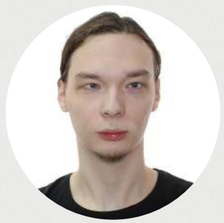
DEEP SCHOOL

лекция

Tensor and matrix methods

спикер



Konstantin Sozykin → Skoltech

Content

Content

Introduction → Matrix Decompositions →

Content

Content

Introduction → Matrix Decompositions → Tensor Decompositions → **Advanced Ideas** →

Content

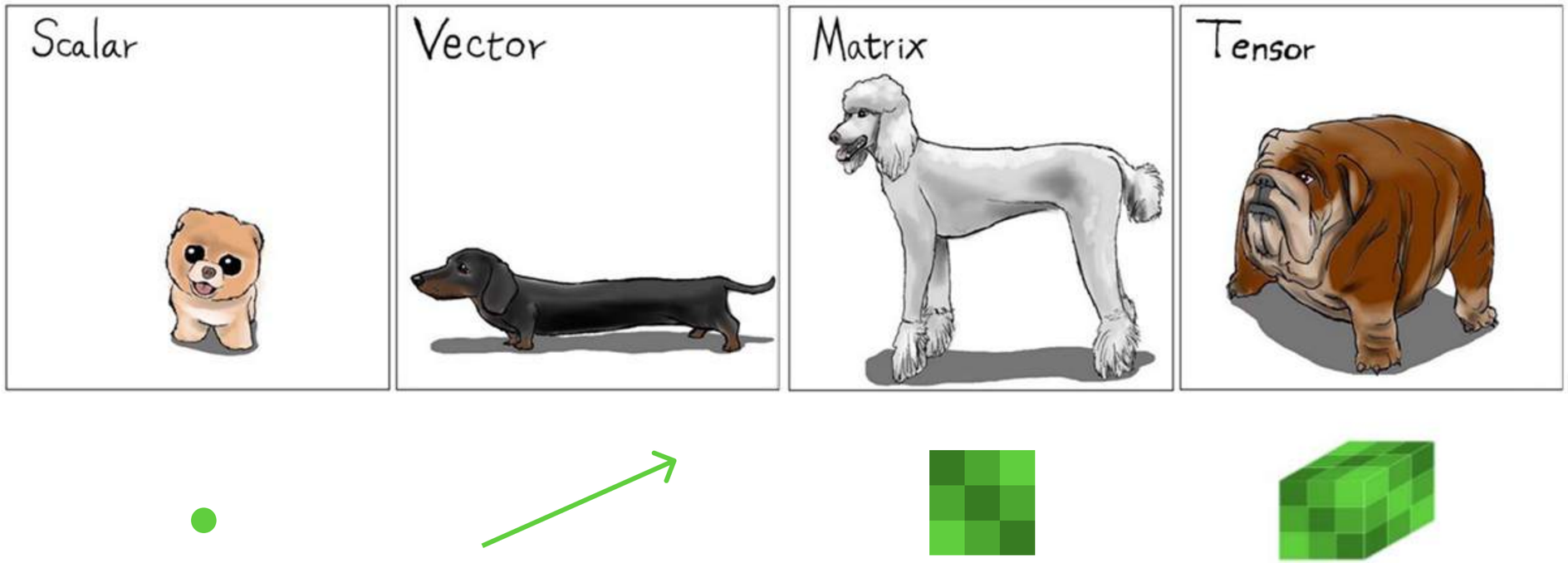
Introduction → Matrix Decompositions → Tensor Decompositions → Advanced Ideas → **Low-rank transformers** →

Content

Introduction

Data and tensors

- → A multidimensional array
- → A multilinear map



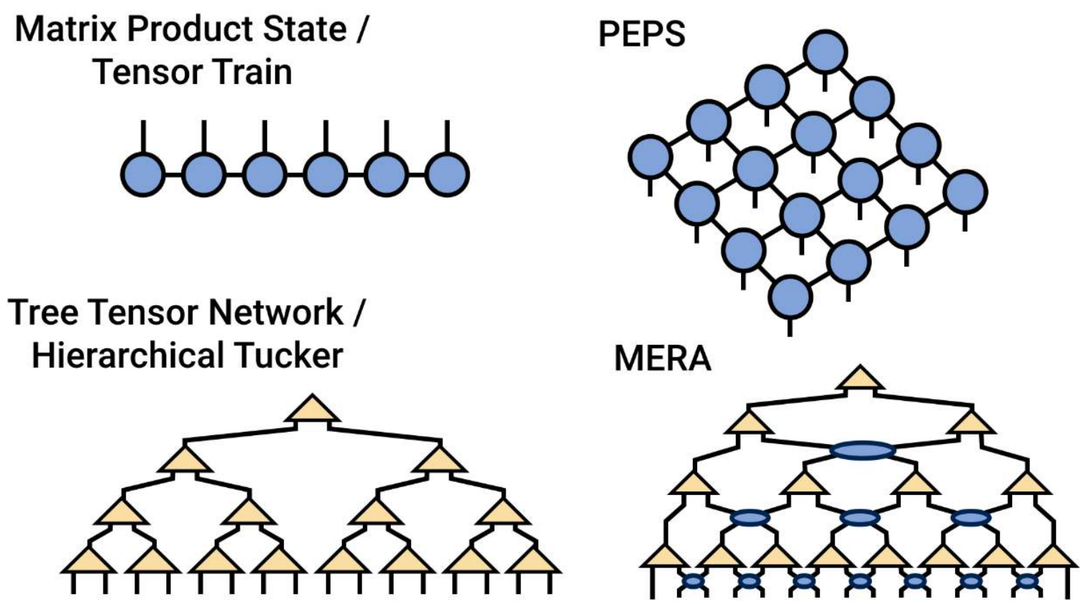
Neriman Tokcan ↗

Why tensor networks

Decompositions, factorizations

- → To handle big data and noisy data
- → To describe and represent some complex physical and biological systems
- → Initially it is more quantum physics and applied mathematics tool

Tensor Network ↗



Why tensor networks

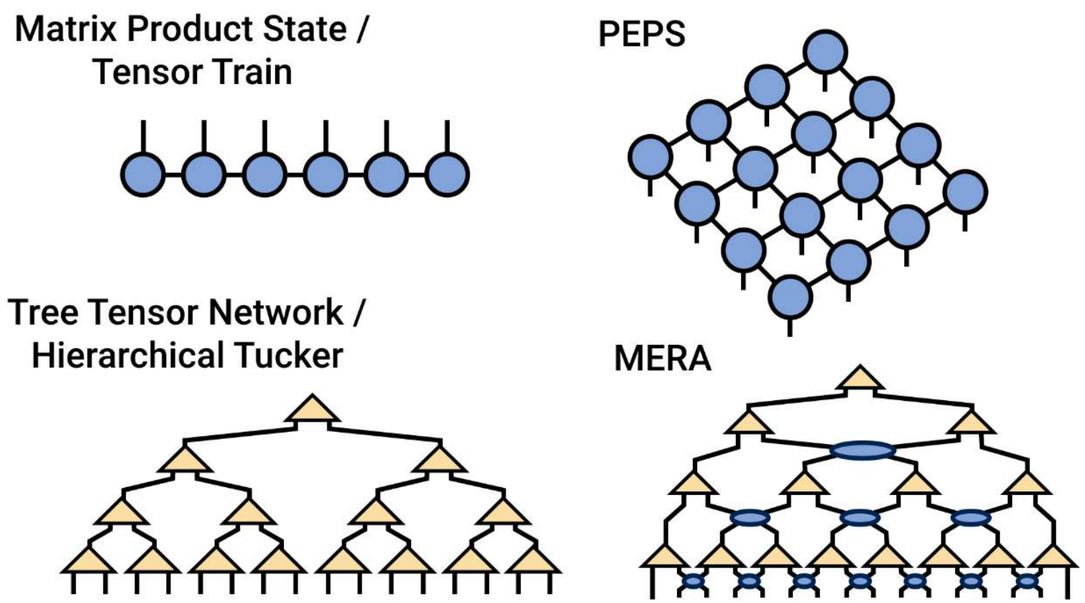
Decompositions, factorizations

→ To compress and restore (complete) data

→ **To reduce computational complexity (almost today topic)**

→ **To able to use an algebraic approach**

Tensor Network ↗



Matrix Decompositions

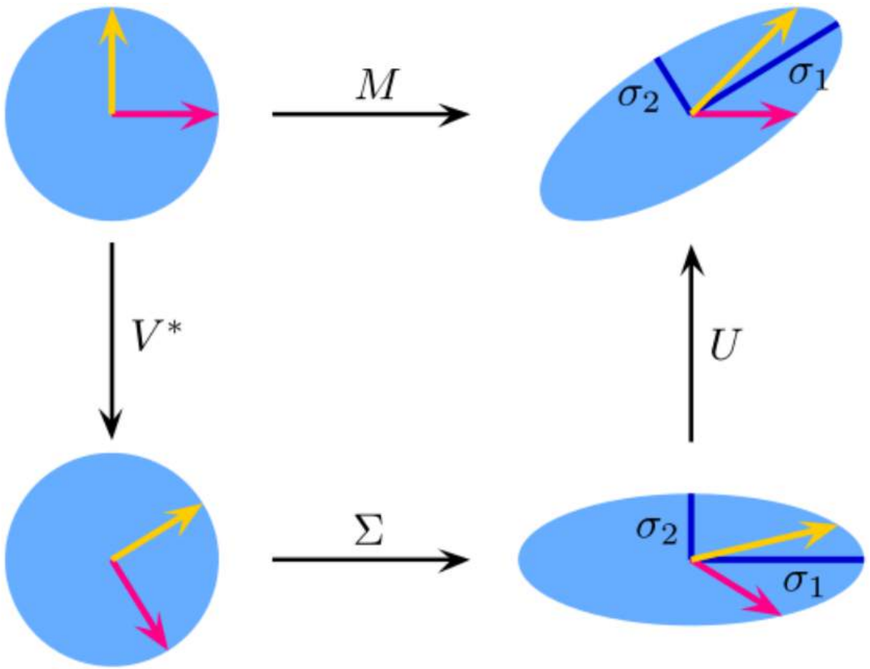
Why tensor networks

Decompositions, factorizations

Illustration of the singular value decomposition **UΣV*** of a real 2 × 2 matrix **M**.

- **Top:** The action of **M**, indicated by its effect on the unit disc *D* and the two canonical unit vectors e_1 and e_2
- **Left:** The action of **V***, a rotation, on *D*, e_1, and e_2.
- **Bottom:** The action of **Σ**, a scaling by the singular values σ_1 horizontally and σ_2 vertically.
- **Right:** The action of **U**, another rotation.

Singular value decomposition ↗

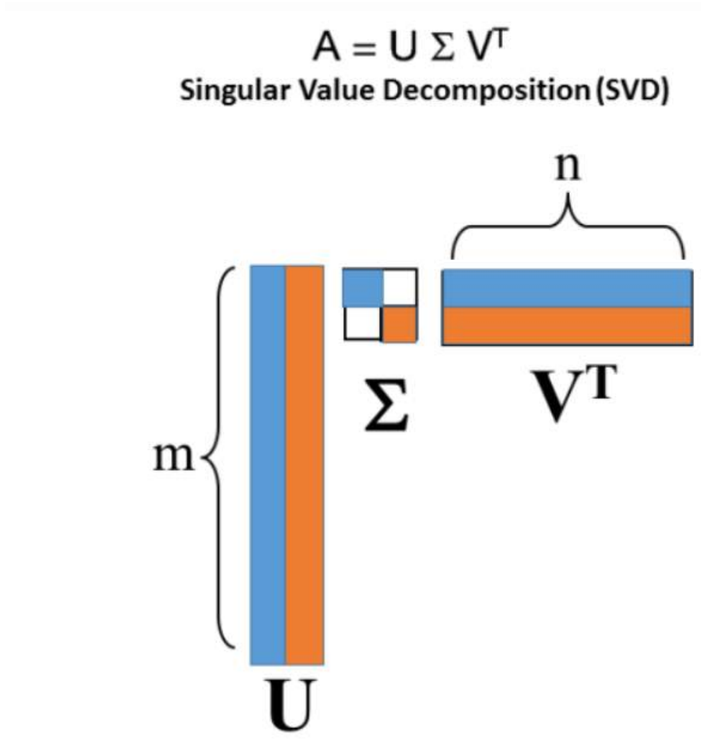


$$M = U \cdot \Sigma \cdot V^*$$

SVD and Linear Layer

- → Neural Networks overparameterized
- → Recover that Linear Layer is W @ x + b
- → then it can be expressed such as ~
- → **U @ [Σ @ V.T]) @ x + b**
- → instead m*n parameters we have only r * (1+m+n) or r * (m+n), where r singular values here
- → Then we **just replace and finetune the layer or whole network**
- → Be aware about transpose of matrix, when doing practical implementation

Singular Value Decomposition (SVD) ↗

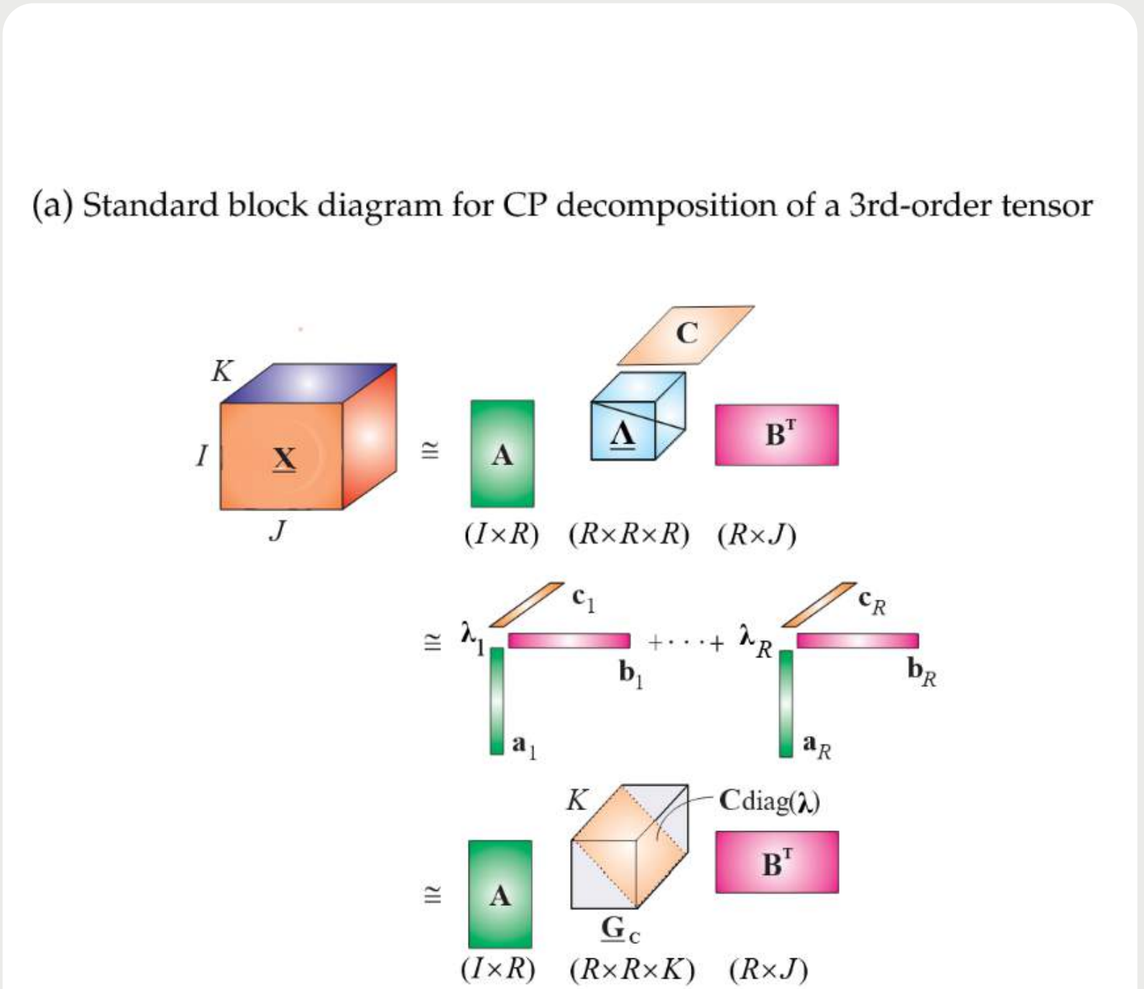


Tensor Decompositions

Canonical Polyadic Decomposition

- → CPD approximates given tensor **X** as a sum of R rank-one tensors (vectors)
- → Optimized with alternating least squares technique
- → Rank selection is NP hard!

Now publishers — Tensor Networks for Dimensionality Reduction and Large-scale Optimization: Part 1 Low-Rank Tensor Decompositions↗



Alternated Least Squares for CPD

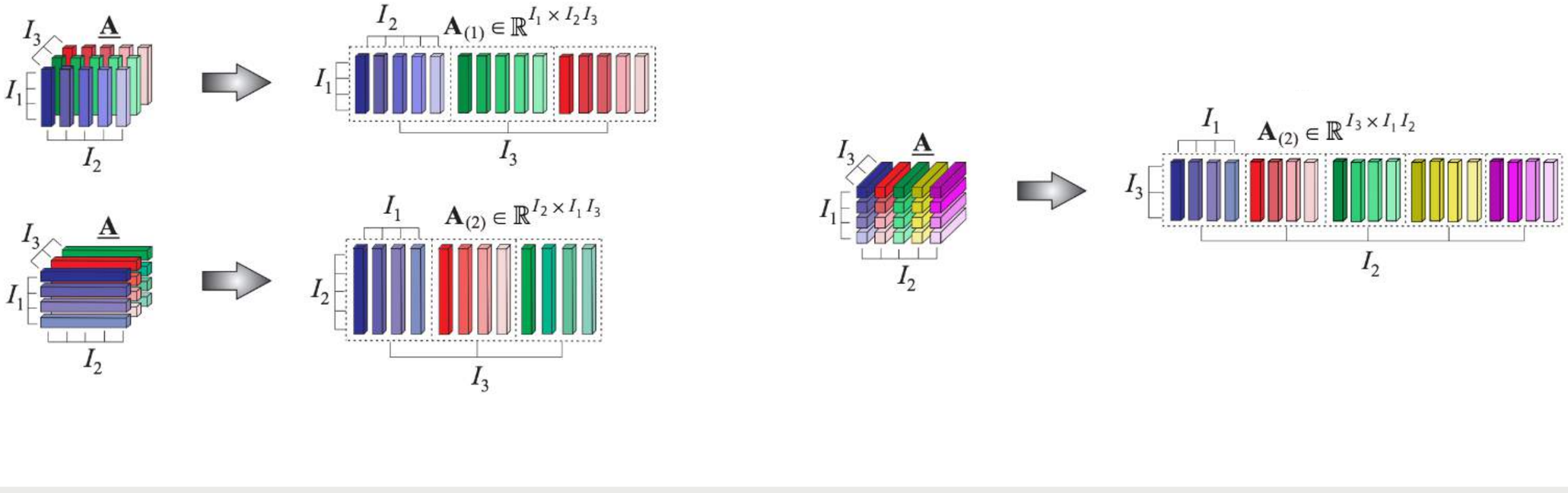
- Loss functions looks easy, but its non-convex
- The ALS approach fixes B and C to solve for A, then fixes A and C to solve for B then etc.

$$\min_{\hat{\mathcal{X}}} \|\mathcal{X} - \hat{\mathcal{X}}\| \quad \text{with} \quad \hat{\mathcal{X}} = \sum_{r=1}^{R} \lambda_r \, \mathbf{a}_r \circ \mathbf{b}_r \circ \mathbf{c}_r = [\![\boldsymbol{\lambda} \, ; \mathbf{A}, \mathbf{B}, \mathbf{C}]\!].$$

Tensor Decompositions and Applications ↗

Unfolding

→ Matricization **(flattening, unfolding)** used in tensor reshaping



Now publishers — Tensor Networks for Dimensionality Reduction and Large-scale Optimization: Part 1 Low-Rank Tensor Decompositions↗

Additional operations

- → Khatri-Rao — blockwise
- → Hadamard — elementwise
- → Pseudo-inverse matrix

- Khatri-Rao product $\mathbf{A} \odot \mathbf{B}$ is the columns-wise Kronecker product. Setting $\mathbf{A} = [\mathbf{a}_1, \ldots, \mathbf{a}_{J_1}]$ and $\mathbf{B} = [\mathbf{b}_1, \ldots, \mathbf{b}_{J_1}]$,

$$\mathbf{A} \odot \mathbf{B} = [\mathbf{a}_1 \boxtimes \mathbf{b}_1, \ldots, \mathbf{a}_{J_1} \boxtimes \mathbf{b}_{J_1}].$$

- Eample $\mathbf{A} = \begin{bmatrix} a & b & c \\ d & e & f \end{bmatrix}, \mathbf{B} = \begin{bmatrix} g & h & i \\ j & k & l \end{bmatrix}$.

$$\begin{aligned} \mathbf{A} \odot \mathbf{B} &= \begin{bmatrix} \mathbf{a}_1 \boxtimes \mathbf{b}_1 & \mathbf{a}_2 \boxtimes \mathbf{b}_2 & \mathbf{a}_3 \boxtimes \mathbf{b}_3 \end{bmatrix} \\ &= \begin{bmatrix} \begin{bmatrix} a \\ d \end{bmatrix} \boxtimes \begin{bmatrix} g \\ j \end{bmatrix} & \begin{bmatrix} b \\ e \end{bmatrix} \boxtimes \begin{bmatrix} h \\ k \end{bmatrix} & \begin{bmatrix} c \\ f \end{bmatrix} \boxtimes \begin{bmatrix} i \\ l \end{bmatrix} \end{bmatrix} \\ &= \begin{bmatrix} ag & bh & ci \\ aj & bk & cl \\ dg & eh & fi \\ dj & ek & fl \end{bmatrix}. \end{aligned}$$

"⊙" denotes the Khatri-Rao product.

"⊛" represents the element-wise Hadamard product.

Now publishers — Tensor Networks for Dimensionality Reduction and Large-scale Optimization: Part 1 Low-Rank Tensor Decompositions ↗

Alternated Least Squares for CPD

- → Pseudocode of basic ALS
- → X : I x J x K
- → X_(1) : I x JK
- → B khatrio C : JK x R
- → A^T A : R x R
- → A update : (I x JK) x (JK x R) x (R x R)

Now publishers — Tensor Networks for Dimensionality Reduction and Large-scale Optimization: Part 1 Low-Rank Tensor Decompositions ↗

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 $\boldsymbol{\lambda}$.

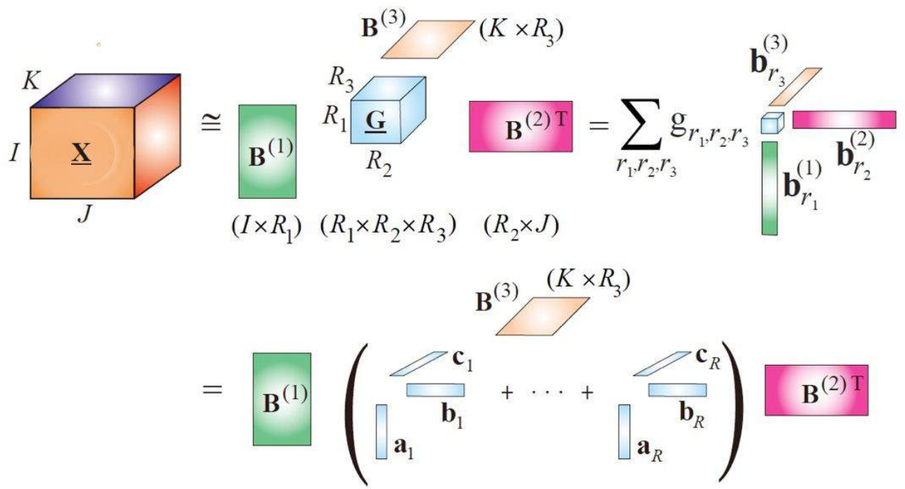
Tucker decomposition

→ **X** decomposed as a set of matrices (2D tensors) and one small core tensor (small 3D tensor)

A New Truncation Strategy for the Higher-Order Singular Value Decomposition | SIAM Journal on Scientific Computing ↗

Now publishers — Tensor Networks for Dimensionality Reduction and Large-scale Optimization: Part 1 Low-Rank Tensor Decompositions ↗

(a) Standard block diagrams of Tucker (top) and Tucker-CP (bottom) decompositions for a 3rd-order tensor



Tucker decomposition

→ Theorem by De Lathauwer, De Moor and Vandewalle proved in following↗ paper

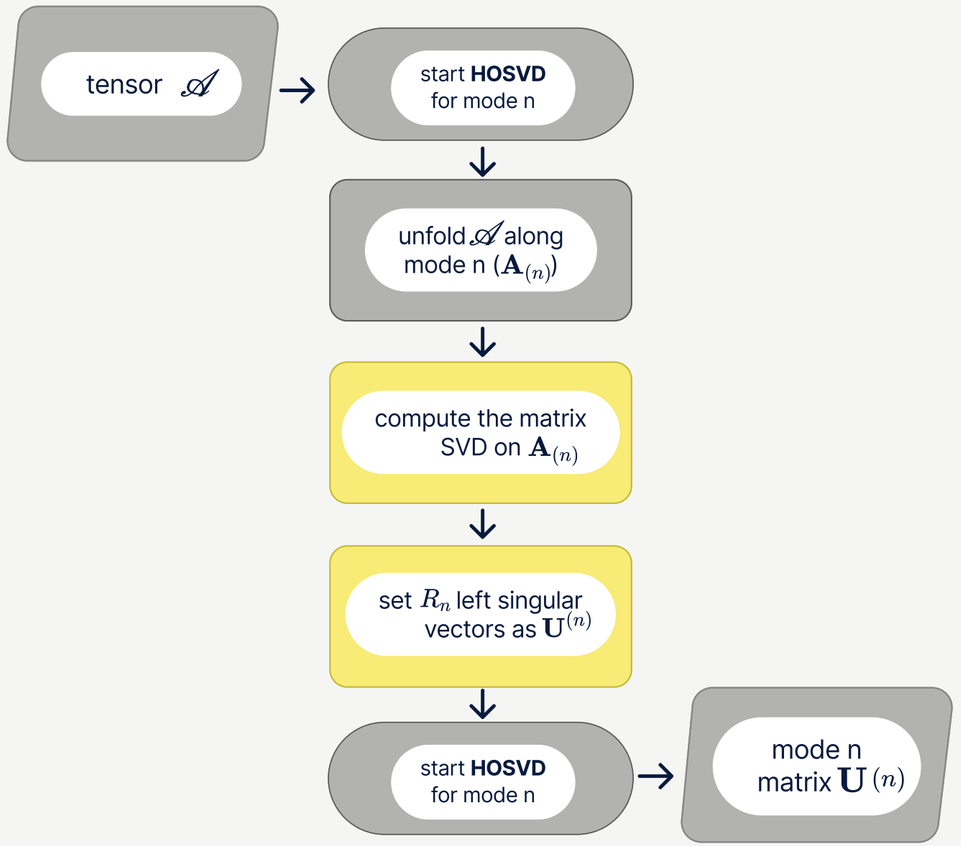
→ SVD on every mode's tensor unfolding $\mathbf{A}_{(n)}$

→ Set basis factor matrices $\mathbf{U}^{(n)}$ as R_n leading left singular vectors of $\mathbf{A}_{(n)}$

→ Derive core $\mathscr{B}$ from original data and inverse factor matrices

→ Defines a Tucker model with $\mathscr{B}\,\mathbf{U}^{(n)}$

$$\mathscr{B} = \mathscr{A} \times_1 \mathbf{U}^{(1)^{-1}} \times_2 \mathbf{U}^{(2)^{-1}} \times_3 \mathbf{U}^{(3)^{-1}}$$



Tucker decomposition

- → HOSVD is not the only solution, for instance see STHOSVD method ↗
- → **Sequential reduce original tensor**
- → No inverse matrices

Now publishers — Tensor Networks for Dimensionality Reduction and Large-scale Optimization: Part 1 Low-Rank Tensor Decompositions ↗

Algorithm 4 Sequentially Truncated HOSVD (STHOSVD) Algorithm

Input : A data tensor $\underline{\mathbf{X}} \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_N}$ and a tolerance ϵ;

Output: Approximative HOSVD of the tensor $\underline{\mathbf{X}}$ $\underline{\mathbf{X}} \cong [\![\underline{\mathbf{S}}; \mathbf{Q}^{(1)}, \mathbf{Q}^{(2)}, \ldots, \mathbf{Q}^{(N)}]\!]$ and multilinear rank $(R_1, R_2, \ldots, R_N)$

1 Set $\underline{\mathbf{S}} = \underline{\mathbf{X}}$
2 **for** $n = 1, 2, \ldots, N$ **do**
3 $\quad [\mathbf{Q}^{(n)}, \sim, \sim] = \text{truncated-svd}\left(\mathbf{S}_{(n)}, \frac{\epsilon}{N}\right)$
4 $\quad \underline{\mathbf{S}} = \underline{\mathbf{S}} \times_n \mathbf{Q}^{(n)\,T}$
5 **end**

Connections

→ There is analogy/connection between matrix and tensor decompositions

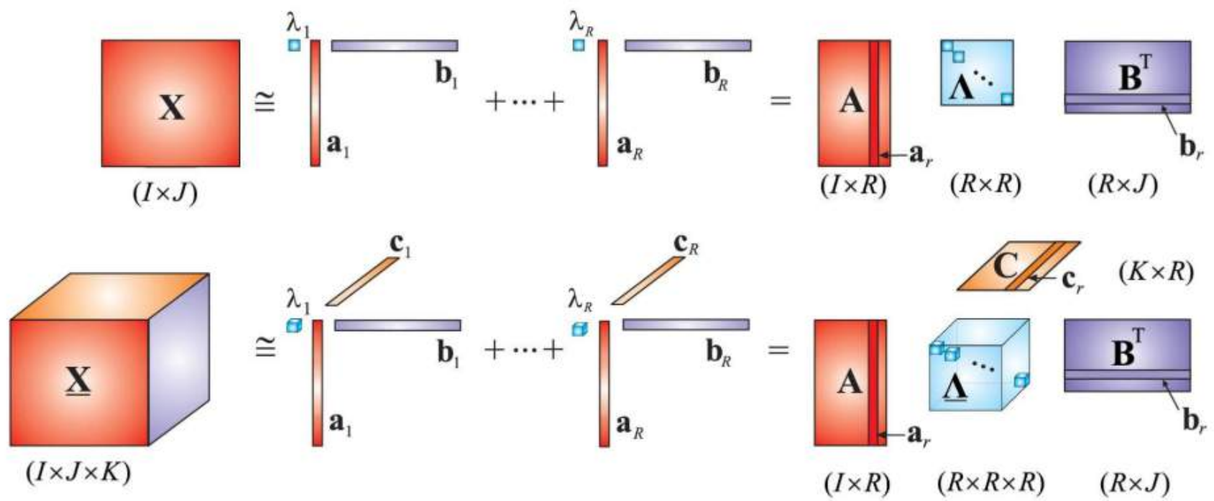


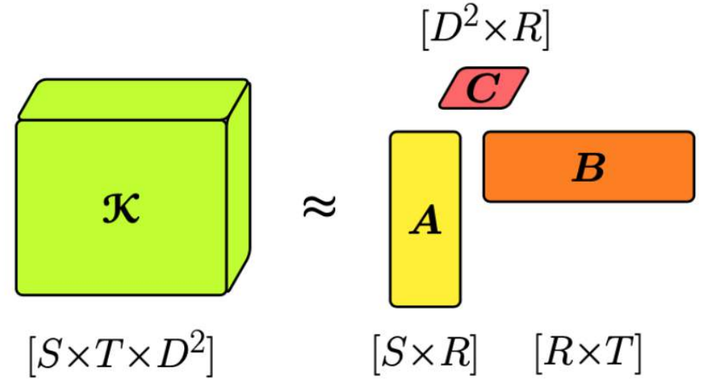
Figure 3.2: Analogy between a low-rank matrix factorization, $\mathbf{X} \cong \mathbf{A}\mathbf{\Lambda}\mathbf{B}^{\mathrm{T}} = \sum_{r=1}^{R} \lambda_r \ \mathbf{a}_r \circ \mathbf{b}_r$ (top), and a simple low-rank tensor factorization (CP decomposition), $\underline{\mathbf{X}} \cong \underline{\mathbf{\Lambda}} \times_1 \mathbf{A} \times_2 \mathbf{B} \times_3 \mathbf{C} = \sum_{r=1}^{R} \lambda_r \ \mathbf{a}_r \circ \mathbf{b}_r \circ \mathbf{c}_r$ (bottom).

Now publishers — Tensor Networks for Dimensionality Reduction and Large-scale Optimization: Part 1 Low-Rank Tensor Decompositions↗

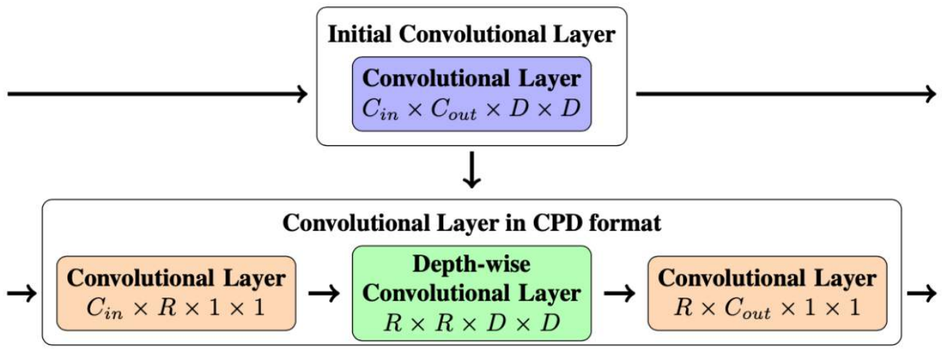
CPD as a depthwise convolution

→ CPD allows to reduce number of parameters from S x T x D^2 x D to (S + T + D) x R.

→ Applying ordinary CPD to convolutional kernel leads to instability (will cover later) for $\hat{\mathcal{K}} = [\![\mathbf{A}, \mathbf{B}, \mathbf{C}]\!]$ the neural network



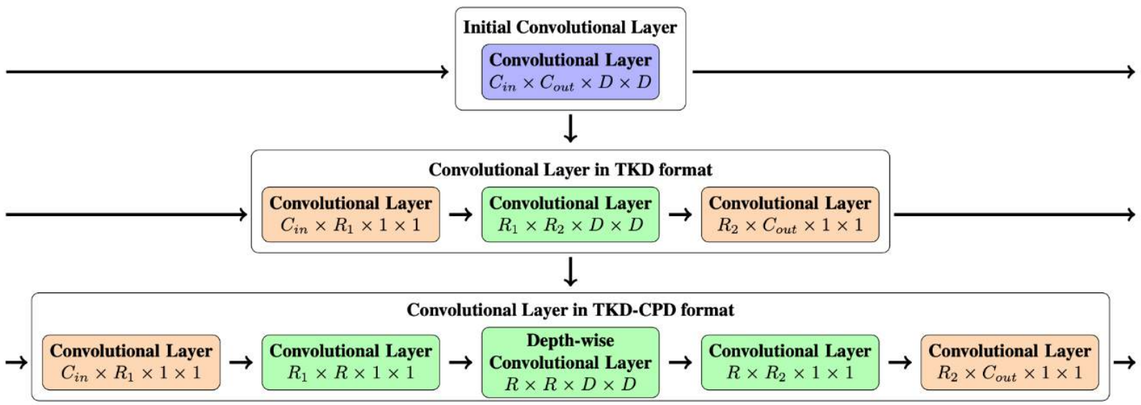
CPD scheme: $\hat{\mathcal{K}} = [\![\mathbf{A}, \mathbf{B}, \mathbf{C}]\!]$



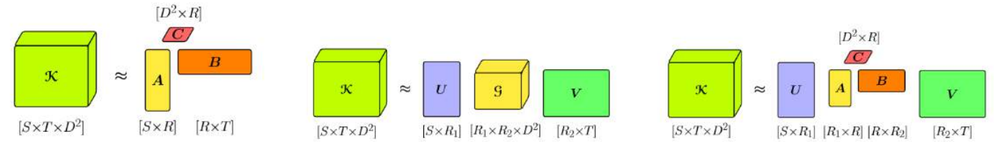
CP-decomposed Convolutional Layer

CPD/Tucker and Convolutions

→ Tensor Networks can be combined to achieve better performance



Stable Tensor Deomposition for Compression of Cnn



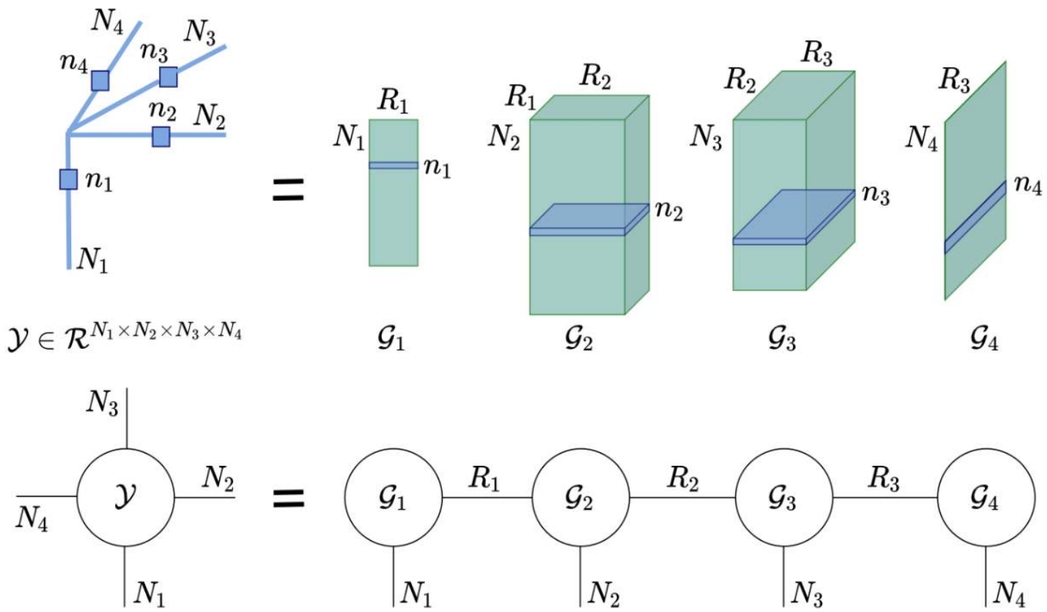
(a) CPD (b) TKD (c) TKD-CPD

Fig. 1: Approximation of a third-order tensor using Canonical Polyadic tensor decomposition (CPD), Tucker-2 tensor decomposition (TKD), and their combination (TKD-CPD). CPD and TKD are common methods applied for CNN compression.

Tensor Train

→ **Tensor Train (TT)** is a contraction (linear chain) of 3d cores/factors (except first and last which are 2D)

$$\widetilde{\mathbf{W}}^{\mathrm{TT}}_{c,y,x,t} = \sum_{r_1=1}^{R_1} \sum_{r_2=1}^{R_2} \sum_{r_3=1}^{R_3} C_{c,r_1} \mathbf{Y}_{r_1,y,r_2} \mathbf{X}_{r_2,x,r_3} T_{r_3,t}.$$



How Informative is the Approximation Error from Tensor Decomposition for Neural Network Compression? | OpenReview↗

Comparison of TT vs CPD and Tucker

→ Notice common index in every factor

$$\widetilde{\mathbf{W}}^{\mathrm{CP}}_{c,y,x,t} = \sum_{r=1}^{R} \boldsymbol{C}_{c,r} \boldsymbol{Y}_{y,r} \boldsymbol{X}_{x,r} \boldsymbol{T}_{t,r}.$$

$$\widetilde{\mathbf{W}}^{\mathrm{Tucker}}_{c,y,x,t} = \sum_{r_1=1}^{R_1} \sum_{r_2=1}^{R_2} \sum_{r_3=1}^{R_3} \sum_{r_4=1}^{R_4} \mathbf{G}_{r_1,r_2,r_3,r_4} \boldsymbol{C}_{c,r_1} \boldsymbol{Y}_{y,r_2} \boldsymbol{X}_{x,r_3} \boldsymbol{T}_{t,r_4}$$

$$\widetilde{\mathbf{W}}^{\mathrm{TT}}_{c,y,x,t} = \sum_{r_1=1}^{R_1} \sum_{r_2=1}^{R_2} \sum_{r_3=1}^{R_3} \boldsymbol{C}_{c,r_1} \mathbf{Y}_{r_1,y,r_2} \mathbf{X}_{r_2,x,r_3} \boldsymbol{T}_{r_3,t}.$$

How Informative is the Approximation Error from Tensor Decomposition for Neural Network Compression? | OpenReview↗

Tensor Network Libraries

Teneva documentation ↗

1	More than 80 packages on tensor networks available
2	See THE LANDSCAPE OF SOFTWARE FOR TENSOR COMPUTATIONS ↗
3	For ML and optimization notable packages are TensorLy TTPY, Teneva etc

Practical and Theoretical Questions Teneva documentation ↗

?	Which decomposition to choose
?	How to optimize/decompose given full tensor
?	How to select ranks
?	Greedy approach — expensive
?	AutoML/Hyperparameter Optimization approach — Optuna, RAY etc. less expensive, but harder to implement
?	How approximation error related to final accuracy
?	Layer-by-layer or a single shot?

Advanced Ideas

MUSCO

Multi-Stage Compression of neural networks

→ Iteratively we do:

1. Compress with **gradually decreased ranks**, using past approximations
2. finetune

→ automatic rank selection with **bayesian optimization**

→ opensource — you can try to play

Click to read full PhD Thesis by E. Ponomarev ↗

MUSCO: MUlti-Stage COmpression of neural networks ↗

Чтобы сжать слой с весовым тензором θ при заданном ранге R первый раз, решается следующая задача минимизации нормы Фробениуса между исходным тензором θ и его факторизованным приближением $\hat{\theta}^R$:

$$\min_{\theta_1^R,\ldots,\theta_N^R} ||\theta - \hat{\theta}^R||, \quad \text{такие, что} \\ \mathcal{F}_{\text{fact}}(\hat{\theta}^R) = (\theta_1^R, \ldots, \theta_N^R), \tag{1.6}$$

где $\theta_1^R, \ldots, \theta_N^R$ обозначают компоненты тензора в факторизованной форме с рангом R и числом факторов N.

Так как предложенный алгоритм сжатия MUSCO заведомо итеративный, то для второго и следующих сжатий слоя производится операция пересчета факторов $\{\theta_n^R\}_{n=1}^N$, полученных на предыдущем шаге метода. А именно, при заданном новом ранге $R' : R' < R$ решается следующая задача минимизации:

$$\min_{\theta_1^{R'},\ldots,\theta_N^{R'}} ||\mathcal{F}_{\text{full}}(\theta_{\text{fact}}) - \hat{\theta}^{R'}||, \quad \text{такие, что} \\ \theta_{\text{fact}} = (\theta_1^R, \ldots, \theta_N^R), \\ \mathcal{F}_{\text{fact}}(\hat{\theta}^{R'}) = (\theta_1^{R'}, \ldots, \theta_N^{R'}), \tag{1.7}$$

где $\theta_1^{R'}, \ldots, \theta_N^{R'}$ обозначают обновленные факторы исходного тензора весов выбранного слоя нейросетевой модели.

MUSCO

Multi-Stage Compression of neural networks

Model	FLOPs	mAP
FASTER R-CNN (VGG-16) @ VOC2007		
[7] baseline	1×	68.7
Channel Prunning [7]	4×	66.9(-1.8)
Accelerating VD [23]	4×	67.8(-0.9)
AutoML Compression [6]	4×	68.8(+0.1)
Used baseline	1.0×	71.1
Tucker2-iter (nx, 3.16)	3.16×	70.7(-0.4)
MUSCO(nx, 1.77, 2)	**3.72×**	**70.4(-0.7)**
MUSCO(nx, 2, 2)	**4.69×**	**70.1(-1.0)**
Tucker2-iter (nx, 10)	9.67×	68.6(-2.5)
MUSCO(nx, 3.16, 2)	**10.49×**	**69.2(-1.9)**
MUSCO(nx, 1.77, 4)	**13.95×**	**68.3(-2.8)**

Table 1: Comparison of Faster R-CNN (with VGG-16 backbone) compressed models on VOC2007 evaluation dataset. MUSCO(nx, 3.16, 2) is a compressed model obtained after two compression steps using 3.16× parameter reduction at each step

Алгоритм 1 Алгоритм итеративной аппроксимации малого ранга для автоматического сжатия сети

Input: Предварительно обученная исходная модель, M

Output: Тонко настроенная сжатая модель, M^*.

$M^* \leftarrow M$

while желаемая степень сжатия не достигнута или автоматически выбранные ранги не стабилизировались **do**

  $R \leftarrow$ автоматически выбранные ранги для низкоранговых тензорных аппроксимаций сверточных и полносвязных весовых тензоров.

  $M \leftarrow$ (дополнительно) сжатая модель, полученная из M заменой весов слоев их тензорными аппроксимациями ранга R.

  $M^* \leftarrow$ тонко настроенная модель M.

end while

MUSCO: MUlti-Stage COmpression of neural networks ↗

Solution Sensitivity

Sensitivity of the tensor $\mathcal{T} = [\![\mathbf{A}, \mathbf{B}, \mathbf{C}]\!]$ of size $I \times J \times K$ is a measure of factorized tensor norm change with respect to perturbations in individual factor matrices

$$\mathrm{ss}([\![\mathbf{A}, \mathbf{B}, \mathbf{C}]\!]) = \lim_{\sigma^2 \to 0} \frac{1}{R\sigma^2} E\{\|\mathcal{T} - [\![\mathbf{A} + \delta\mathbf{A}, \mathbf{B} + \delta\mathbf{B}, \mathbf{C} + \delta\mathbf{C}]\!]\|_F^2\}$$

where $\delta\mathbf{A}, \delta\mathbf{B}, \delta\mathbf{C}$ have random i.i.d. elements from $N(0, \sigma^2)$.

Sensitivity can be computed as

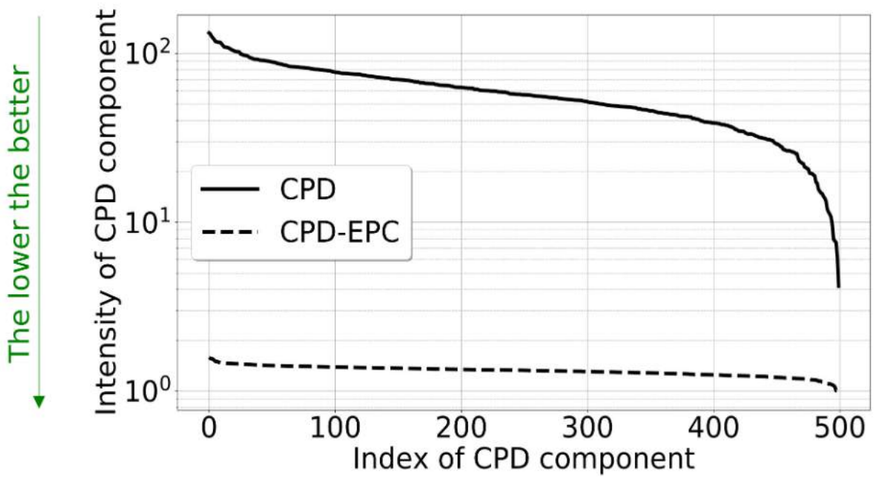
$$\mathrm{ss}([\![\mathbf{A}, \mathbf{B}, \mathbf{C}]\!]) = K \operatorname{tr}\{(\mathbf{A}^T\mathbf{A}) \circledast (\mathbf{B}^T\mathbf{B})\} + I \operatorname{tr}\{(\mathbf{B}^T\mathbf{B}) \circledast (\mathbf{C}^T\mathbf{C})\} + J \operatorname{tr}\{(\mathbf{A}^T\mathbf{A}) \circledast (\mathbf{C}^T\mathbf{C})\}$$

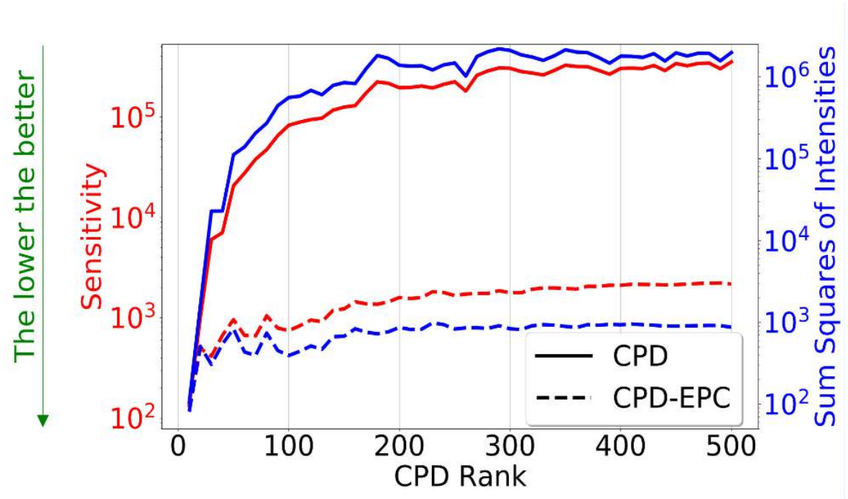
Solution CPD-EPC

CPD
with minimal
sensitivity:

$$\min_{\{\mathbf{A},\mathbf{B},\mathbf{C}\}} \quad \mathtt{ss}(\llbracket \mathbf{A}, \mathbf{B}, \mathbf{C} \rrbracket)$$
$$\text{s.t.} \quad \|\mathcal{K} - \llbracket \mathbf{A}, \mathbf{B}, \mathbf{C} \rrbracket\|_F^2 \leq \delta^2$$

Bound δ^2 can be the approximation error of the CPD with diverging components.





Stable CPD

- → No python source code available unfortunately due to industrial copyright
- → There is matlab prototype for Stable CPD

GitHub — phananhhuy/TensorBox: Matlab package for Tensor decompositions↗

Table 1: Comparison of different model compression methods on ILSVRC-12 validation dataset. The baseline models are taken from Torchvision.

Model	Method	↓ FLOPs	Δ top-1	Δ top-5
VGG-16	Asym. [57]	≈ 5.00	-	-1.00
	TKD+VBMF [26]	4.93	-	-0.50
	Our (EPS[1]=0.005)	**5.26**	**-0.92**	**-0.34**
ResNet-18	Channel Gating NN [24]	1.61	-1.62	-1.03
	Discrimination-aware Channel Pruning [58]	1.89	-2.29	-1.38
	FBS [13]	1.98	-2.54	-1.46
	MUSCO [14]	2.42	-0.47	-0.30
	Our (EPS[1]=0.00325)	**3.09**	**-0.69**	**-0.15**
ResNet-50	**Our** (EPS[1]=0.0028)	**2.64**	**-1.47**	**-0.71**

[1] EPS: accuracy drop threshold. Rank of the decomposition is chosen to maintain the drop in accuracy lower than EPS.

Data and tensors

→ Inference time on different hardware

Table 2: Inference time and acceleration for ResNet-50 on different platforms.

Platform	Model inference time	
	Original	Compressed
Intel® Xeon®Silver 4114 CPU 2.20 GHz	3.92 ± 0.02 s	2.84 ± 0.02 s
NVIDIA®Tesla®V100	102.3 ± 0.5 ms	89.5 ± 0.2 ms
Qualcomm®Snapdragon™845	221 ± 4 ms	171 ± 4 ms

Low-rank transformers

Stable CPD

- → Nice Paper from Huawei
- → Many other papers but it might hard-to-follow for beginners
- → kinda MUSCO + KD + some tricks

Strategies for Applying Low Rank Decomposition to Transformer-Based Models ↗

In progressive LRD, we use a rank selection approach in which, we start from a lower compression ratio and apply low rank decomposition to the entire model altogether (just like the single-shot method). The model is then fine-tuned for some epochs in order to recover the accuracy. The rank R is decreased incrementally in the next steps so that we have a higher compression ratio. This process is repeated until the desired compression ratio is achieved for the model. Starting from the second compression step, we only decompose the second decomposed layer of previously decomposed layers and multiply them so that it results in two decomposed layers again. This way, the number of layers will not increase again in the next stages of decomposition as we use matrix multiplication for merging the newly generated weight matrices in order to keep the number of layers the same as the first stage of LRD. This process is shown in Fig. 1.

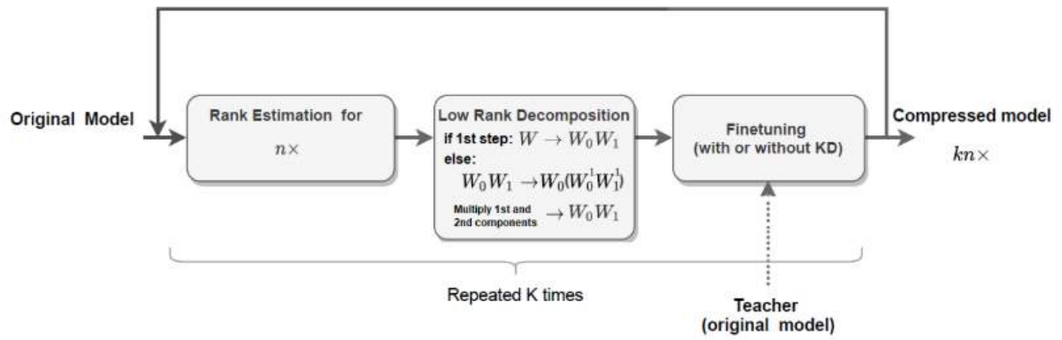


Figure 1: Progressive decomposition scheme

Progressive low-rank compression for VITå

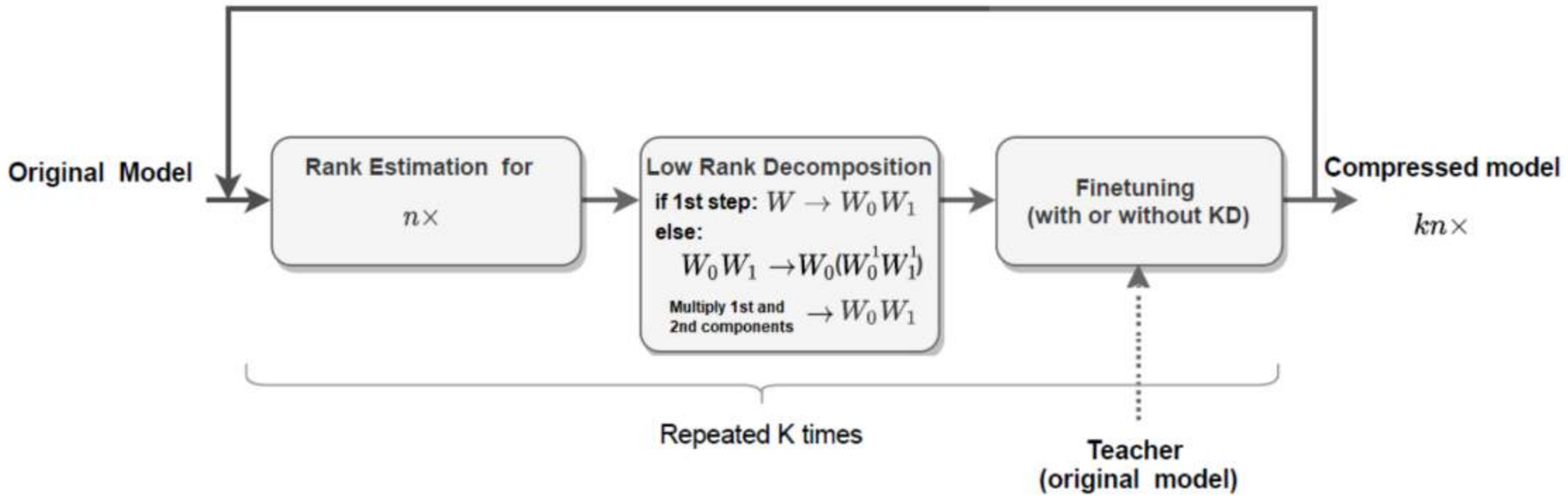


Figure 1: Progressive decomposition scheme

Strategies for Applying Low Rank Decomposition to Transformer-Based Models ↗

Progressive low-rank compression for VITå

Table 1: Experimental results for ViT model decomposed LRD using different strategies including single-shot, layer-by-layer and progressive LRD. The model is pretrained on ImageNet-21k and finetuned on CIFAR-10 and CIFAR-100 datasets.

Method	# LRD Steps	Comp Ratio	CIFAR-10 Top-1 (%)	CIFAR-100 Top-1 (%)
ViT (Org)	0	1×	98.76	92.95
Single-Shot	1	2.63×	94.53	87.09
Layer-by-Layer	4	2.63×	95.36	87.51
Progressive	4	2.63×	96.01	88.62

Strategies for Applying Low Rank Decomposition to Transformer-Based Models ↗

Summary

Tensor methods provides:

An elegant tool to reduce overparameterization of neural networks

Homework

Using provided template and previous homework:

- Implement SVD layer 1
- Replace linear layers in decoder head 2
- Compare it within different settings 3

Links and References

Further readings

1. Tensor Networks for Dimensionality Reduction and Large-scale Optimization: Part 1 ↗
2. Tensor Networks for Dimensionality Reduction and Large-scale Optimization: Part 2 ↗
3. Stable Low-rank Tensor Decomposition for Compression of Convolutional Neural Network ↗
4. MUSCO: Multi-Stage Compression of neural networks ↗
5. Quantization Aware Factorization for Deep Neural Network Compression ↗
6. How Informative is the Approximation Error from Tensor Decomposition for Neural Network Compression? | OpenReview ↗