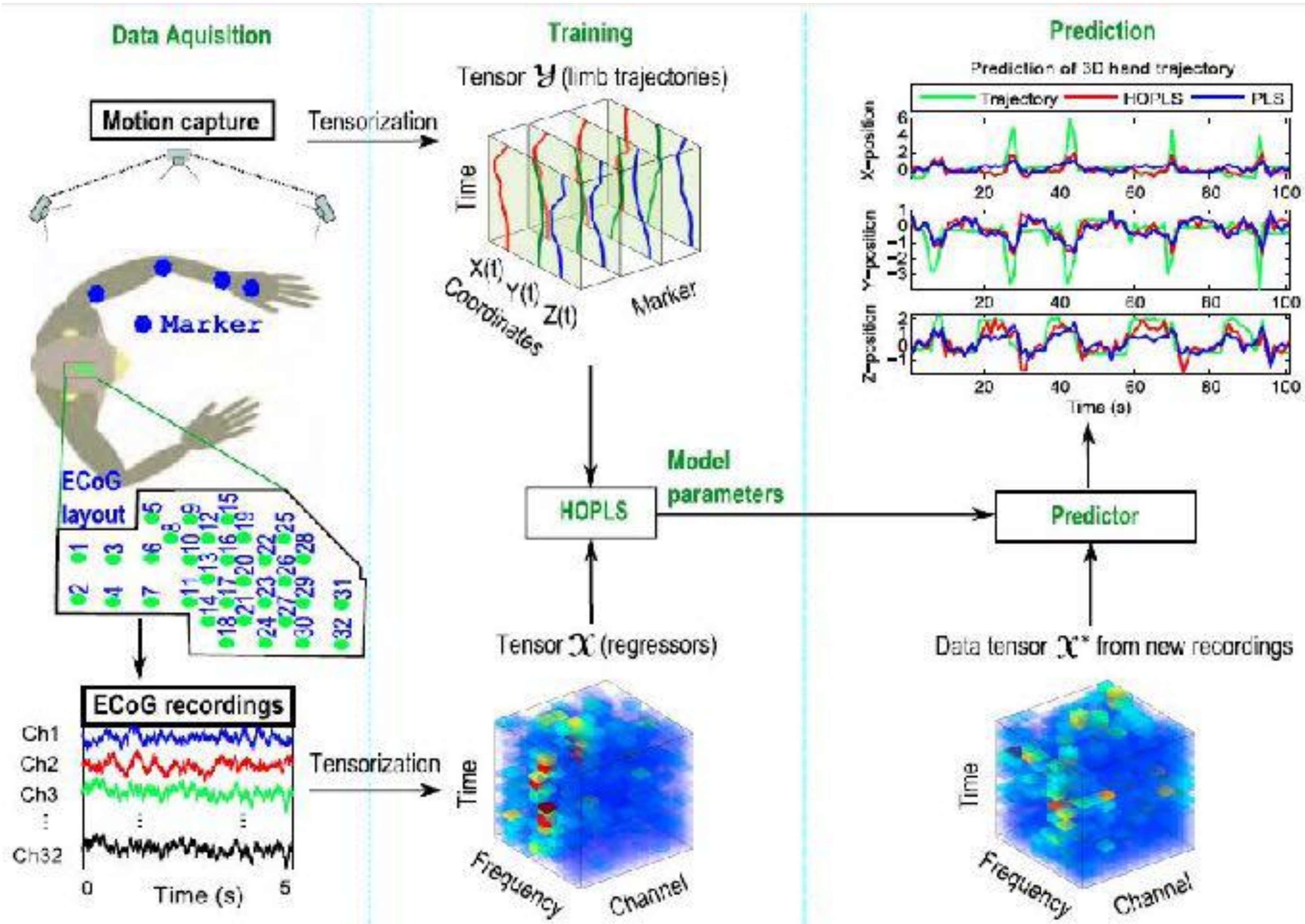


figure credit [Zhao et. al 2013]

Outline

- Tensor Regression
- TensorNets for Deep Neural Networks Compression
- (Multi-)Tensor Completion
- Tensor Denoising

Background

- Deep Neural Networks (DNNs) archives the state-of-art performance in many large-scale machine learning applications
  - ✓ i.e. computation vision, speech recognition and text processing etc

- DNNs have thousands of nodes and millions of learnable parameters and are trained using millions of images on GPUs
- DNNs reaches the hardware limits both in terms the computational power and the memory
- DNNs reaches the memory limit with 89% [Simonyan and Zisserman, 2015] or even 100% [Xue et al, 2013] memory occupied by the weight matrices of the fully-connected layers

VGGNet Example

- The huge number of parameters of FC layers is the bottleneck in a typical DNN like VGGNet [Simonyan and Zisserman, 2015]

(not counting biases)

```
INPUT: [224x224x3]        memory: 224*224*3=150K   params: 0
CONV3-64: [224x224x64]  memory: 224*224*64=3.2M   params: (3*3*3)*64 = 1,728
CONV3-64: [224x224x64]  memory: 224*224*64=3.2M   params: (3*3*64)*64 = 36,864
POOL2: [112x112x64]  memory: 112*112*64=800K   params: 0
CONV3-128: [112x112x128]  memory: 112*112*128=1.6M   params: (3*3*64)*128 = 73,728
CONV3-128: [112x112x128]  memory: 112*112*128=1.6M   params: (3*3*128)*128 = 147,456
POOL2: [56x56x128]  memory: 56*56*128=400K   params: 0
CONV3-256: [56x56x256]  memory: 56*56*256=800K   params: (3*3*128)*256 = 294,912
CONV3-256: [56x56x256]  memory: 56*56*256=800K   params: (3*3*256)*256 = 589,824
CONV3-256: [56x56x256]  memory: 56*56*256=800K   params: (3*3*256)*256 = 589,824
POOL2: [28x28x256]  memory: 28*28*256=200K   params: 0
CONV3-512: [28x28x512]  memory: 28*28*512=400K   params: (3*3*256)*512 = 1,179,648
CONV3-512: [28x28x512]  memory: 28*28*512=400K   params: (3*3*512)*512 = 2,359,296
CONV3-512: [28x28x512]  memory: 28*28*512=400K   params: (3*3*512)*512 = 2,359,296
POOL2: [14x14x512]  memory: 14*14*512=100K   params: 0
CONV3-512: [14x14x512]  memory: 14*14*512=100K   params: (3*3*512)*512 = 2,359,296
CONV3-512: [14x14x512]  memory: 14*14*512=100K   params: (3*3*512)*512 = 2,359,296
CONV3-512: [14x14x512]  memory: 14*14*512=100K   params: (3*3*512)*512 = 2,359,296
POOL2: [7x7x512]  memory: 7*7*512=25K  params: 0
FC: [1x1x4096]  memory: 4096  params: 7*7*512*4096 = 102,760,448
FC: [1x1x4096]  memory: 4096  params: 4096*4096 = 16,777,216
FC: [1x1x1000]  memory: 1000  params: 4096*1000 = 4,096,000
```

ConvNet Configuration			
B	C	D	
13 weight layers	16 weight layers	16 weight layers	19
put (224 × 224 RGB image			
conv3-64 **conv3-64**	conv3-64 conv3-64	conv3-64 conv3-64	cc cc
maxpool			
conv3-128 **conv3-128**	conv3-128 conv3-128	conv3-128 conv3-128	co co
maxpool			
conv3-256 conv3-256	conv3-256 conv3-256 **conv1-256**	conv3-256 conv3-256 **conv3-256**	co co co co
maxpool			
conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	co co co co
maxpool			
conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	co co co co
maxpool			
FC-4096			
FC-4096			
FC-1000			
soft-max			

TensorNet for DNN Compression

- TensorNet [Novikov et. al, 2015] applies tensor train (TT) [Oseledet, 2011] format to represent the dense weight matrix of the fully-connected layers using fewer parameters while keeping enough flexibility to perform signal transformations

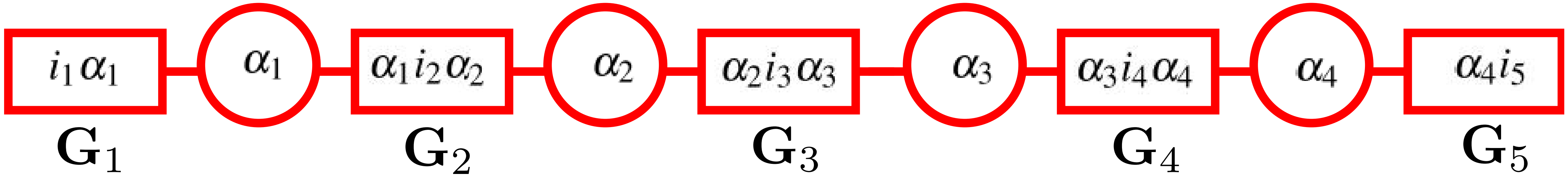
- The advantages of TensorNet
  - ✓ compatible with the existing training algorithms for neural networks
  - ✓ match the performance of the uncompressed counterparts with compression factor of the weights of FC layer up to 200, 000 times leading to the compression factor of the whole network up to 7 times
  - ✓ able to use more hidden units than was available before

Tensor Train Decomposition

- Recall that in index form, tensor train decomposition (TTD) can be represented by

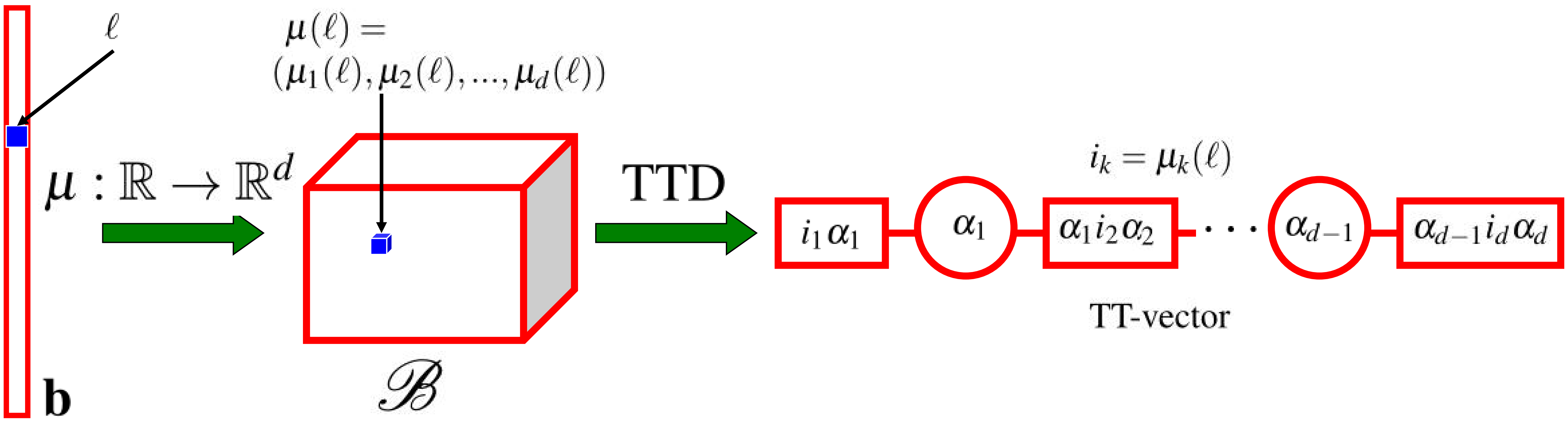
$$\mathcal{X}(i_1, i_2, ..., i_d) \approx \sum_{\alpha_0,\dots,\alpha_d} \mathbf{G}_1[i_1](\alpha_0, \alpha_1)\mathbf{G}_2[i_2](\alpha_1, \alpha_2)\cdots\mathbf{G}_d[i_d](\alpha_{d-1}, \alpha_d)$$

  - ✓ i.e. an illustration of TTD of 5th-order tensor



TT-vector

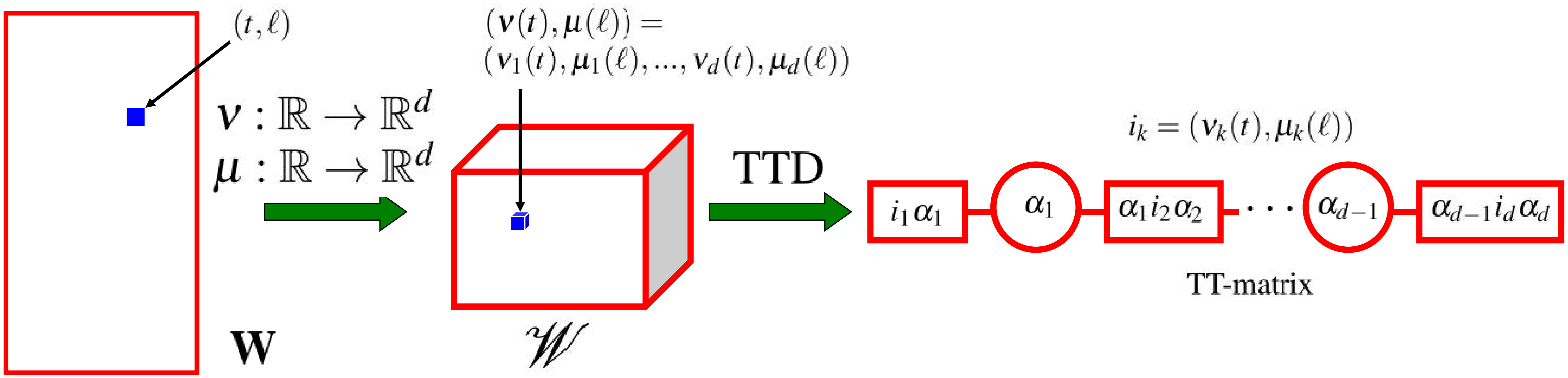
- TT-vector converts a long vector into a TT-format



- ✓ vector $\mathbf{b} \in \mathbb{R}^N$ where $N = \prod_{k=1}^{d} n_k$
- ✓ coordinate $\ell \in \{1, ..., N\}$ of vector $\mathbf{b} \in \mathbb{R}^N$
- ✓ d-dimensional vector-index $\mu(\ell) = (\mu_1(\ell), \mu_2(\ell), ..., \mu_d(\ell))$ of tensorized $\mathcal{B}$, where $\mu_k(\ell) \in \{1, ..., n_k\}$
- ✓ $\mathcal{B}(\mu(\ell)) = \mathbf{b}_\ell$ holds
- ✓ TT-format of $\mathcal{B}$ is called TT-vector

TT-matrix

- TT-matrix converts a big matrix into a TT-format



✓ matrix $\mathbf{W} \in \mathbb{R}^{M\times N}$ where $M = \prod_{k=1}^{d} m_k$ and $N = \prod_{k=1}^{d} n_k$

✓ row coordinate $t \in \{1,\ldots,M\}$ and column coordinate $\ell \in \{1,\ldots,N\}$ of

✓ d-dimensional vector-indices $(\nu(t),\mu(\ell)) = (\nu_1(t),\mu_1(\ell),\ldots,\nu_d(t),\mu_d(\ell))$ of $\mathbf{W}$

tensorized $\mathcal{W}$, where $\mu_k(t) \in \{1,\ldots,m_k\}$ and $\mu_k(\ell) \in \{1,\ldots,n_k\}$

✓ $\mathcal{W}(\nu(t),\mu(\ell)) = \mathbf{W}(t,\ell)$ holds

✓ TT-format of $\mathcal{W}$ is called TT-matrix

$$\mathcal{W}(\nu(t),\mu(\ell)) = \mathbf{G}_1[\nu_1(t),\mu_1(\ell)]\mathbf{G}_2[\nu_2(t),\mu_2(\ell)]\cdots\mathbf{G}_d[\nu_d(t),\mu_d(\ell)]$$

TT-layer

- Fully connected layers apply a linear transformation to N-dimensional input vector $\mathbf{x}$

$$\mathbf{y} = \mathbf{W}\mathbf{x} + \mathbf{b}$$

where the weight matrix $\mathbf{W} \in \mathbb{R}^{M \times N}$ and bias vector $\mathbf{b} \in \mathbb{R}^{M}$

- TT-layer transforms input $\mathbf{x}$ (in TT-vector) by the weight $\mathbf{W}$ (in TT-matrix), to the output

$$\mathcal{Y}(i_1, ..., i_d) = \sum_{j_1,...,j_d} \mathbf{G}_1[i_1, j_1] \cdots \mathbf{G}_d[i_d, j_d]\mathcal{X}(j_1, ..., j_d) + \mathcal{B}(i_1, ..., i_d)$$

- The application of TT-matrix-by-vector operation yields low computational complexity of forward pass $\mathcal{O}(dr^2 m \max(M, N))$

- The learning can be performed by applying back-propagation to FC layers to compute gradients w.r.t the tensor cores

TensorNet Experimental Results Cont

- Substitution of FC layers with the TT-layers in VGG-16 and VGG-19 networks
  - ✓ FC stands for a fully-connected layer
  - ✓ TT'\$' stands for a TT-layer with all the TT-ranks equal '\$'
  - ✓ MR'\$' stands for a fully-connected layer with the matrix ranks restricted to '\$'
  - ✓ The experiments report the compression factor of TT-layers; the resulting compression factor of the whole network; the top1 and top5 classification errors

Architecture	TT-layers compr.	vgg-16 compr.	vgg-19 compr.	vgg-16 top 1	vgg-16 top 5	vgg-19 top 1	vgg-19 top 5
FC FC FC	1	1	1	30.9	11.2	29.0	10.1
TT4 FC FC	50 972	3.9	3.5	31.2	11.2	29.8	10.4
TT2 FC FC	194 622	3.9	3.5	31.5	11.5	30.4	10.9
TT1 FC FC	713 614	3.9	3.5	33.3	12.8	31.9	11.8
TT4 TT4 FC	37 732	7.4	6	32.2	12.3	31.6	11.7
MR1 FC FC	3 521	3.9	3.5	99.5	97.6	99.8	99
MR5 FC FC	704	3.9	3.5	81.7	53.9	79.1	52.4
MR50 FC FC	70	3.7	3.4	36.7	14.9	34.5	15.8

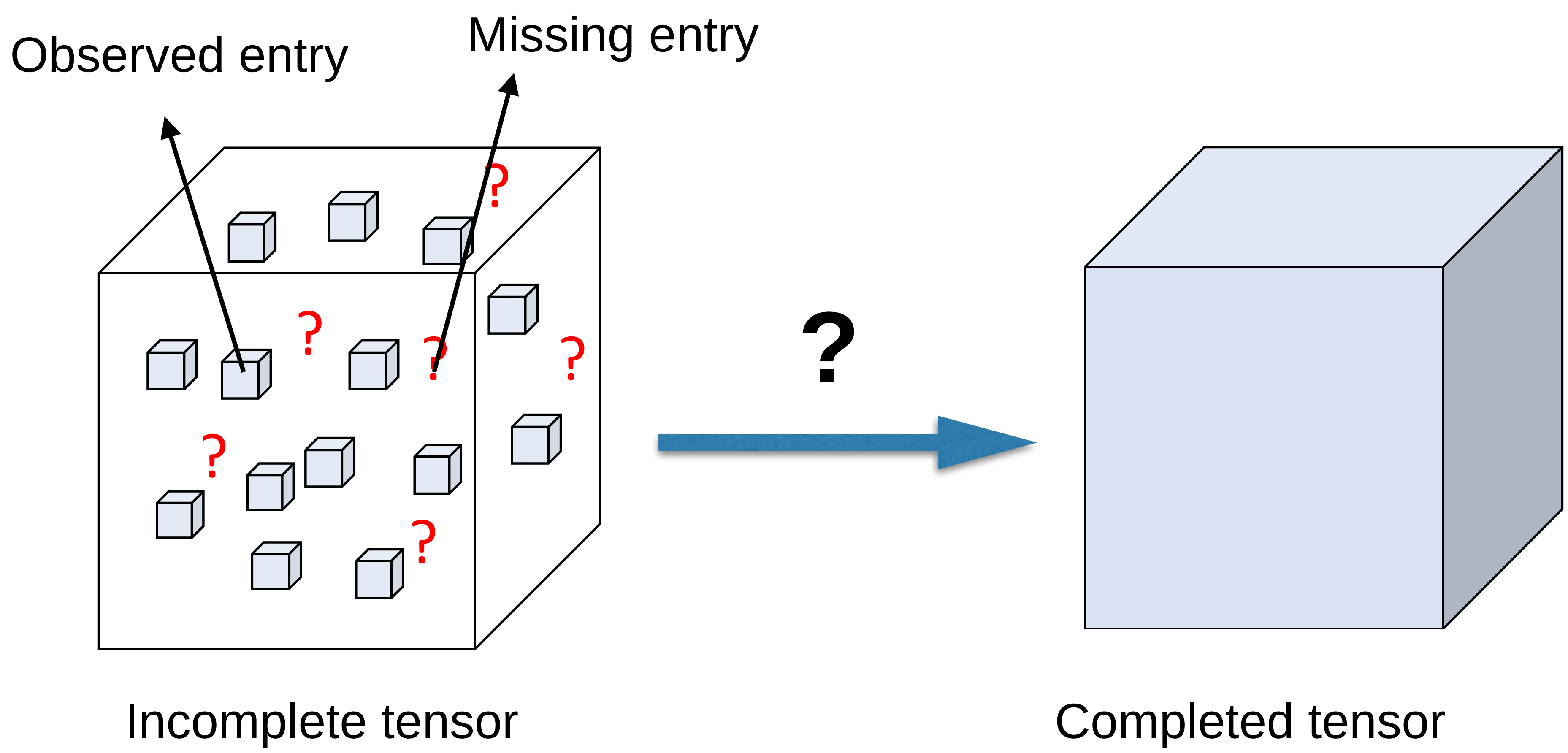
Outline

- Tensor Regression
- TensorNets for Deep Neural Networks Compression
- (Multi-)Tensor Completion
- Tensor Denoising

Tensor Completion

Tensor completion problem:

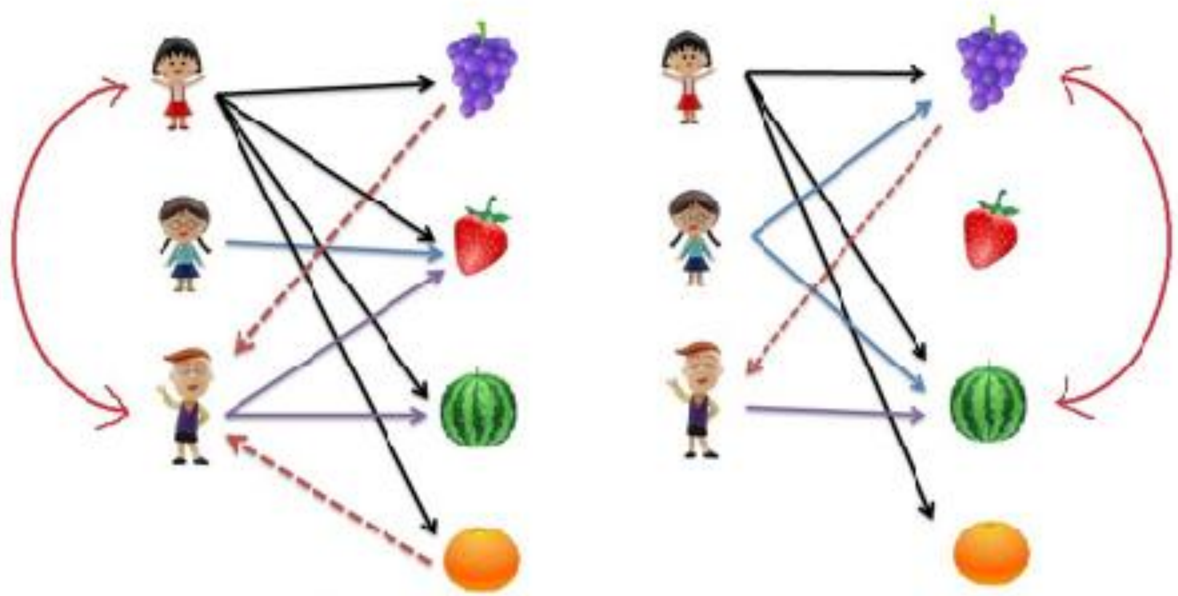
Tensor completion is to apply tensor method to infer a tensor with missing entries from partial observations.



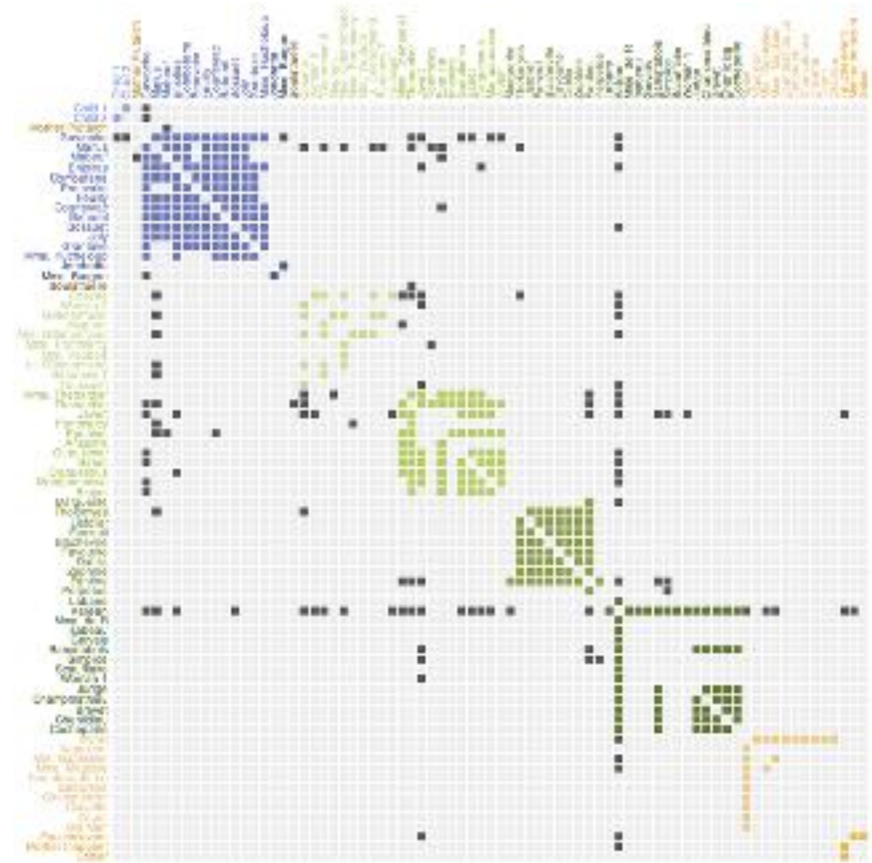
Motivation

Recommender system

Collaborative filtering



Social network analysis



	item1	item2	item3	item4	item5	item6
user1						
user2						
user3						
user4						

Movie ratings (Netflix)

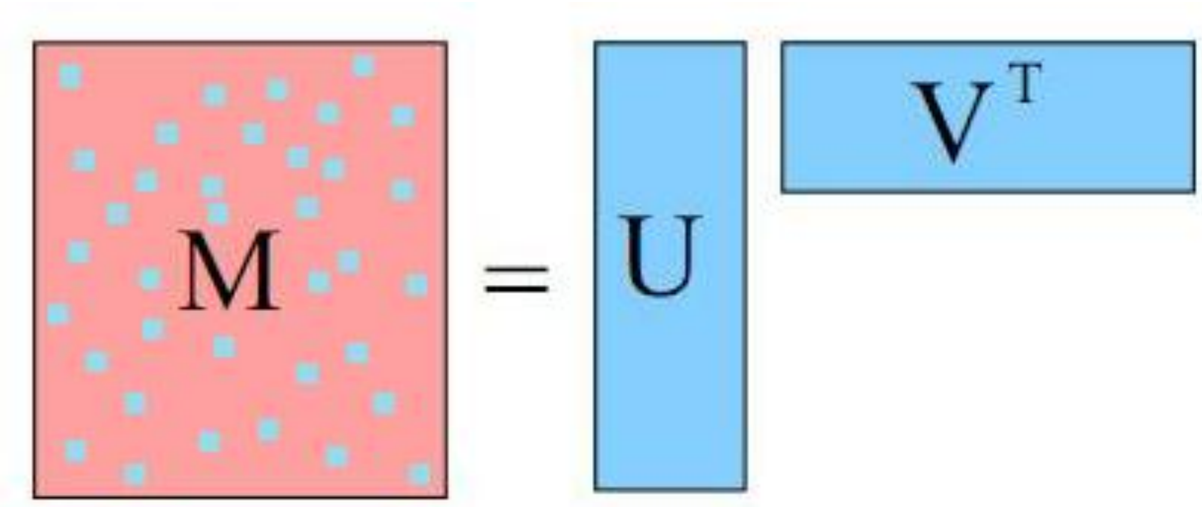
	i_1	i_2	i_3	i	i_5	...	[illegible]
Jack	5	1	3	4	3	...	1
u_2	0	4	0	0	0	...	5
u_3	0	2	0	0	0	...	4
Tom	4	1	4	5	4	...	2
...	...	...	...	...	...	...	...
Myself	5	1	4	?	3	...	1

similar users based on profile

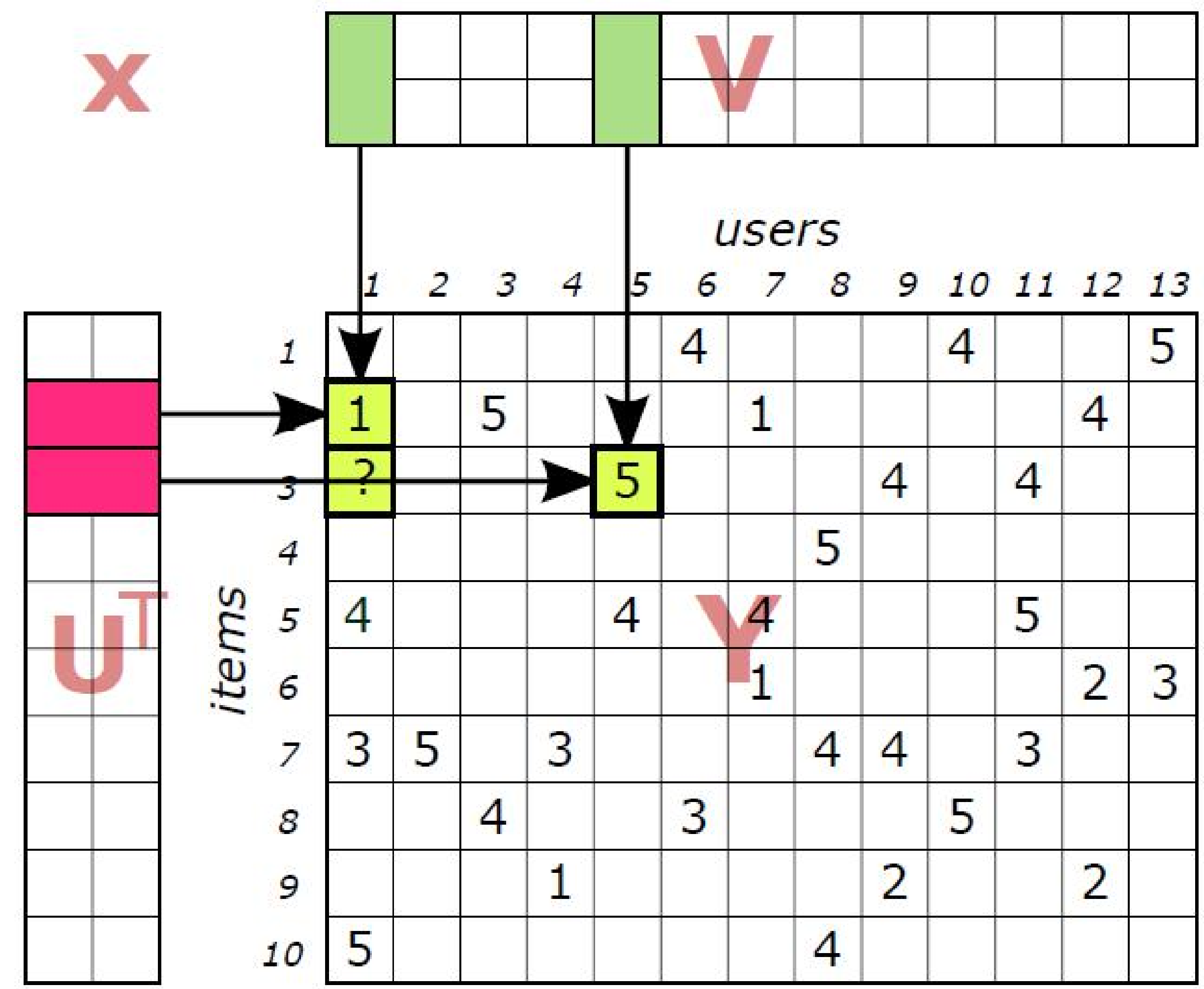
target user

predicted rating: 4.5

Matrix Factorization for Incomplete Data



$\mathbf{Y} = \mathbf{UV}$



Challenges:

- **ill-posed problem**
- **infinite solutions**

Regularizations:

- **Low-rank assumption**
- **Smoothness, non-negativity**

- Singular Value Decomposition (SVD)
- Non-negative Matrix Factorization (NMF)
- Probabilistic Matrix Factorization (PMF)
- Gaussian Process Latent Variable Models (GPLVM)

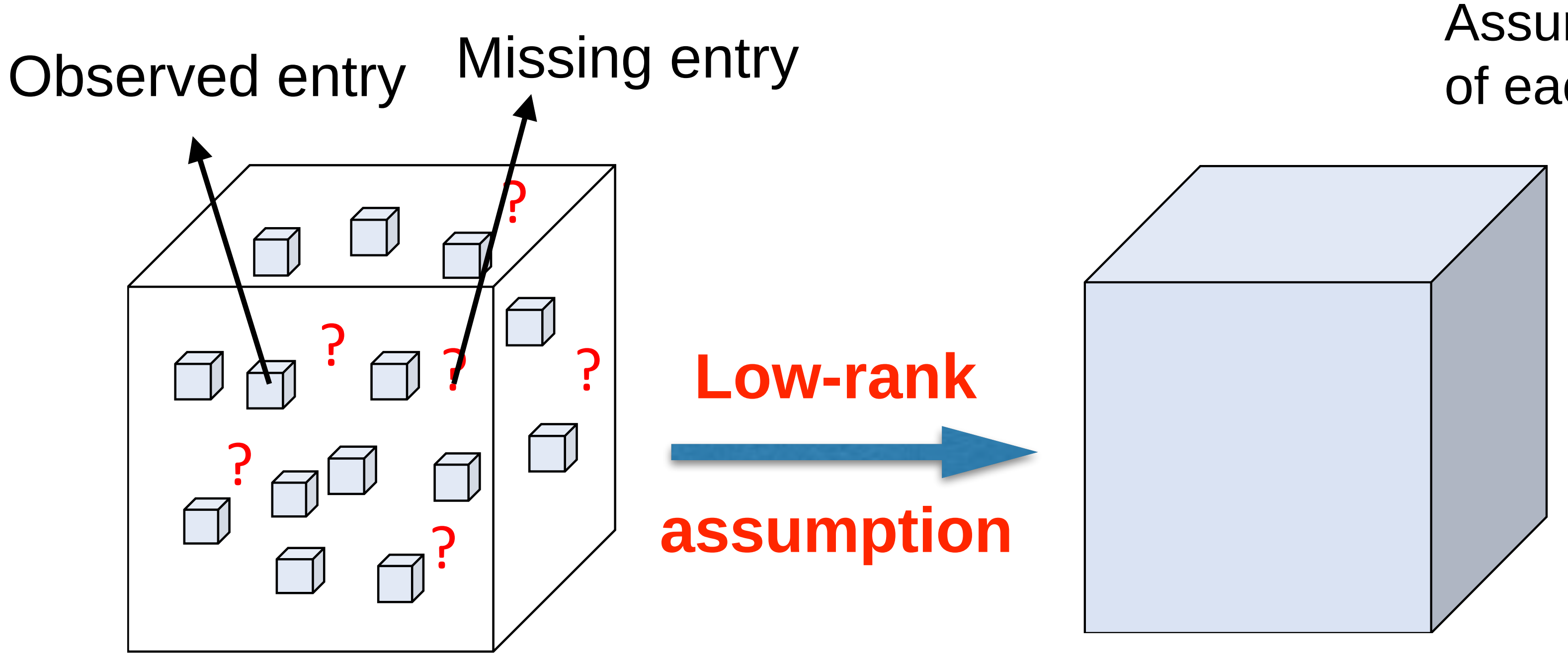
Tensor Completion

Solving scheme 1:
low-rank assumption on tensor

Example: High accuracy LRTC (HaLRTC)

$$\boxed{\begin{aligned} \min_{\mathcal{X}} :\quad & \|\mathcal{X}\|_* \\ s.t.:\quad & \mathcal{X}_\Omega = \mathcal{T}_\Omega \end{aligned}} \xrightarrow{\text{HaLRTC}} \boxed{\begin{aligned} \min_{\mathcal{X}} :\quad & \sum_{i=1}^{n} \alpha_i \|\mathcal{X}_{(i)}\|_* \\ s.t.:\quad & \mathcal{X}_\Omega = \mathcal{T}_\Omega . \end{aligned}}$$

Assume the tensor matricization of each mode is low -rank



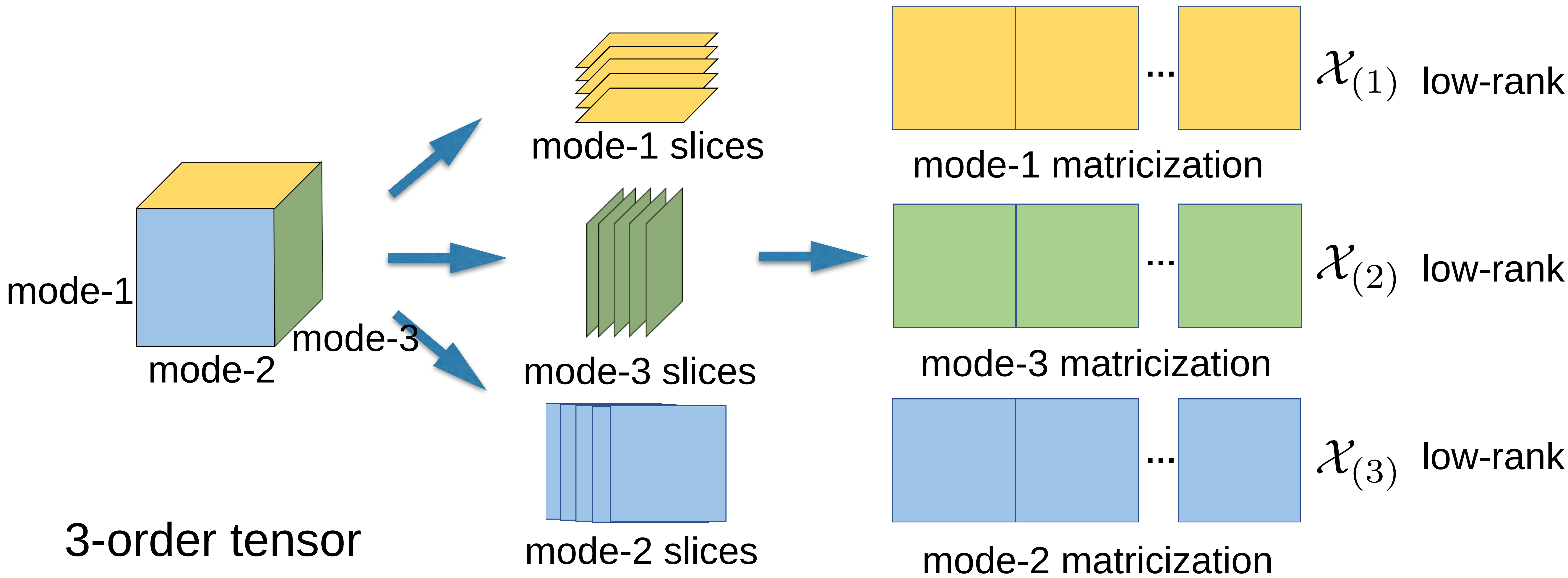
Incomplete tensor

Completed tensor

[Liu, et al., 2013]

Tensor Completion

Mode-n matricization of a three-order tensor:



[Kolda, et al., 2009]

Technical problems

- Model selection problem
    - Rank determination; tuning parameter selection
- Uncertainty information (confidence region)
    - Point estimation by ML, MAP, or optimisation methods
    - Overfitting problem
- Efficiency (MCMC, Gibbs inference - easy derivation but slow convergence; no analytic solution)

Tensor factorization with missing values

- Problem: Nth-order tensor is partially observed.

$$\boldsymbol{\mathcal{Y}} = \boldsymbol{\mathcal{X}} + \boldsymbol{\varepsilon}, \qquad \boldsymbol{\varepsilon} \sim \prod_{i_1,\ldots,i_N} \mathcal{N}(0, \tau^{-1})$$

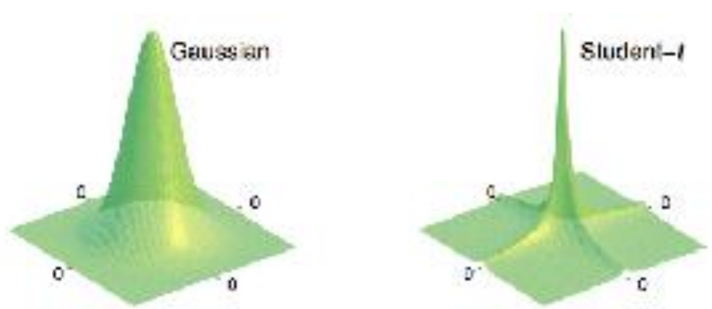
Ω indicates observed indices $\qquad \mathcal{O}$ is a indicator tensor

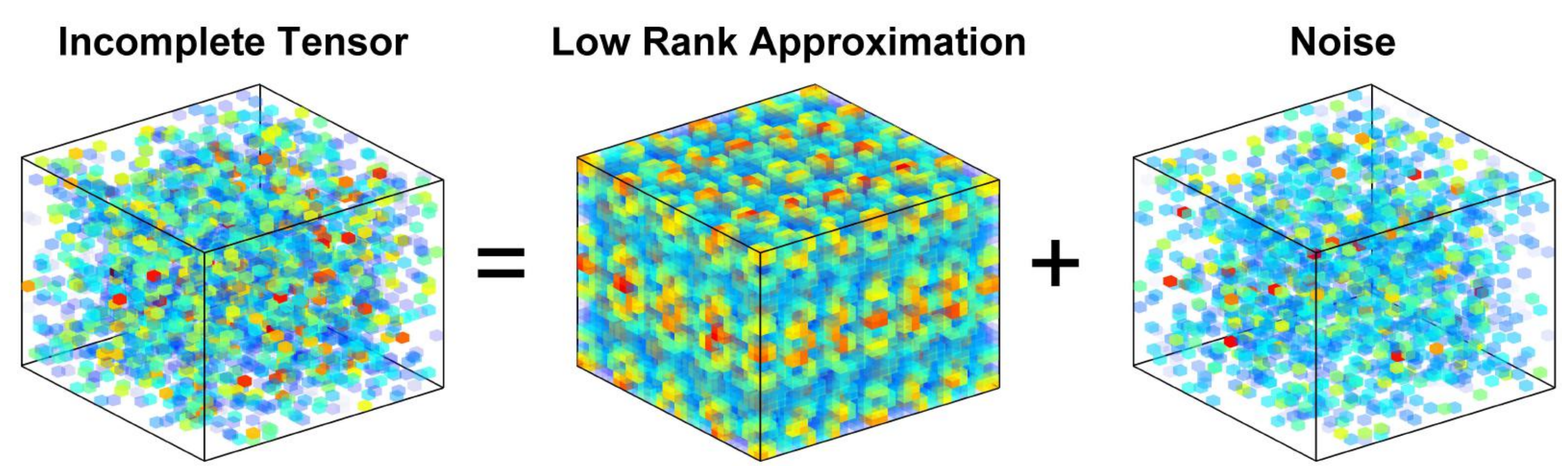
- True latent tensor is represented by a CP model with the minimum R

$$\boldsymbol{\mathcal{X}} = \sum_{r=1}^{R} \mathbf{a}_r^{(1)} \circ \cdots \circ \mathbf{a}_r^{(N)} = [\![\mathbf{A}^{(1)}, \ldots, \mathbf{A}^{(N)}]\!],$$

- Sparsity imposed on latent dimensions of factors

$$\mathcal{T}(x|0,\lambda,\nu) = \int \mathcal{N}(x|0,\tau) Ga(\tau|a,b) d\tau$$

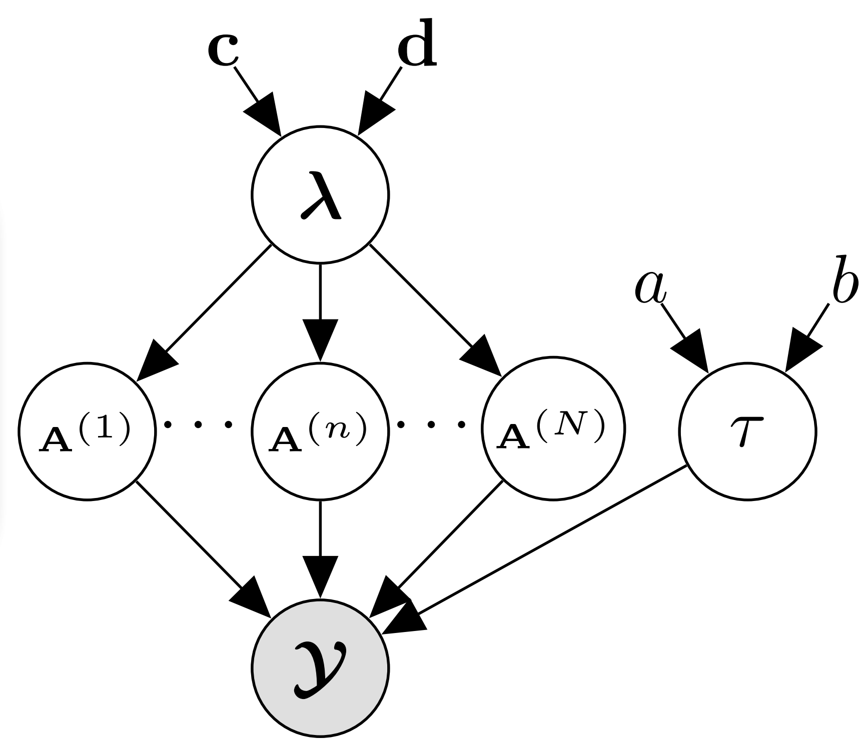




Bayesian CP factorization

- Observation model (likelihood)

$$p\left(\mathcal{Y}_\Omega \middle| \{\mathbf{A}^{(n)}\}_{n=1}^N, \tau\right) = \prod_{i_1=1}^{I_1} \cdots \prod_{i_N=1}^{I_N} \mathcal{N}\left(\mathcal{Y}_{i_1 i_2 \ldots i_N} \middle| \left\langle \mathbf{a}_{i_1}^{(1)}, \mathbf{a}_{i_2}^{(2)}, \cdots, \mathbf{a}_{i_N}^{(N)} \right\rangle, \tau^{-1}\right)^{\mathcal{O}_{i_1 \cdots i_n}},$$



- Priors of latent factors

$$p(\mathbf{A}^{(n)}|\boldsymbol{\lambda}) = \prod_{i_n=1}^{I_n} \mathcal{N}(\mathbf{a}_{i_n}^{(n)}|\mathbf{0}, \boldsymbol{\Lambda}^{-1}), \forall n \in [1, N], \qquad \boldsymbol{\Lambda} = \mathrm{diag}(\boldsymbol{\lambda})$$

- Priors of hyper parameters

$$p(\boldsymbol{\lambda}) = \prod_{r=1}^{R} \mathrm{Ga}(\lambda_r|c_0^r, d_0^r), \qquad p(\tau) = \mathrm{Ga}(\tau|a_0, b_0). \qquad \mathrm{Ga}(x|a,b) = \frac{b^a x^{a-1} e^{-bx}}{\Gamma(a)}$$

Q. Zhao et al, IEEE TPAMI 2015

Our objective

- The posterior distribution of all unknowns

$$p(\Theta|\boldsymbol{\mathcal{Y}}_\Omega) = \frac{p(\Theta, \boldsymbol{\mathcal{Y}}_\Omega)}{\int p(\Theta, \boldsymbol{\mathcal{Y}}_\Omega)\, d\Theta}. \qquad \Theta = \{\mathbf{A}^{(1)}, \ldots, \mathbf{A}^{(N)}, \boldsymbol{\lambda}, \tau\}$$

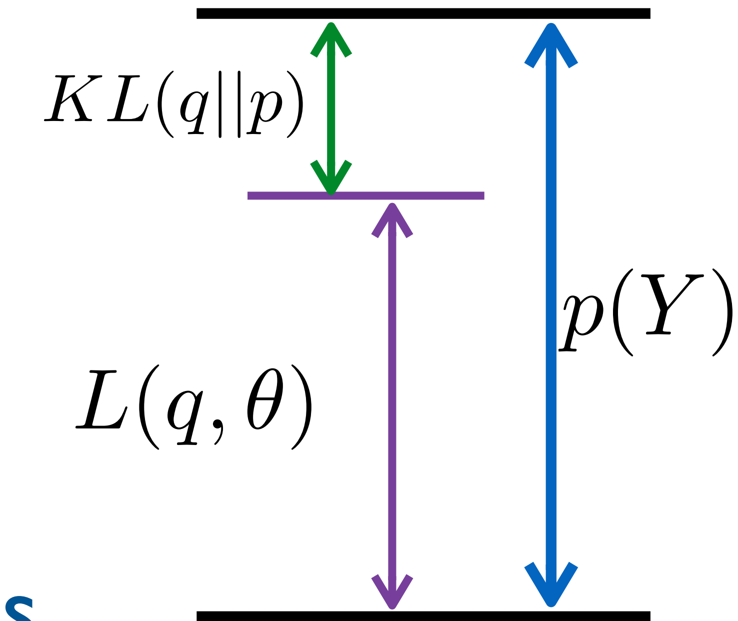
- Predictive distribution for missing entries

$$p(\boldsymbol{\mathcal{Y}}_{\backslash\Omega}|\boldsymbol{\mathcal{Y}}_\Omega) = \int p(\boldsymbol{\mathcal{Y}}_{\backslash\Omega}|\Theta)p(\Theta|\boldsymbol{\mathcal{Y}}_\Omega)\,\mathrm{d}\Theta,$$

- Analytic intractable and resort to approximate inference
  - Variation Bayesian inference; Expectation propagation
  - Sampling methods such as MCMC gibbs

Model learning via Bayesian Inference

- KL divergence between approximation and true posterior distributions



$$\begin{aligned}\mathrm{KL}\left(q(\Theta)||p(\Theta|\boldsymbol{\mathcal{Y}})\right) &= \int q(\Theta)\ln\left\{\frac{q(\Theta)}{p(\Theta|\boldsymbol{\mathcal{Y}})}\right\} \\ &= \ln p(\boldsymbol{\mathcal{Y}}) - \int q(\Theta)\ln\left\{\frac{p(\boldsymbol{\mathcal{Y}},\Theta)}{q(\Theta)}\right\} d\Theta\end{aligned}$$

- Factorization of approximation distributions

$$q(\Theta) = q_\lambda(\boldsymbol{\lambda})q_\tau(\tau)\prod_{n=1}^{N} q_n\left(\mathbf{A}^{(n)}\right).$$

- Approximation for posterior distributions

$$q_n(\mathbf{A}^{(n)}) = \prod_{i_n=1}^{I_n}\mathcal{N}\left(\mathbf{a}_{i_n}^{(n)}\middle|\tilde{\mathbf{a}}_{i_n}^{(n)},\mathbf{V}_{i_n}^{(n)}\right), \qquad q_{\boldsymbol{\lambda}}(\boldsymbol{\lambda}) = \prod_{r=1}^{R}\mathrm{Ga}(\lambda_r|c_M^r, d_M^r), \quad q_\tau(\tau) = \mathrm{Ga}(\tau|a_M, b_M),$$

Model learning

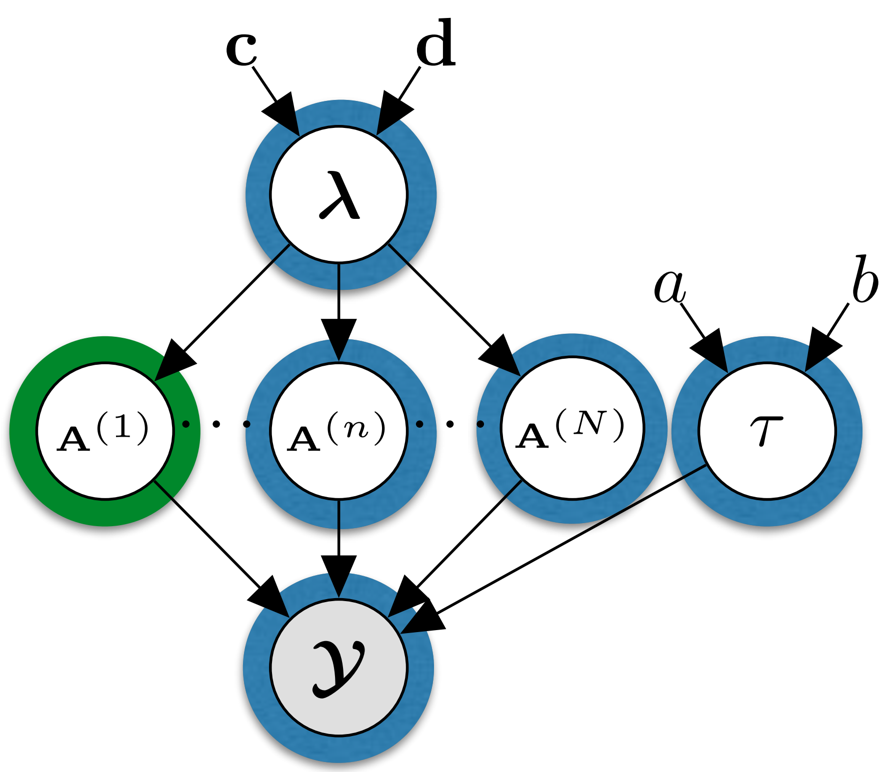
- Posterior of latent factors

$$q_n(\mathbf{A}^{(n)}) = \prod_{i_n=1}^{I_n} \mathcal{N}\left(\mathbf{a}_{i_n}^{(n)} \middle| \tilde{\mathbf{a}}_{i_n}^{(n)}, \mathbf{V}_{i_n}^{(n)}\right),$$

$$\tilde{\mathbf{a}}_{i_n}^{(n)} = \mathbb{E}_q[\tau]\mathbf{V}_{i_n}^{(n)}\mathbb{E}_q\left[\mathbf{A}_{i_n}^{(\setminus n)T}\right]\mathrm{vec}\left(\boldsymbol{\mathcal{Y}}_{\mathbb{I}(\mathcal{O}_{i_n}=1)}\right)$$

$$\mathbf{V}_{i_n}^{(n)} = \left(\mathbb{E}_q[\tau]\mathbb{E}_q\left[\mathbf{A}_{i_n}^{(\setminus n)T}\mathbf{A}_{i_n}^{(\setminus n)}\right] + \mathbb{E}_q[\boldsymbol{\Lambda}]\right)^{-1},$$

$$\mathbf{A}_{i_n}^{(\setminus n)T} = \left(\bigodot_{k\neq n}\mathbf{A}^{(k)}\right)^{T}_{\mathbb{I}(\mathcal{O}_{i_n}=1)},$$



Variational Message Passing

Model learning

- Posterior of hyper parameters- precision of latent factor

$$q_{\boldsymbol{\lambda}}(\boldsymbol{\lambda}) = \prod_{r=1}^{R} \mathrm{Ga}(\lambda_r | c_M^r, d_M^r), \qquad c_M^r = c_0^r + \frac{1}{2}\sum_{n=1}^{N} I_n$$

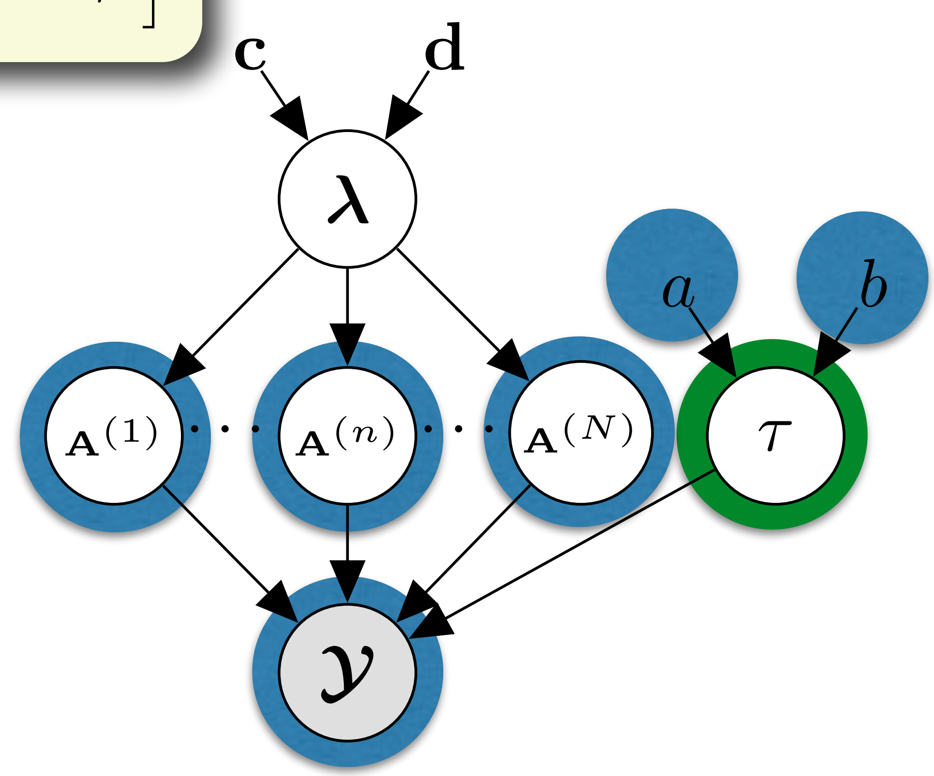
$$d_M^r = d_0^r + \frac{1}{2}\sum_{n=1}^{N} \mathbb{E}_q\left[\mathbf{a}_{\cdot r}^{(n)T}\mathbf{a}_{\cdot r}^{(n)}\right].$$

- Posterior of noise precision

$$q_\tau(\tau) = \mathrm{Ga}(\tau | a_M, b_M),$$

$$a_M = a_0 + \frac{1}{2}\sum_{i_1,\ldots,i_N} \mathcal{O}_{i_1,\ldots,i_N}$$

$$b_M = b_0 + \frac{1}{2}\mathbb{E}_q\left[\left\|\mathcal{O} \circledast \left(\mathcal{Y} - [\![\mathbf{A}^{(1)},\ldots,\mathbf{A}^{(N)}]\!]\right)\right\|_F^2\right]. \tag{29}$$



Variational Message Passing

Demonstration of learning procedure

- Size 10x10x10
- Rank =5
- $\forall n, \forall i_n, \mathbf{a}_{i_n}^{(n)} \sim \mathcal{N}(\mathbf{0}, \mathbf{I}_R),$
- $\varepsilon \sim \prod_{i_1,\ldots,i_N} \mathcal{N}(0, \sigma^2)$

  $\sigma^{-2} = 1000$

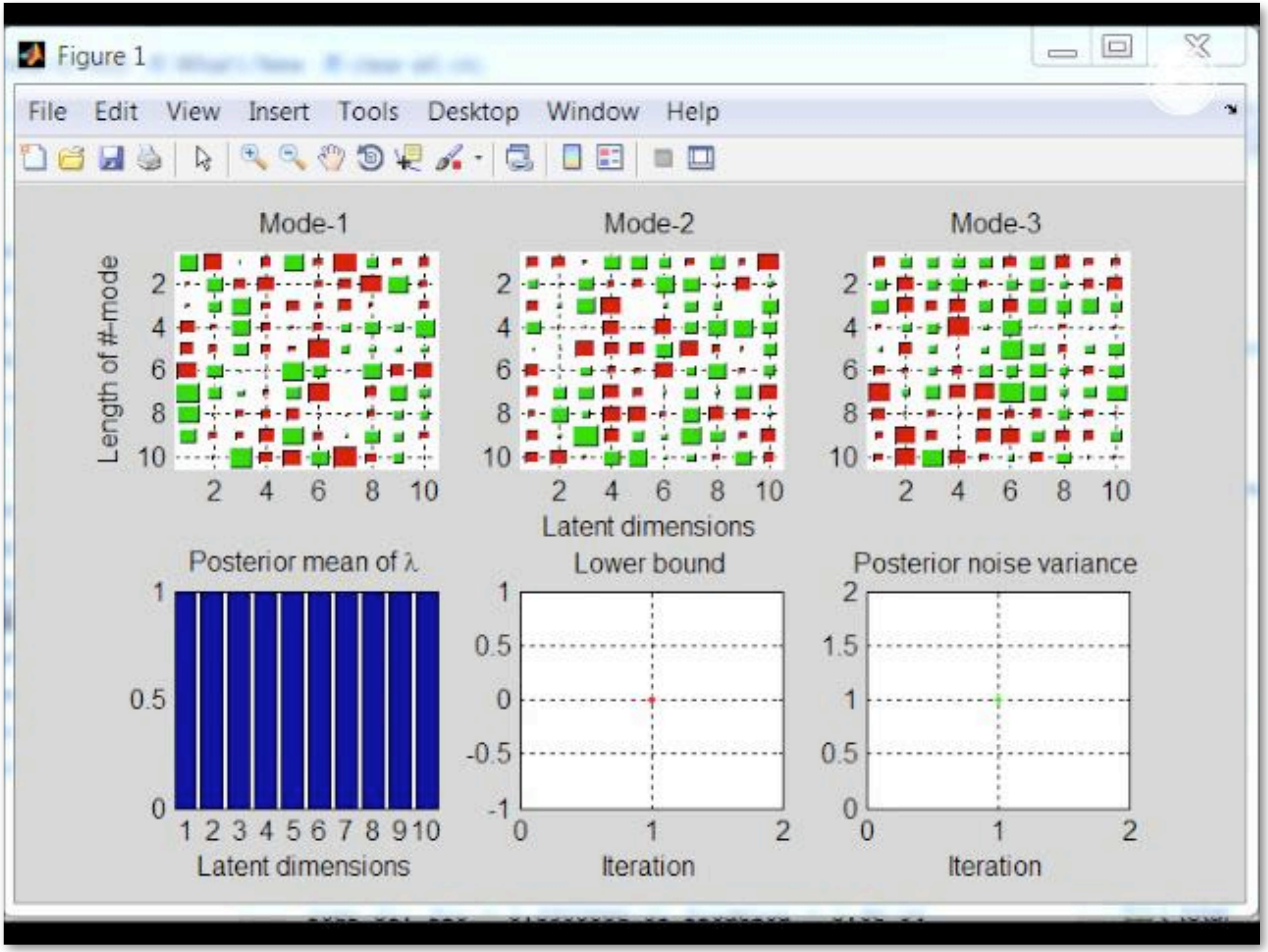
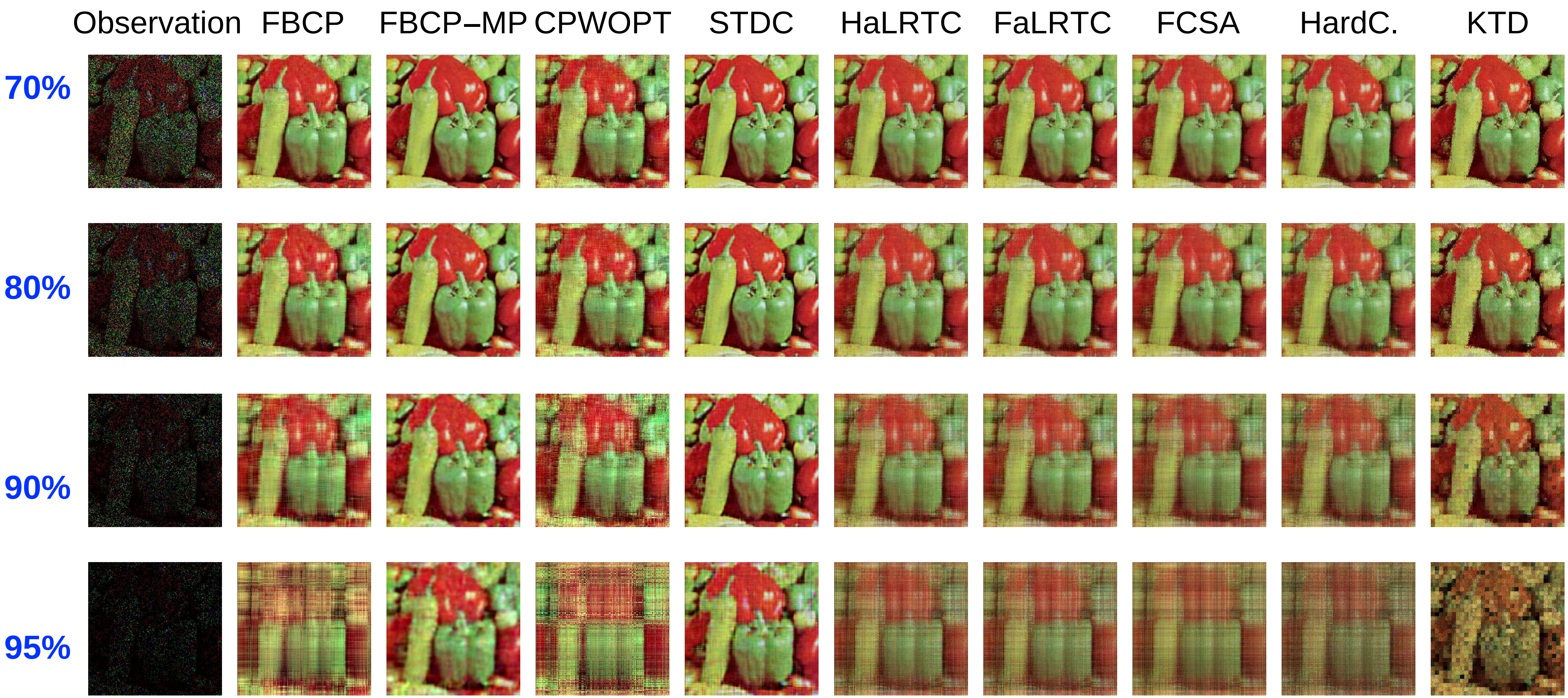


Image Completion

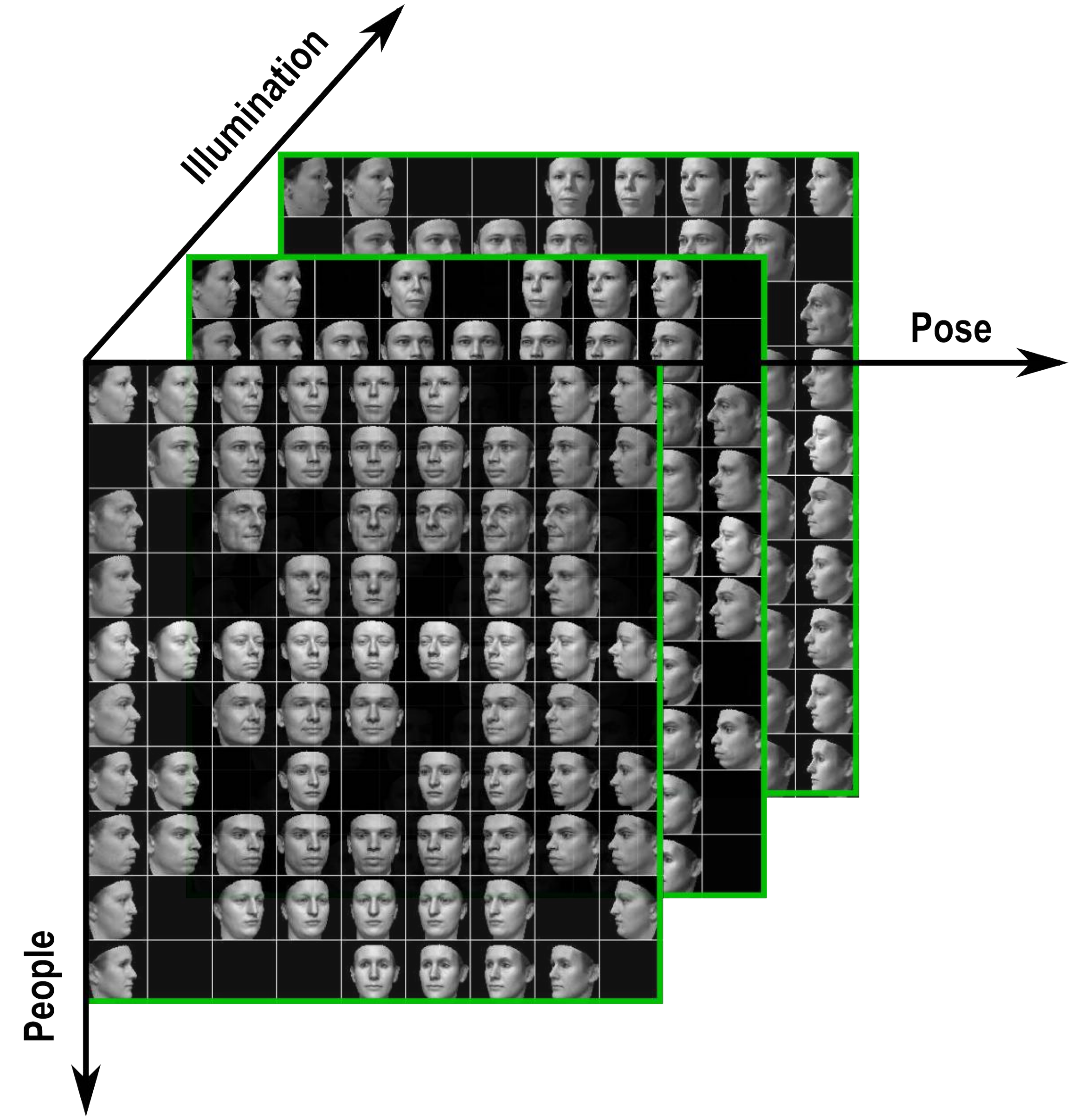
Missing rate



Facial image synthesis

Matrix factorization does not work when one entire row or column is missing

- 3D basel face model
- image size 68 x 68
- 10 people x 9 poses x 3 illuminations
- large variants of faces captured from surveillance video
- Robust face recognition



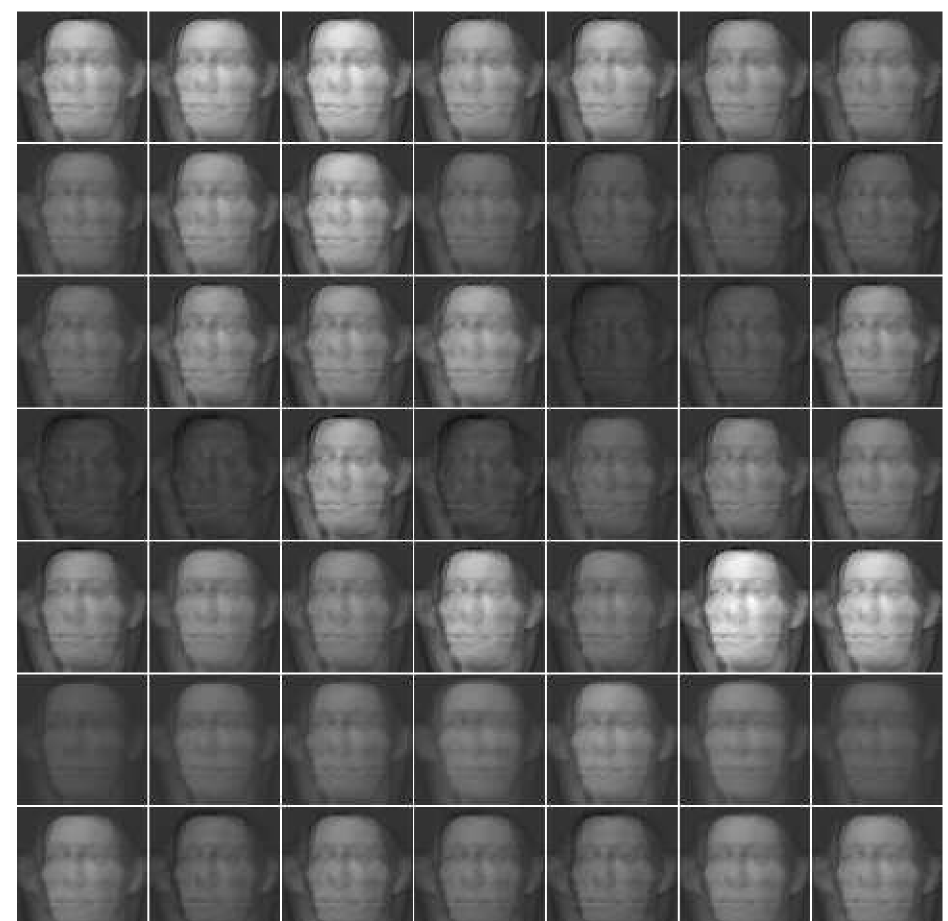
Method	36/270		49/270		64/270		81/270	
	T	M	T	M	T	M	T	M
FBCP	**0.06**	**0.10**	**0.06**	**0.10**	**0.09**	**0.15**	**0.12**	**0.20**
CPWOPT	0.53	0.65	0.56	0.61	0.58	0.59	0.65	0.73
FaLRTC	0.11	0.28	0.13	0.30	0.15	0.31	0.19	0.34
HardC.	0.37	0.37	0.37	0.40	0.37	0.40	0.37	0.40

Ground truth

FALRTC

FBCP

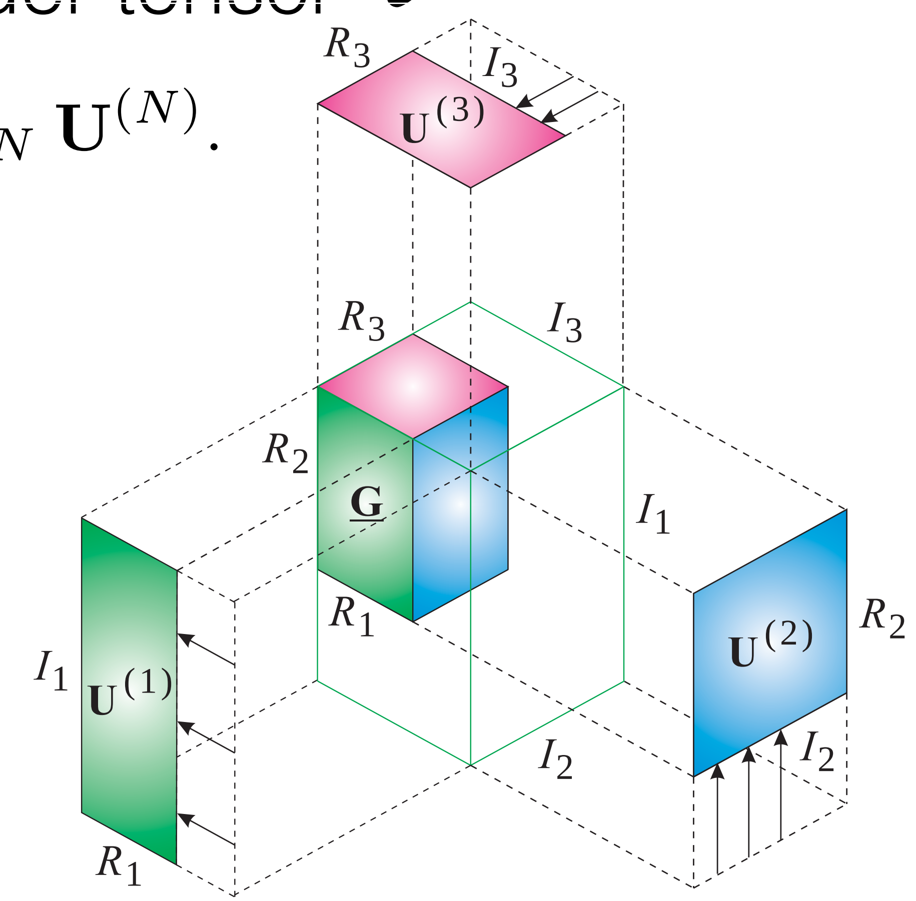
CPWOPT



Bayesian Sparse Tucker Decomposition

- Model assumption: Observed *N*th-order tensor $\mathcal{Y} \in \mathbb{R}^{I_1 \times \cdots \times I_N}$

$$\mathcal{Y} = \mathcal{X} + \varepsilon, \qquad \mathcal{X} = \mathcal{G} \times_1 \mathbf{U}^{(1)} \times_2 \mathbf{U}^{(2)} \times \cdots \times_N \mathbf{U}^{(N)}.$$



- Likelihood function:

$$\mathrm{vec}(\mathcal{Y}) \Big| \left\{\mathbf{U}^{(n)}\right\}, \mathcal{G}, \tau \sim \mathcal{N}\left(\left(\bigotimes_n \mathbf{U}^{(n)}\right)\mathrm{vec}(\mathcal{G}),\ \tau^{-1}\mathbf{I}\right)$$

- Priors over model parameters:

$$\begin{aligned}
\tau &\sim Ga(a_0^\tau, b_0^\tau),\\
\mathrm{vec}(\mathcal{G}) \Big| \left\{\boldsymbol{\lambda}^{(n)}\right\}, \beta &\sim \mathcal{N}\left(\mathbf{0},\ \left(\beta \bigotimes_n \boldsymbol{\Lambda}^{(n)}\right)^{-1}\right),\\
\beta &\sim Ga(a_0^\beta, b_0^\beta),\\
\mathbf{u}_{i_n}^{(n)} \big| \boldsymbol{\lambda}^{(n)} &\sim \mathcal{N}\left(\mathbf{0},\ \boldsymbol{\Lambda}^{(n)^{-1}}\right), \quad \forall n, \forall i_n,\\
\text{Student-t:}\quad \lambda_{r_n}^{(n)} &\sim Ga(a_0^\lambda, b_0^\lambda), \quad \forall n, \forall r_n,\\
\text{Laplace:}\quad \lambda_{r_n}^{(n)} &\sim IG(1, \tfrac{\gamma}{2}), \quad \forall n, \forall r_n,\\
\gamma &\sim Ga(a_0^\gamma, b_0^\gamma).
\end{aligned}$$

- ❖ Group Sparsity priors over factors
- ❖ Slice sparsity priors over cores
- ❖ Shared sparsity patterns between cores and factors

Joint distribution of the model

$$p(\mathcal{Y}, \Theta) = p(\mathcal{Y}|\{\mathbf{U}^{(n)}\}, \mathcal{G}, \tau) \prod_n p(\mathbf{U}^{(n)}|\boldsymbol{\lambda}^{(n)}) \times p(\mathcal{G}|\{\boldsymbol{\Lambda}^{(n)}\}, \beta) \prod_n p(\boldsymbol{\lambda}^{(n)}|\gamma) p(\gamma) p(\beta) p(\tau).$$

Model Inference

- Variational Bayesian

$$q(\Theta) = q(\boldsymbol{\mathcal{G}})q(\beta)\prod_n q(\mathbf{U}^{(n)})\prod_n q(\boldsymbol{\lambda}^{(n)})q(\gamma)q(\tau).$$

- Posterior of the core tensor

$$q(\boldsymbol{\mathcal{G}}) = \mathcal{N}\left(\mathrm{vec}(\boldsymbol{\mathcal{G}})\middle|\mathrm{vec}(\tilde{\boldsymbol{\mathcal{G}}}), \Sigma_G\right),$$

$$\mathrm{vec}(\tilde{\boldsymbol{\mathcal{G}}}) = \mathbb{E}[\tau]\,\Sigma_G\left(\bigotimes_n \mathbb{E}\left[\mathbf{U}^{(n)T}\right]\right)\mathrm{vec}\left(\boldsymbol{\mathcal{Y}}\right),$$

$$\Sigma_G = \left\{\mathbb{E}[\beta]\bigotimes_n \mathbb{E}\left[\boldsymbol{\Lambda}^{(n)}\right] + \mathbb{E}[\tau]\bigotimes_n \mathbb{E}\left[\mathbf{U}^{(n)T}\mathbf{U}^{(n)}\right]\right\}^{-1}.$$

- Posterior of noise precision τ

$$q(\tau) = Ga(a_M^\tau, b_M^\tau)$$

$$a_M^\tau = a_0^\tau + \frac{1}{2}\prod_n I_n,$$

$$b_M^\tau = b_0^\tau + \frac{1}{2}\mathbb{E}\left[\left\|\mathrm{vec}(\boldsymbol{\mathcal{Y}}) - \left(\bigotimes_n \mathbf{U}^{(n)}\right)\mathrm{vec}(\boldsymbol{\mathcal{G}})\right\|_F^2\right]$$

- Posterior of factor matrices

$$q(\mathbf{U}^{(n)}) = \prod_{i_n=1}^{I_n}\mathcal{N}\left(\mathbf{u}_{i_n}^{(n)}\middle|\tilde{\mathbf{u}}_{i_n}^{(n)}, \Psi^{(n)}\right),\ n = 1,\dots,N,$$

$$\tilde{\mathbf{U}}^{(n)} = \mathbb{E}[\tau]\,\mathbf{Y}_{(n)}\left(\bigotimes_{k\neq n}\mathbb{E}\left[\mathbf{U}^{(k)}\right]\right)\mathbb{E}\left[\mathbf{G}_{(n)}^T\right]\Psi^{(n)}, \tag{17}$$

$$\Psi^{(n)} = \left\{\mathbb{E}[\boldsymbol{\Lambda}^{(n)}] + \mathbb{E}[\tau]\mathbb{E}\left[\mathbf{G}_{(n)}\left(\bigotimes_{k\neq n}\mathbf{U}^{(k)T}\mathbf{U}^{(k)}\right)\mathbf{G}_{(n)}^T\right]\right\}^{-1}. \tag{18}$$

- Posterior of $\boldsymbol{\lambda}^{(n)}, \quad n = 1,\dots,N$

$$q(\boldsymbol{\lambda}^{(n)}) = \prod_{r_n=1}^{R_n} Ga(\lambda_{r_n}^{(n)}|\tilde{a}_{r_n}^{(n)}, \tilde{b}_{r_n}^{(n)}),$$

$$\tilde{a}_{r_n}^{(n)} = a_0^\lambda + \frac{1}{2}\left(I_n + \prod_{k\neq n} R_k\right),$$

$$\tilde{b}_{r_n}^{(n)} = b_0^\lambda + \frac{1}{2}\mathbb{E}\left[\mathbf{u}_{\cdot r_n}^{(n)T}\mathbf{u}_{\cdot r_n}^{(n)}\right] + \frac{1}{2}\mathbb{E}[\beta]\mathbb{E}\left[\mathrm{vec}(\boldsymbol{\mathcal{G}}^2_{\cdots r_n \cdots})^T\right]\bigotimes_{k\neq n}\mathbb{E}\left[\boldsymbol{\lambda}^{(k)}\right].$$

Bayesian Sparse Tucker Completion

- Model assumption: Nth-order tensor $\mathcal{Y} \in \mathbb{R}^{I_1 \times \cdots \times I_N}$

$$\mathcal{Y}_\Omega = \mathcal{X}_\Omega + \varepsilon \qquad \mathcal{X} = \mathcal{G} \times_1 \mathbf{U}^{(1)} \times_2 \mathbf{U}^{(2)} \times \cdots \times_N \mathbf{U}^{(N)}.$$

Ω denotes a set of N-tuple indices $\qquad \mathcal{O}_{i_1 \cdots i_N} = 1 \text{ if } (i_1, \ldots, i_N) \in \Omega$

- Likelihood function:

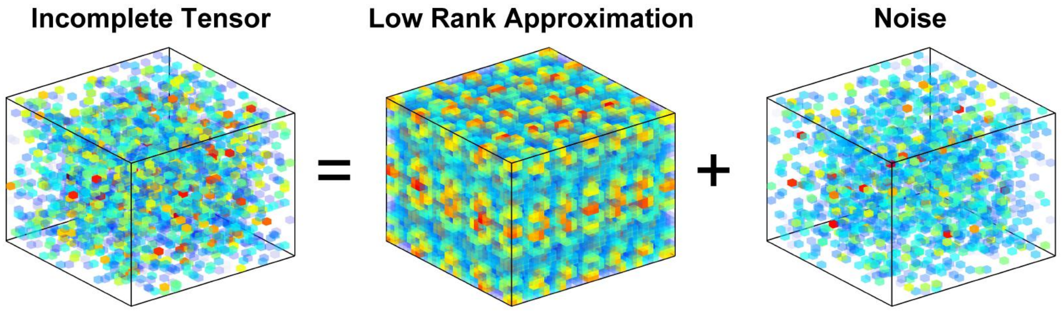
$$\mathcal{Y}_{i_1 \cdots i_N} \Big| \left\{ \mathbf{u}_{i_n}^{(n)} \right\}, \mathcal{G}, \tau \sim \mathcal{N}\left(\left(\bigotimes_n \mathbf{u}_{i_n}^{(n)T} \right) \mathrm{vec}(\mathcal{G}),\ \tau^{-1} \right)^{\mathcal{O}_{i_1 \cdots i_N}}$$

- Priors over model parameters are same as BSTD

- Model inference are different for core tensor ***G***, factor matrices ***U***, and noise precision τ

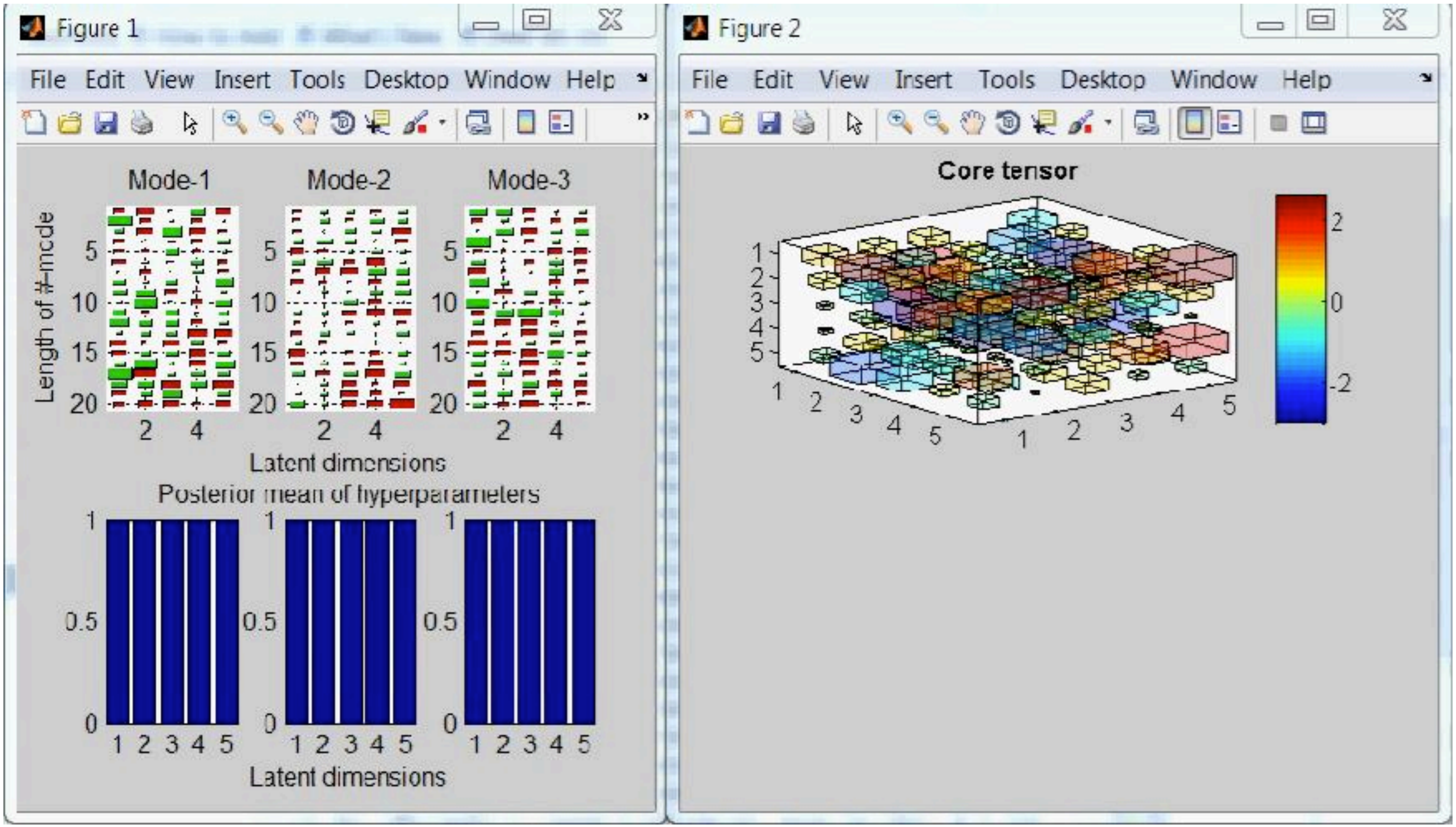
- Predictive distribution over missing entries

$$p(\mathcal{Y}_{i_1 \cdots i_N} | \mathcal{Y}_\Omega) = \int p(\mathcal{Y}_{i_1 \cdots i_N} | \Theta) p(\Theta | \mathcal{Y}_\Omega) \, \mathrm{d}\Theta$$

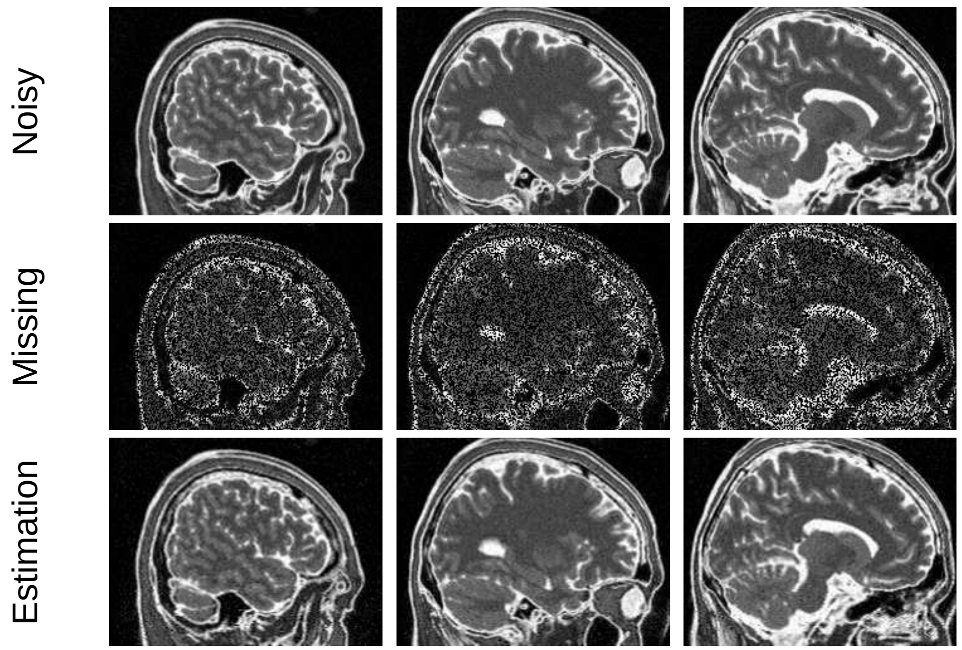


Demonstration of Learning Procedure

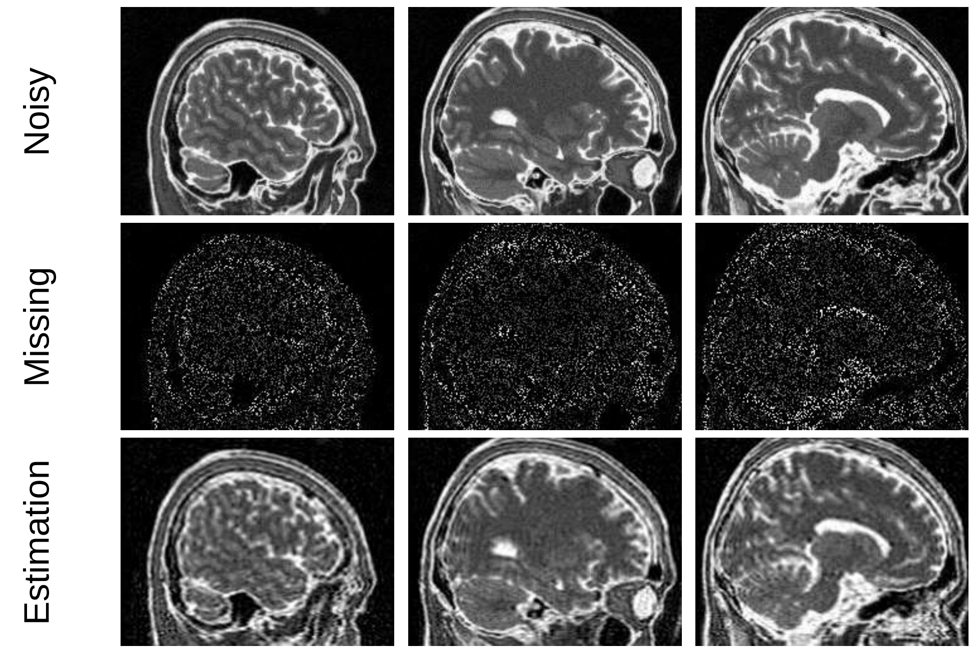
- Tensor: 20 x 20 x 20 with 70% missing elements
- Multilinear rank: 2 x 3 x 3



MRI Dataset



(a) 50% missing



(b) 80% missing

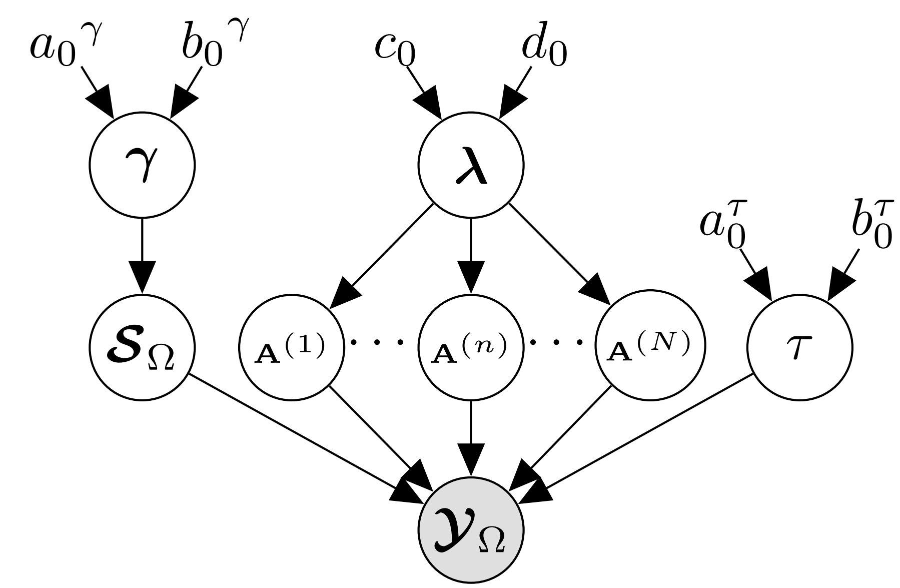
TABLE III
THE PERFORMANCE OF MRI COMPLETION EVALUATED BY PSNR AND RRSE. FOR NOISY MRI, THE STANDARD DERIVATION OF GAUSSIAN NOISE IS 3% OF BRIGHTEST TISSUE. MRI TENSOR IS OF SIZE $181 \times 217 \times 165$ AND EACH BLOCK TENSOR IS OF SIZE $50 \times 50 \times 10$.

	50%				60%				70%				80%			
	Original		Noisy		Original		Noisy		Original		Noisy		Original		Noisy	
BSTC-T	27.32	0.11	26.18	0.12	25.30	0.14	24.60	0.15	22.81	0.18	22.35	0.19	20.14	0.25	20.00	0.25
BSTC-L	26.91	0.11	25.57	0.13	24.84	0.15	23.95	0.16	22.76	0.19	22.09	0.20	20.12	0.25	19.80	0.26
iHOOI	22.69	0.19	21.45	0.22	22.47	0.19	21.16	0.22	21.63	0.21	20.11	0.25	18.65	0.30	17.89	0.32
HaLRTC	24.84	0.15	23.60	0.17	22.35	0.19	21.65	0.21	19.93	0.26	19.55	0.27	17.37	0.34	17.15	0.35

Bayesian Robust Tensor Factorization

- **Model specification**

$$p\left(\mathcal{Y}_\Omega \middle| \{\mathbf{A}^{(n)}\}_{n=1}^{N}, \mathcal{S}_\Omega, \tau\right) = \prod_{i_1=1}^{I_1} \cdots \prod_{i_N=1}^{I_N} \mathcal{N}\left(\mathcal{Y}_{i_1 \ldots i_N} \middle| \left\langle \mathbf{a}_{i_1}^{(1)}, \cdots, \mathbf{a}_{i_N}^{(N)} \right\rangle + \mathcal{S}_{i_1 \ldots i_N}, \tau^{-1}\right)^{\mathcal{O}_{i_1 \cdots i_N}}, \tag{6}$$



- **Priors**

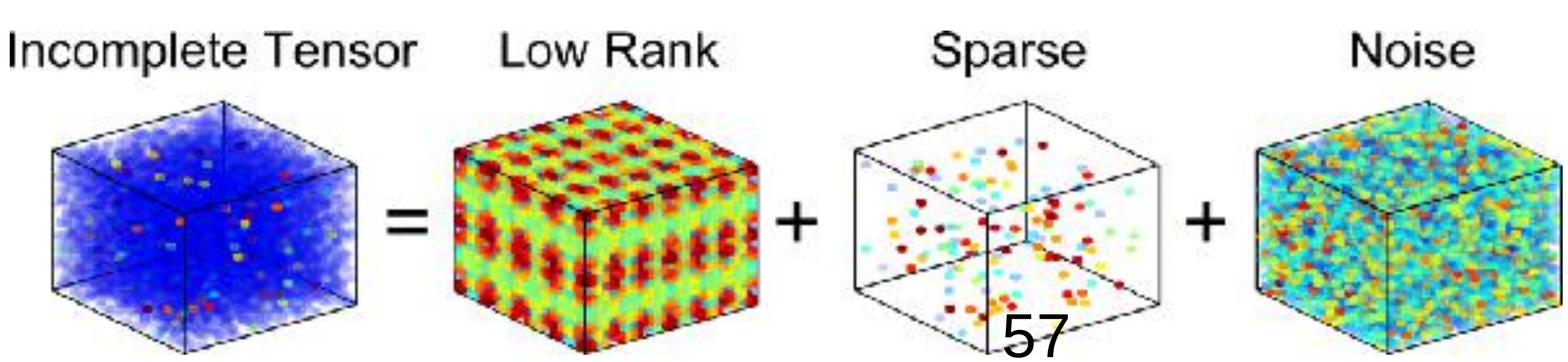
$$p(\tau) = \mathrm{Ga}(\tau | a_0^{\tau}, b_0^{\tau}).$$

$$p(\mathbf{A}^{(n)} | \boldsymbol{\lambda}) = \prod_{i_n=1}^{I_n} \mathcal{N}(\mathbf{a}_{i_n}^{(n)} | \mathbf{0}, \boldsymbol{\Lambda}^{-1}), \ \forall n \in [1, N]$$
$$p(\boldsymbol{\lambda}) = \prod_{r=1}^{R} \mathrm{Ga}(\lambda_r | c_0, d_0), \tag{7}$$

$$p(\mathcal{S}_\Omega | \boldsymbol{\gamma}) = \prod_{i_1, \ldots, i_N} \mathcal{N}(\mathcal{S}_{i_1 \ldots i_N} | 0, \gamma_{i_1 \ldots i_N}^{-1})^{\mathcal{O}_{i_1 \cdots i_N}},$$
$$p(\boldsymbol{\gamma}) = \prod_{i_1, \ldots, i_N} \mathrm{Ga}(\gamma_{i_1 \ldots i_N} | a_0^{\gamma}, b_0^{\gamma}). \tag{8}$$

- **Joint distribution**

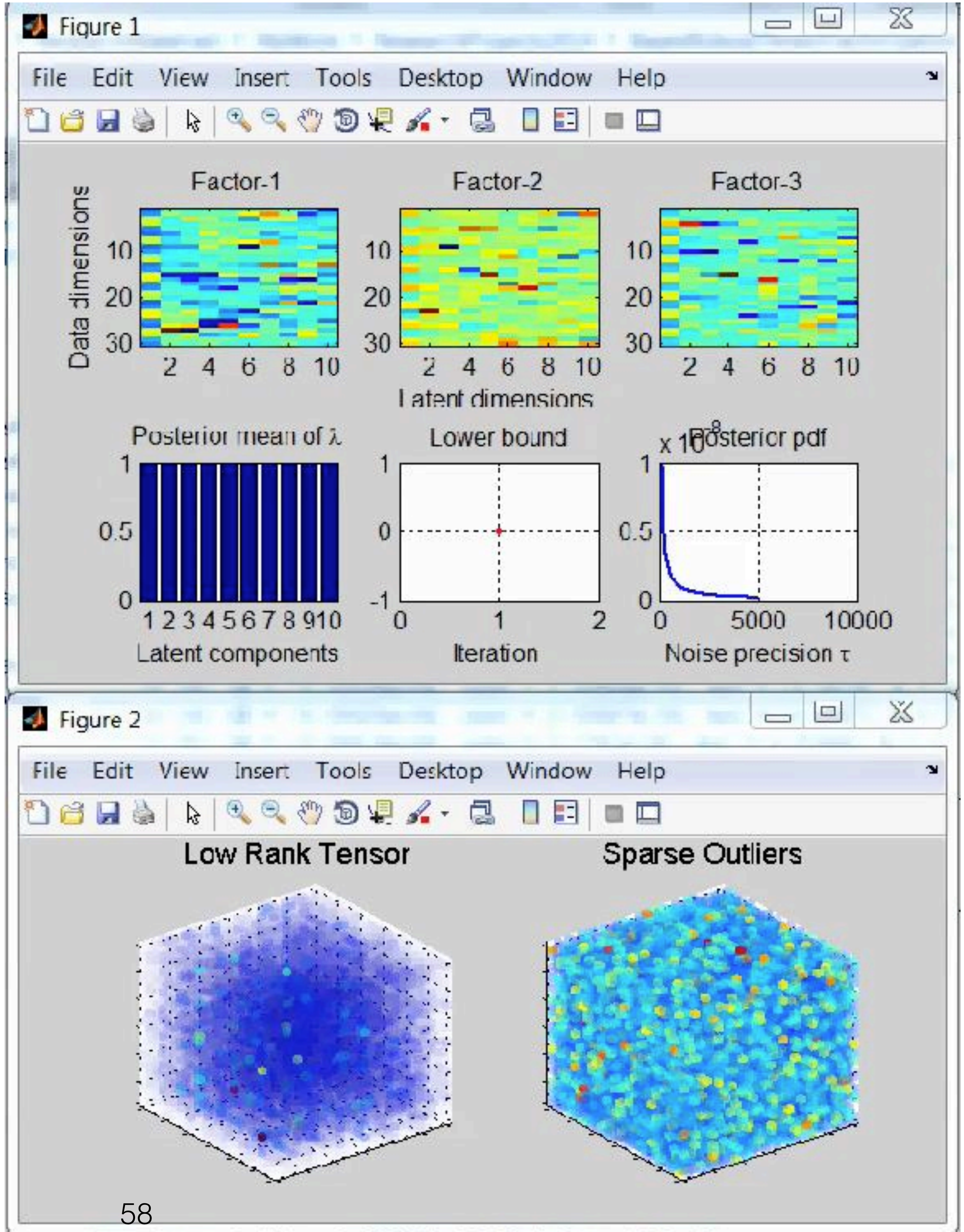
$$p\left(\mathcal{Y}_\Omega \middle| \{\mathbf{A}^{(n)}\}_{n=1}^{N}, \mathcal{S}_\Omega, \tau\right) \prod_{n=1}^{N} p(\mathbf{A}^{(n)} | \boldsymbol{\lambda}) p(\mathcal{S}_\Omega | \boldsymbol{\gamma}) p(\boldsymbol{\lambda}) p(\boldsymbol{\gamma}) p(\tau).$$

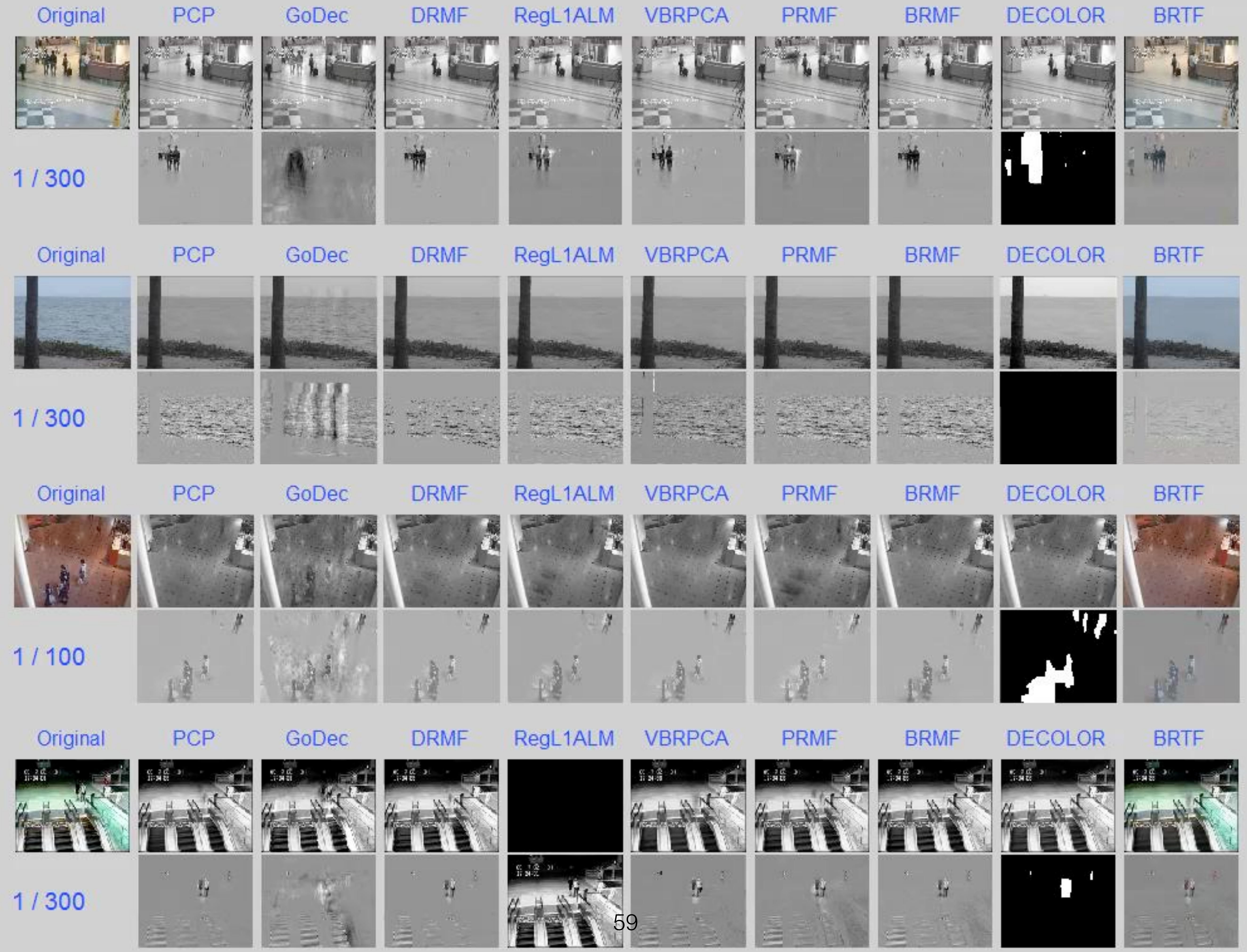


Q. Zhao et al, IEEE TNNLS,2016

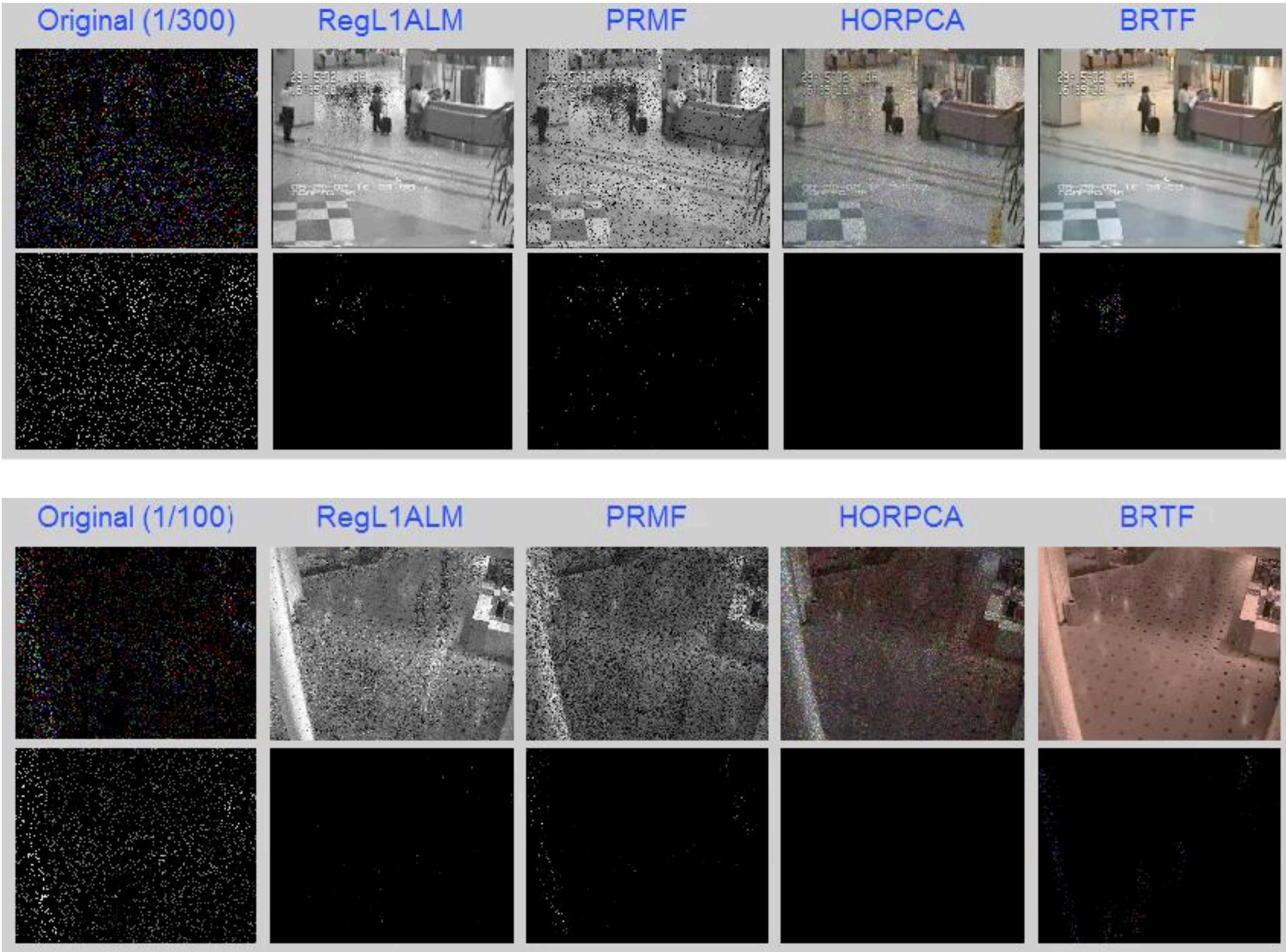
Demo of the model learning procedure

- Tensor size: 30 x 30 x 30
- CP rank: R = 3
- Gaussian noise: SNR = 20dB
- Missing rate: 80%
- Outliers: rate = 5%, M= 10*std(X);
- Maximal rank is set to 10.




Original
PCP
GoDec
DRMF
RegL1ALM
VBRPCA
PRMF
BRMF
DECOLOR
BRTF
1 / 300
1 / 300
1 / 100
1 / 300

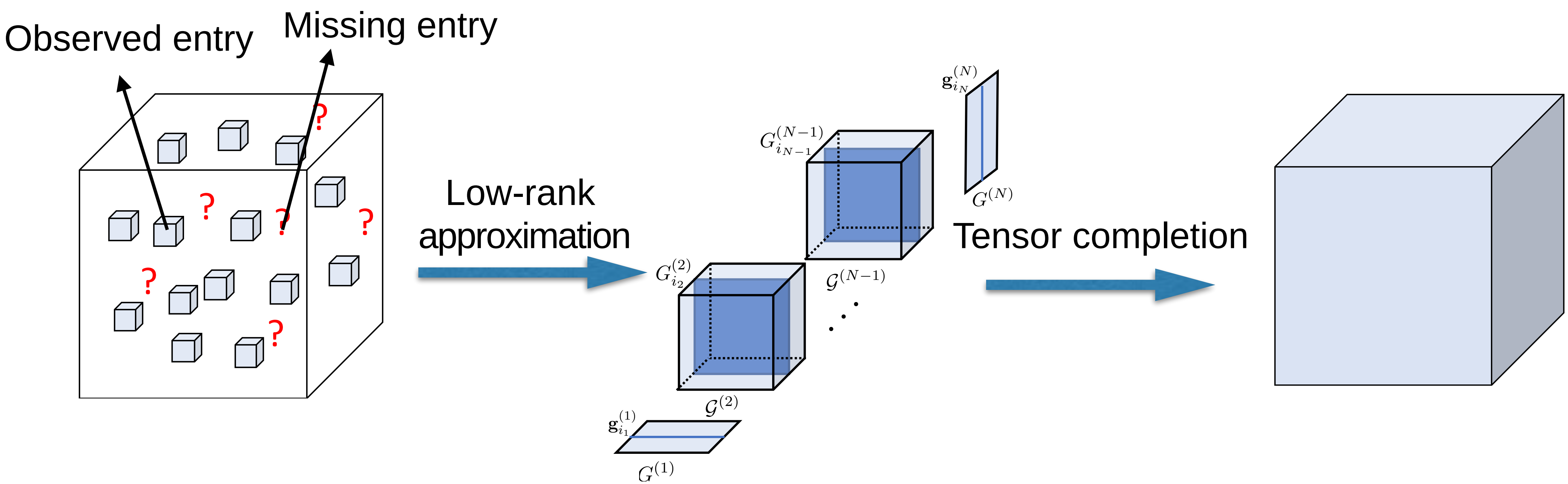
Videos with 90% missing pixels



Tensor Completion

Solving scheme 3:
tensor decomposition by gradient-based optimization

Find the low-rank tensor decomposition by observed entries.



Incomplete tensor

Tensor decomposition

Completed tensor

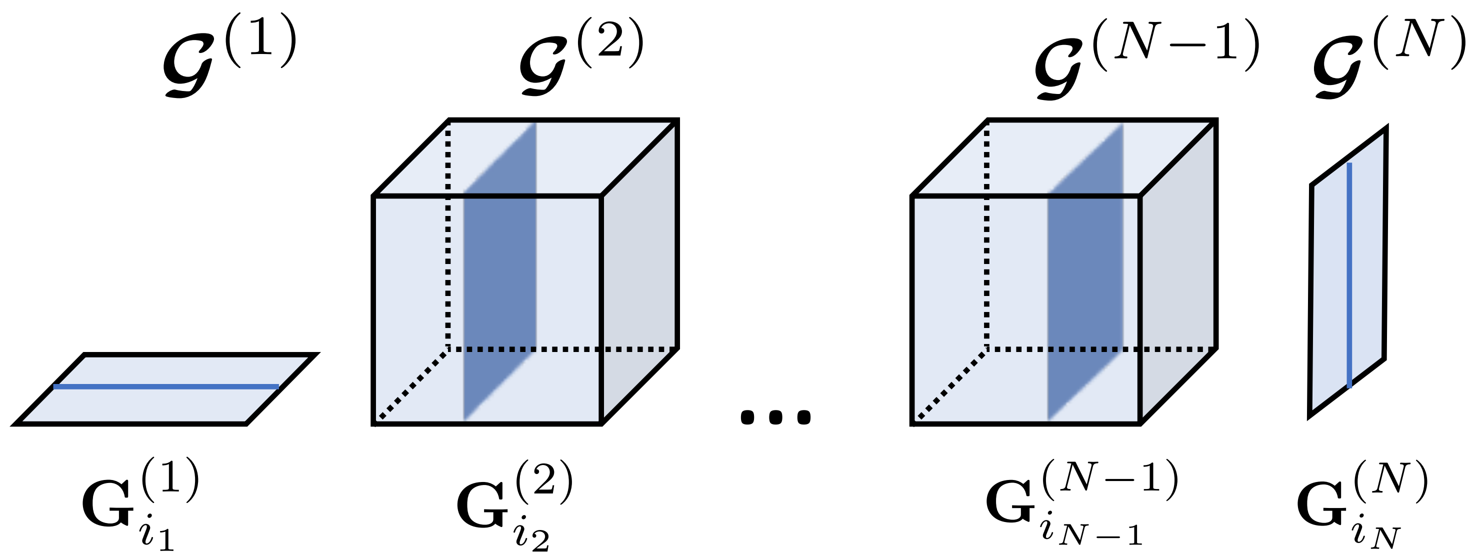
[Yuan, et al., 2017]

Tensor Completion

Tensor train decomposition (TTD)

Decompose a tensor $\boldsymbol{\mathcal{X}} \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_N}$ to TT format:

tensor train: $\boldsymbol{\mathcal{X}}$:



Core tensor: $\boldsymbol{\mathcal{G}}^{(n)} \in \mathbb{R}^{r_{n-1} \times I_n \times r_n}$,

Silce: $\mathbf{G}^{(n)} \in \mathbb{R}^{r_{n-1} \times r_n}$,

TT-rank: $\{r_0, r_1, \cdots, r_N\}, r_0 = r_N = 1,$
$n = 1, 2, \cdots, N.$

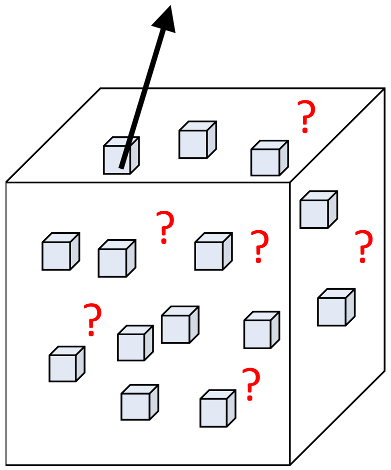
For each element:

$$x_{i_1 \cdots i_N} = \prod_{n=1}^{N} \mathbf{G}^{(n)}_{i_n}$$

[Oseledets, et al., 2011]

Tensor Completion

Tensor train stochastic gradient descent (**TT-SGD**)



For **one** observed entry:

The approximation of TTD: $x_m = \prod_{n=1}^{N} \mathbf{G}_{i_n^m}^{(n)}$

Loss function: $f(\mathbf{G}_{i_1^m}^{(1)}, \mathbf{G}_{i_2^m}^{(2)}, \cdots, \mathbf{G}_{i_N^m}^{(N)}) = \frac{1}{2} \left\| y_m - \prod_{n=1}^{N} \mathbf{G}_{i_n^m}^{(n)} \right\|_F^2$

The gradient for according slice of core tensor:

$$\frac{\partial f}{\partial \mathbf{G}_{i_n^m}^{(n)}} = (x_m - y_m)(\mathbf{G}_{i_n^m}^{>n} \mathbf{G}_{i_n^m}^{<n})^T, n = 1, ..., N$$

Where $\mathbf{G}_{i_n^m}^{>n} = \prod_{n=n+1}^{N} \mathbf{G}_{i_n^m}^{(n)}, \mathbf{G}_{i_n^m}^{<n} = \prod_{n=1}^{n-1} \mathbf{G}_{i_n^m}^{(n)}.$

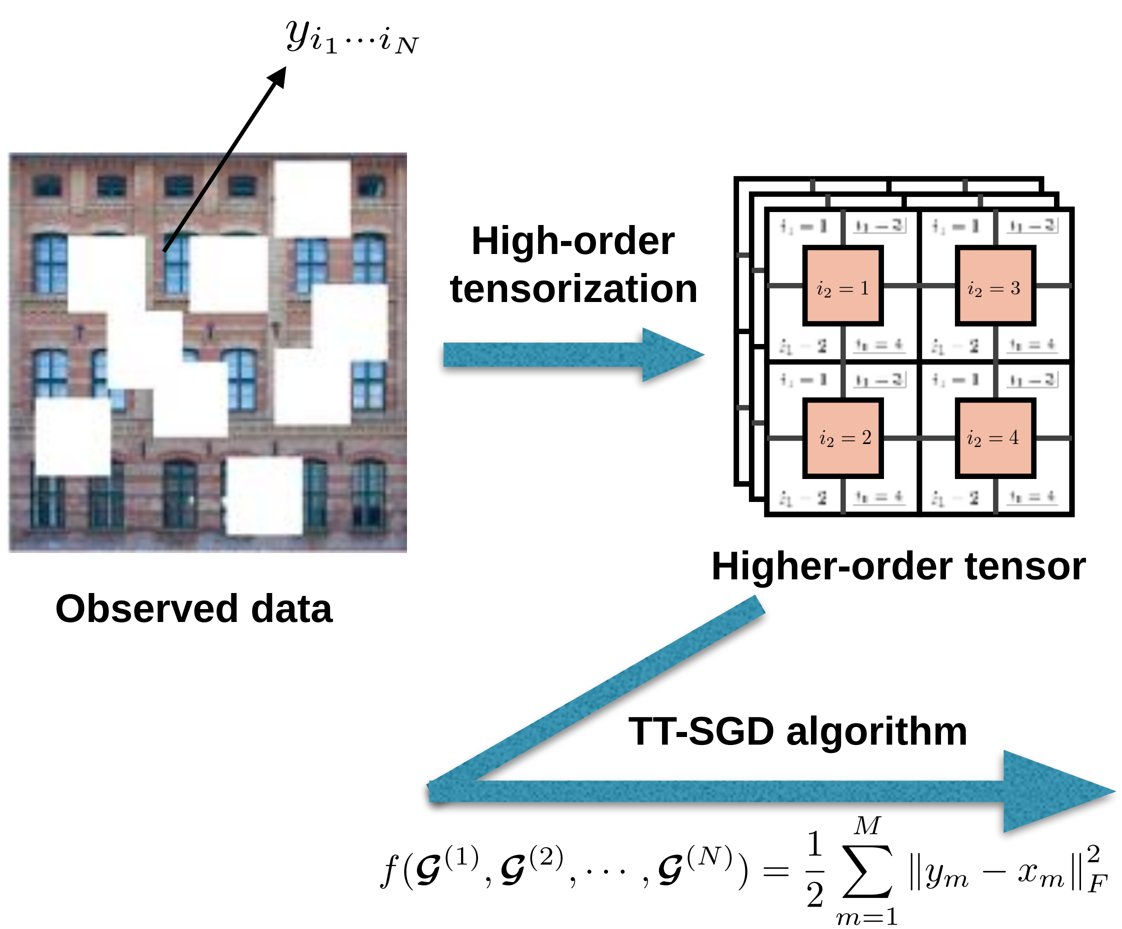
[Yuan, et al., 2018]

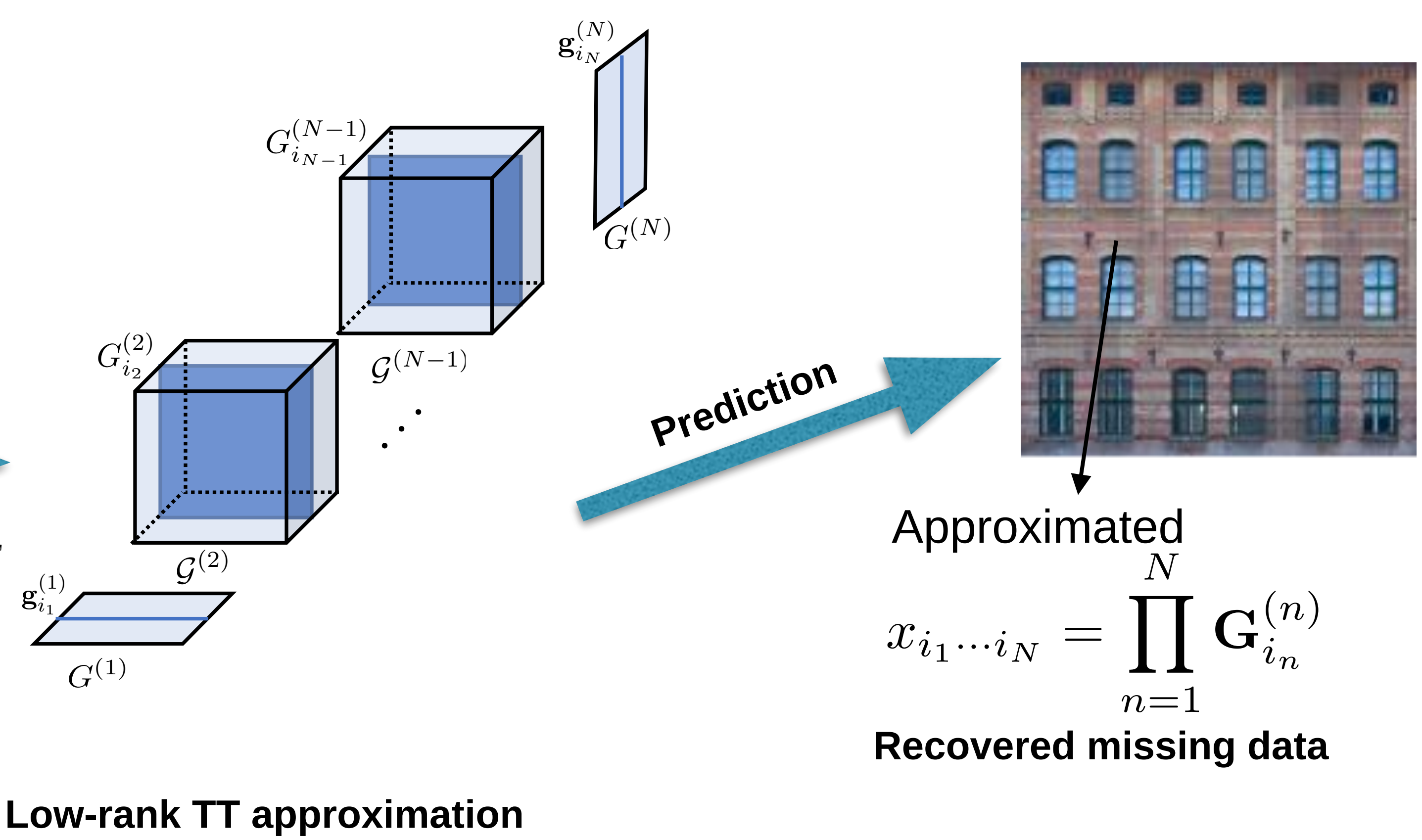
Tensor Completion

TT-SGD overview

Algorithm 2 Tensor-train Stochastic Gradient Descent (TT-SGD)

1: **Input**: incomplete tensor $\mathcal{Y}$ and $TT-rank$ $\mathbf{r}$.
2: **Initialization**: core tensors $\mathcal{G}^{(1)}, \mathcal{G}^{(2)}, \cdots, \mathcal{G}^{(N)}$ of approximated tensor $\mathcal{X}$.
3: **While** the optimization stopping condition is not satisfied
4: Randomly sample one observed entry from $\mathcal{Y}$ *w.r.t.* index $\{i_1, i_2, \cdots i_N\}$.
5: **For** n=1:N
6: Compute the gradients of the according tensor slices by equation (11).
7: **End**
8: Update $\mathbf{G}_{i_1}^{(1)}, \mathbf{G}_{i_2}^{(2)}, \cdots, \mathbf{G}_{i_N}^{(N)}$ by gradient descent method.
9: **End while**
10: **Output**: $\mathcal{G}^{(1)}, \mathcal{G}^{(2)}, \cdots, \mathcal{G}^{(N)}$.

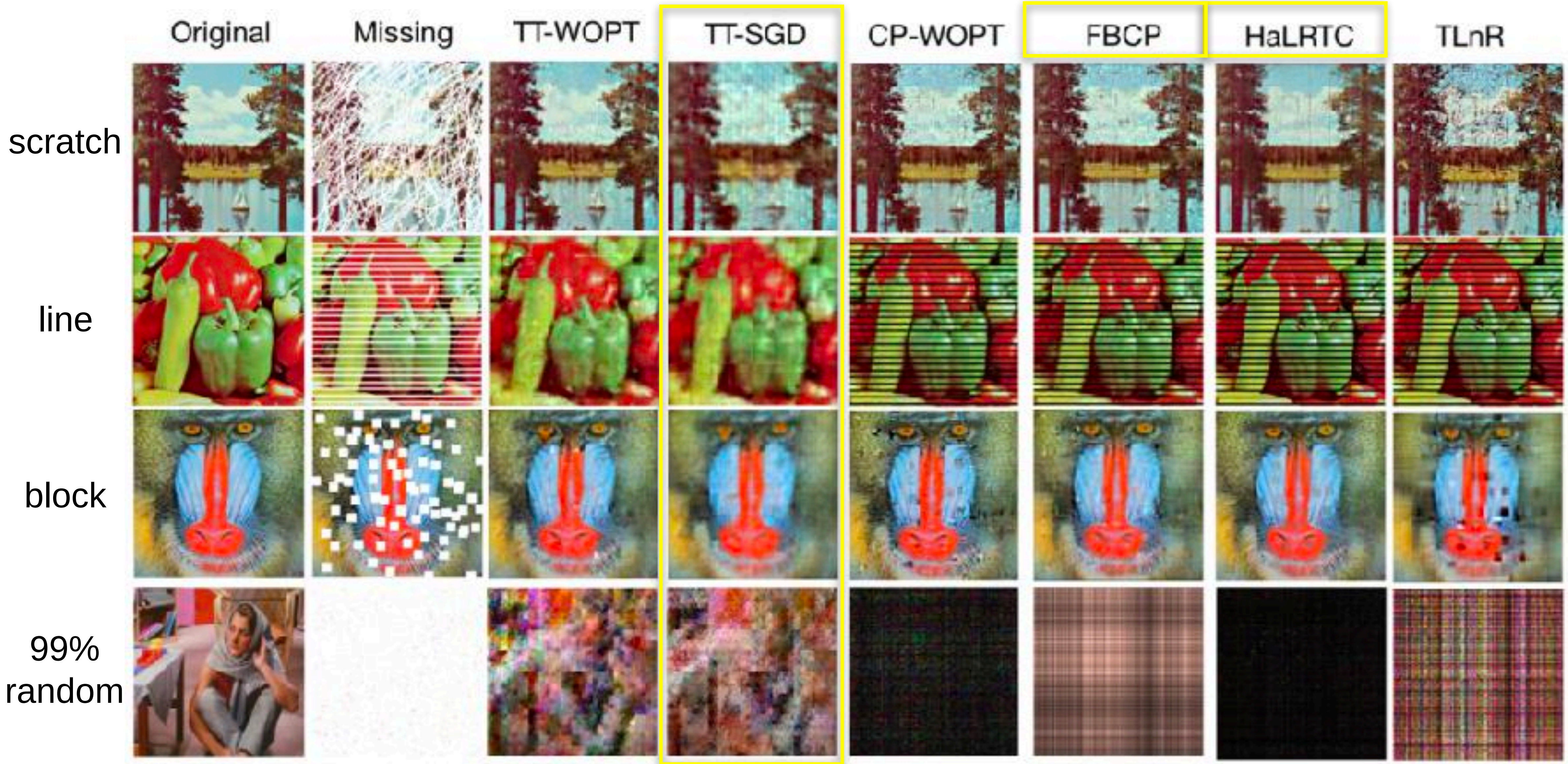




[Yuan, et al., 2018]

Tensor Completion

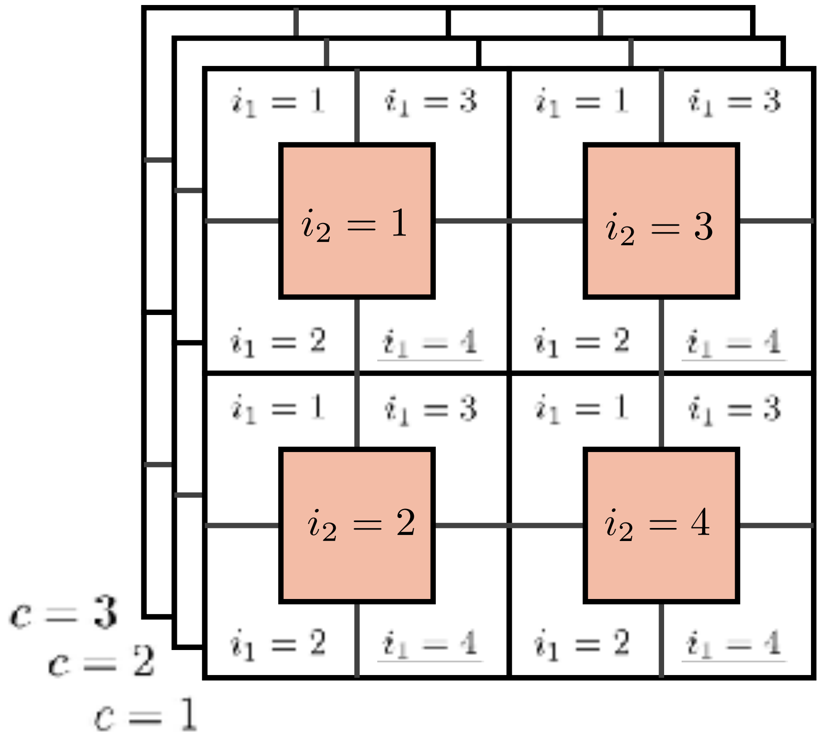
Experiment results

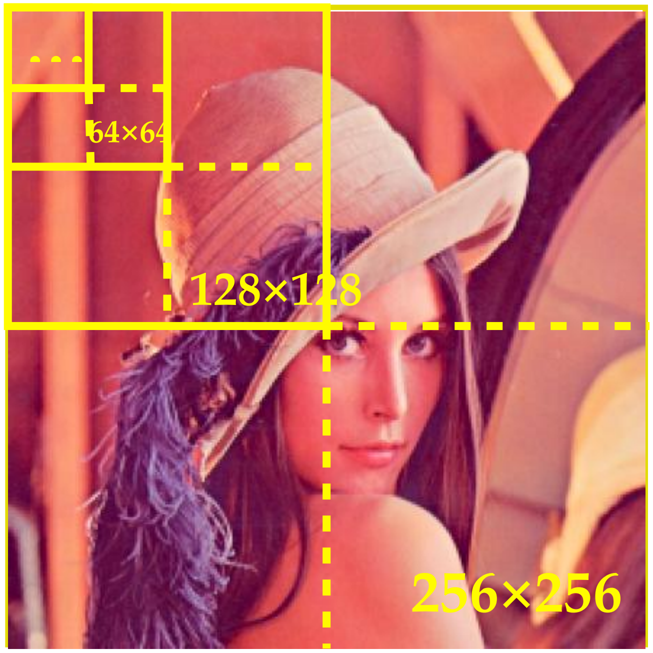


[Yuan, et al., 2018]

Tensor Completion

High-order tensorization





Tensorization for a 256×256×3 image

From 3-way to 9-way

1.Reshape 256×256×3 to 2×2×…×2×3 (17-way tensor).

2.Permute by {1 9 2 10 3 11 4 12 5 13 6 14 7 15 8 16 17}.

3.Reshape to 4×4×4×4×4×4×4×4×3 (9-way tensor).

Better data structure

The first order represent a 2×2 pixel block.

The second order represent four 2×2 pixel block.

…

This can catch more structure relation of data.

[Yuan, et al., 2017]

Tensor Completion

Comparison of applying tensorization

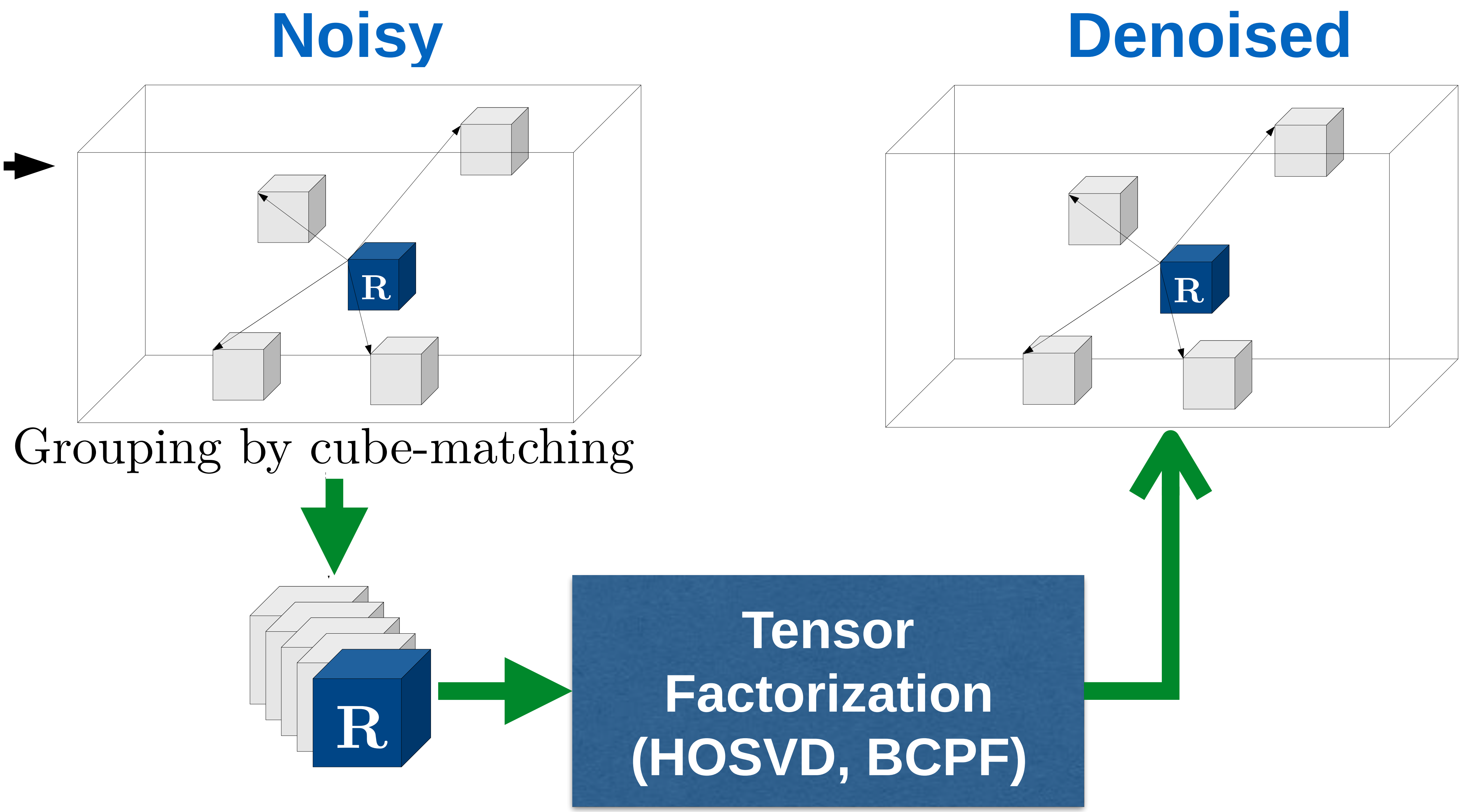
90% random missing

[Yuan, et al., 2017]

Outline

- Tensor Regression
- TensorNets for Deep Neural Networks Compression
- (Multi-)Tensor Completion
- Tensor Denoising

Tensor denoising



IEEE TIP 2013;
IEEE TPAMI 2013

Noise variance is required!!! → **Automatic noise estimation**

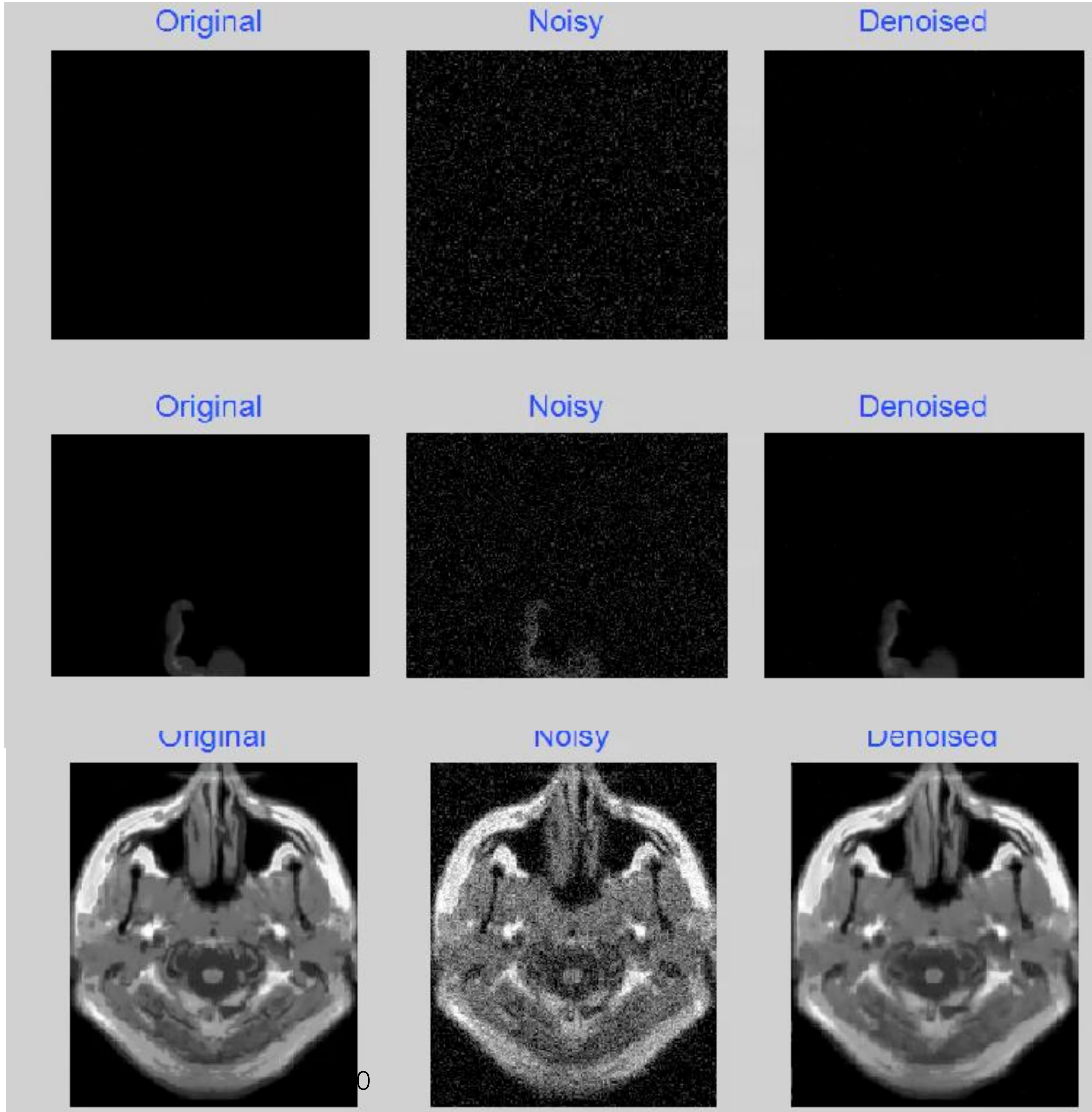
69

Noisy MRI (T1)

- 181 x 217 x 165
- Noise std = 10% max value
- PSNR = 22dB

Denoised MRI

- PSNR = **36dB**



Learning efficient tensor representations with ring structure networks (ICLR Workshop 2018)

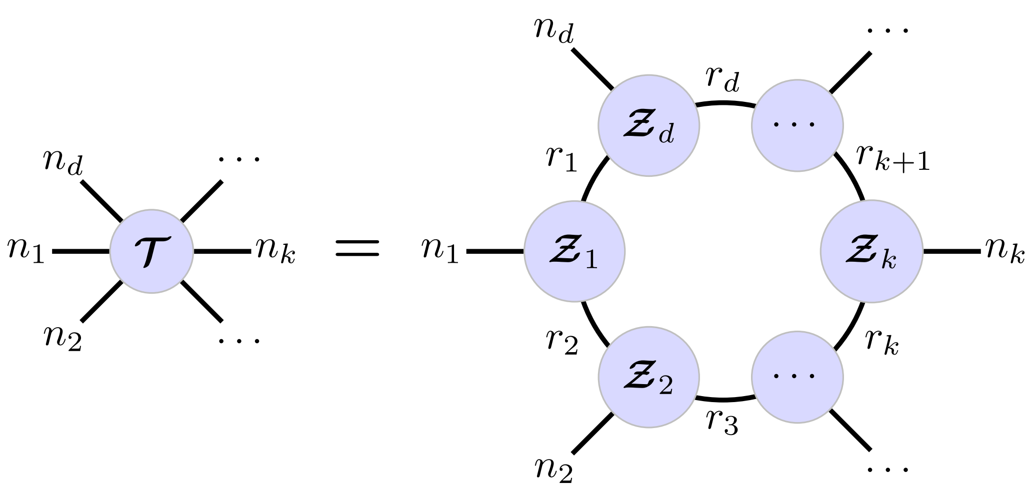
- **Motivation:**
  - Tensor train is too strict due to $r_1 = r_{d+1} = 1$
  - TT-ranks are bounded by the rank of k-unfolding matricization
  - Inconsistent solution from permutation of data
- **Proposed model:**
  - More generalized model without constraint $r_1 = r_{d+1} = 1$
  - Sum of TT with partially shared core tensors
  - Tensor ring ranks: $\exists k, r_1 r_{k+1} \leq R_{k+1}$. $Rank(\mathbf{T}_{\langle k \rangle}) = R_{k+1}$

$$T(i_1, i_2, \ldots, i_d) = \mathrm{Tr}\{\mathbf{Z}_1(i_1)\mathbf{Z}_2(i_2)\cdots\mathbf{Z}_d(i_d)\} = \mathrm{Tr}\left\{\prod_{k=1}^{d} \mathbf{Z}_k(i_k)\right\}.$$

Tensor Ring Decomposition

- Scalar representation

$$T(i_1, i_2, \ldots, i_d) = \sum_{\alpha_1,\ldots,\alpha_d=1}^{r_1,\ldots,r_d} \prod_{k=1}^{d} Z_k(\alpha_k, i_k, \alpha_{k+1}).$$



- Slice representation

$$T(i_1, i_2, \ldots, i_d) = \mathrm{Tr}\left\{\mathbf{Z}_1(i_1)\mathbf{Z}_2(i_2)\cdots\mathbf{Z}_d(i_d)\right\},$$

$$\begin{aligned} T(i_1, i_2, \ldots, i_d) &= \mathrm{Tr}(\mathbf{Z}_2(i_2), \mathbf{Z}_3(i_3), \ldots, \mathbf{Z}_d(i_d), \mathbf{Z}_1(i_1)) \\ &= \cdots = \mathrm{Tr}(\mathbf{Z}_d(i_d), \mathbf{Z}_1(i_1), \ldots, \mathbf{Z}_{d-1}(i_{d-1})). \end{aligned} \quad (4)$$

Circular dimensional permutation invariance

Algorithms:

- Sequential SVDs

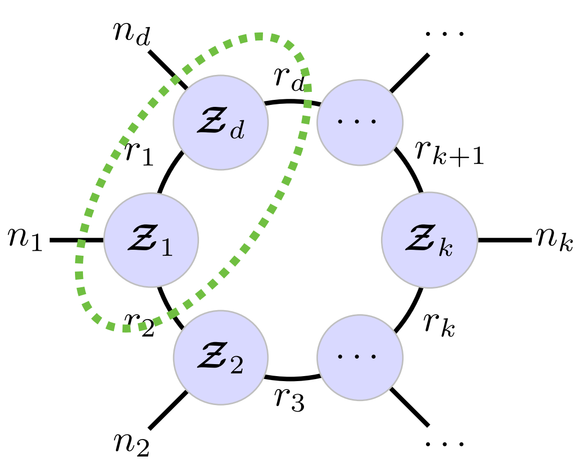
$$T_{\langle 1\rangle}(i_1, \overline{i_2\cdots i_d}) = \sum_{\alpha_1,\alpha_2} Z^{\leq 1}(i_1, \overline{\alpha_1\alpha_2}) Z^{>1}(\overline{\alpha_1\alpha_2}, \overline{i_2\cdots i_d}).$$

$$Z^{>1}(\overline{\alpha_2 i_2}, \overline{i_3\cdots i_d\alpha_1}) = \sum_{\alpha_3} Z_2(\overline{\alpha_2 i_2}, \alpha_3) Z^{>2}(\alpha_3, \overline{i_3\cdots i_d\alpha_1}).$$

- ALS algorithm

$$\mathbf{T}_{[k]} = \mathbf{Z}_{k(2)} \left(\mathbf{Z}^{\neq k}_{[2]}\right)^T,$$

- Block-wise ALS algorithm



Properties of TR Representation

- Sum of tensors

$$\mathcal{T}_1 = \Re(\mathcal{Z}_1, \ldots, \mathcal{Z}_d) \quad \mathcal{T}_3 = \mathcal{T}_1 + \mathcal{T}_2,$$
$$\mathcal{T}_2 = \Re(\mathcal{Y}_1, \ldots, \mathcal{Y}_d), \quad \mathcal{T}_3 = \Re(\mathcal{X}_1, \ldots, \mathcal{X}_d),$$

$$\mathbf{X}_k(i_k) = \begin{pmatrix} \mathbf{Z}_k(i_k) & 0 \\ 0 & \mathbf{Y}_k(i_k) \end{pmatrix}, \quad \begin{array}{c} i_k = 1, \ldots, n_k, \\ k = 1, \ldots, d. \end{array}$$

- Multilinear products

$$\mathcal{T} = \Re(\mathcal{Z}_1, \ldots, \mathcal{Z}_d) \qquad c = \mathcal{T} \times_1 \mathbf{u}_1^T \times_2 \cdots \times_d \mathbf{u}_d^T$$

$$c = \Re(\mathbf{X}_1, \ldots, \mathbf{X}_d) \textit{ where } \mathbf{X}_k = \sum_{i_k=1}^{n_k} \mathbf{Z}_k(i_k) u_k(i_k).$$

- Hadamard product of tensors

$$\mathcal{T}_3 = \mathcal{T}_1 \circledast \mathcal{T}_2 = \Re(\mathcal{X}_1, \ldots, \mathcal{X}_d), \qquad \mathbf{X}_k(i_k) = \mathbf{Z}_k(i_k) \otimes \mathbf{Y}_k(i_k), \quad k = 1, \ldots, d.$$

- Inner product of two tensors
    - Apply Hadamard product followed by multilinear products with vectors of all ones.

Relation to Other Models

- CP decomposition is a special case of TR when cores are slice diagonal

$$\mathcal{T} = \sum_{\alpha=1}^{r} \mathbf{u}_\alpha^{(1)} \circ \cdots \circ \mathbf{u}_\alpha^{(d)},$$

$$\mathcal{T} = \Re(\mathcal{V}_1, \ldots, \mathcal{V}_d)$$
$$\mathbf{V}_k(i_k) \;=\; \mathrm{diag}(\mathbf{u}_{i_k}^{(k)})$$

- Tucker decomposition

$$\mathcal{T} = \mathcal{G} \times_1 \mathbf{U}^{(1)} \times_2 \cdots \times_d \mathbf{U}^{(d)}$$
$$\mathcal{G} \;=\; \Re(\mathcal{V}_1, \ldots, \mathcal{V}_d)$$

$$\mathcal{T} \;=\; \Re(\mathcal{Z}_1, \ldots, \mathcal{Z}_d)$$
$$\mathcal{Z}_k = \mathcal{V}_k \times_2 \mathbf{U}^{(k)},$$

- TT decomposition is a special case of TR when $\exists n, r_n = 1$

$$T(i_1, \ldots, i_d) = \mathrm{Tr}\left\{\mathbf{Z}_1(i_1)\mathbf{Z}_2(i_2)\cdots\mathbf{Z}_d(i_d)\right\}$$
$$= \sum_{\alpha_1=1}^{r_1} \mathbf{z}_1(\alpha_1, i_1, :)^T \mathbf{Z}_2(i_2)\cdots\mathbf{Z}_{d-1}(i_{d-1})\mathbf{z}_d(:, i_d, \alpha_1)$$

TR is a sum of TT representation

Data Structure Reconstruction

16×16 block

The first order represent a 2×2 pixel block.
The second order represent four above block.
…

16×16 block image to 4×4×4×4 block format:

1. **Reshape to 2×2×2×2×2×2×2×2.**
2. **Permute by {1,5,2,6,3,7,4,8}.**
3. **Reshape to 4×4×4×4.**

Figure 1: The effects of noise corrupted tensor cores. From left to right, each figure shows noise corruption by adding noise to one specific tensor core.

Learning efficient tensor representations with ring structure networks (ICLR Workshop 2018)

- Representation of original data or model parameters
- Tensorization is important and unexplored

(b)

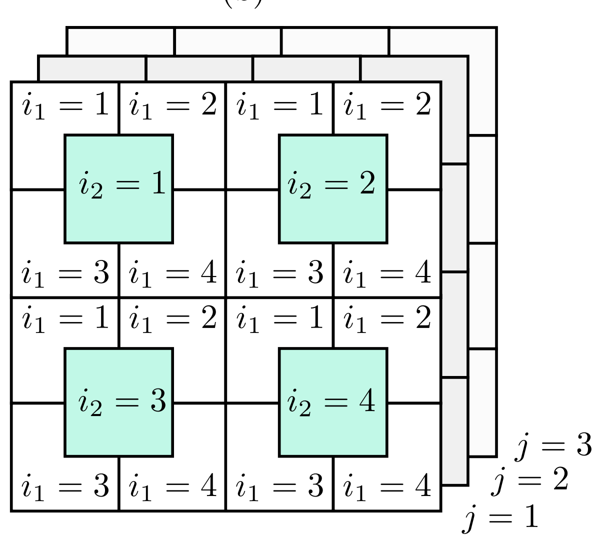


Table 4: Image representation by using tensorization and TR decomposition. The number of parameters is compared for SVD, TT and TR given the same approximation errors.

Data	$\epsilon = 0.1$		$\epsilon = 0.01$		$\epsilon = 9e-4$		$\epsilon = 2e-15$	
	SVD	TT/TR	SVD	TT/TR	SVD	TT/TR	SVD	TT/TR
$n = 256, d = 2$	9.7e3	9.7e3	7.2e4	7.2e4	1.2e5	1.2e5	1.3e5	1.3e5
Tensorization	$\epsilon = 0.1$		$\epsilon = 0.01$		$\epsilon = 2e-3$		$\epsilon = 1e-14$	
	TT	TR	TT	TR	TT	TR	TT	TR
$n = 16, d = 4$	5.1e3	3.8e3	6.8e4	6.4e4	1.0e5	7.3e4	1.3e5	7.4e4
$n = 4, d = 8$	4.8e3	4.3e3	7.8e4	7.8e4	1.1e5	9.8e4	1.3e5	1.0e5
$n = 2, d = 16$	7.4e3	7.4e3	1.0e5	1.0e5	1.5e5	1.5e5	1.7e5	1.7e5

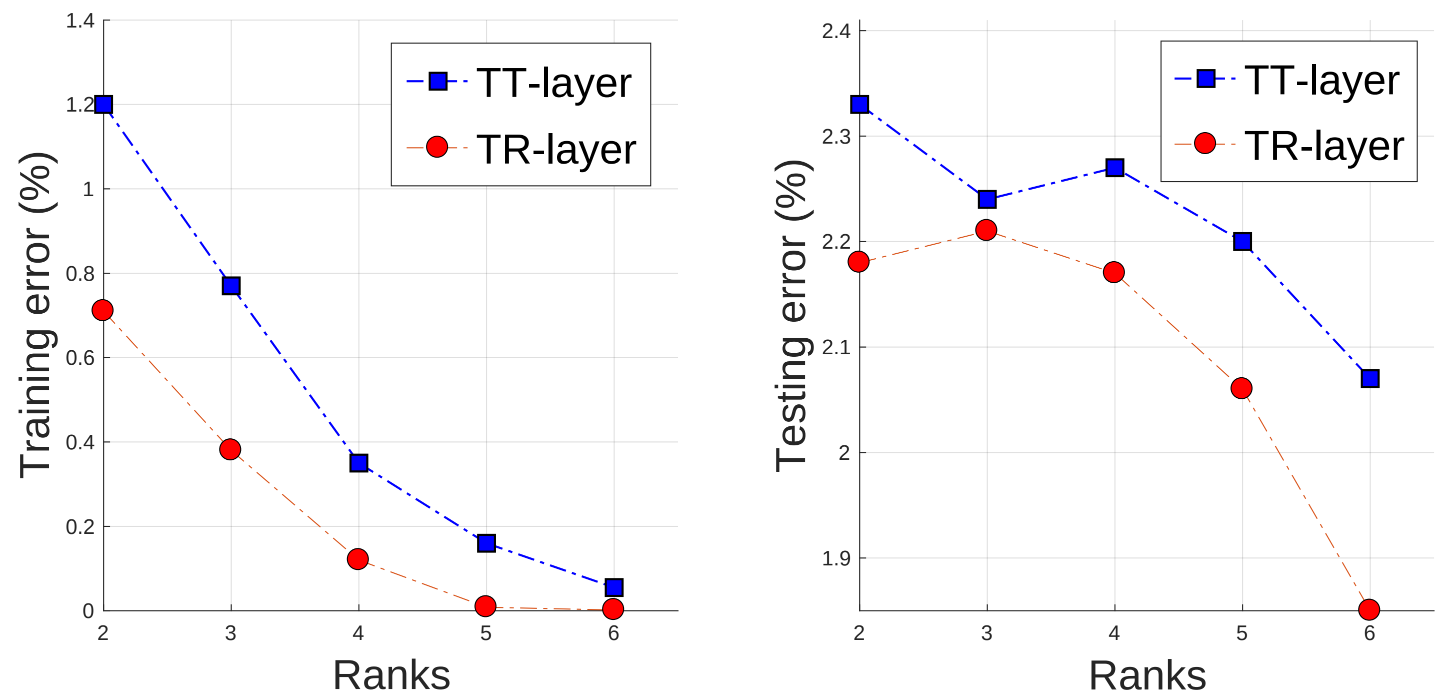


Figure 7: The classification performances of tensorizing neural networks by using TR representation.

Discussions

- What are the most important advantages of tensor methods?
- Which kind of tensor methods is promising in the future?