LECTURE 3: TENSOR DECOMPOSITIONS

IVAN OSELEDETS

PLAN OF THE COURSE

Lecture 1:

Basic machine learning models. Supervised/unsupervised learning.Deep learning. Convolutional neural networks.

Lecture 2:

Modern deep learning models. Concept of attention. Transformers (natural language processing / vision transformers). Idea of generative models (GANs). Application to image processing.

Lecture 3:

Tensor decompositions: Basic tensor factorizations (canonical polyadic, Tucker, tensor-train, H-Tucker). Algorithms for computing tensor factorizations. Applications of tensor decompositions, including image processing

Lecture 4:

Multivariate function approximation. Cross approximation. Approximation of smooth functions.

Lecture 5: Tensor decompositions and machine learning for compression of signals, images and videos. Neural Radiance Fields, signed distance functions.

RECAP OF LECTURE 2

- Convolutional neural networks
- Concept of attention
- Transformer architecture
- Transformers in NLP
- Transformers in Vision (Vision Transformers)
- Idea of GAN, unsupervised learning.

PLAN OF LECTURE 3

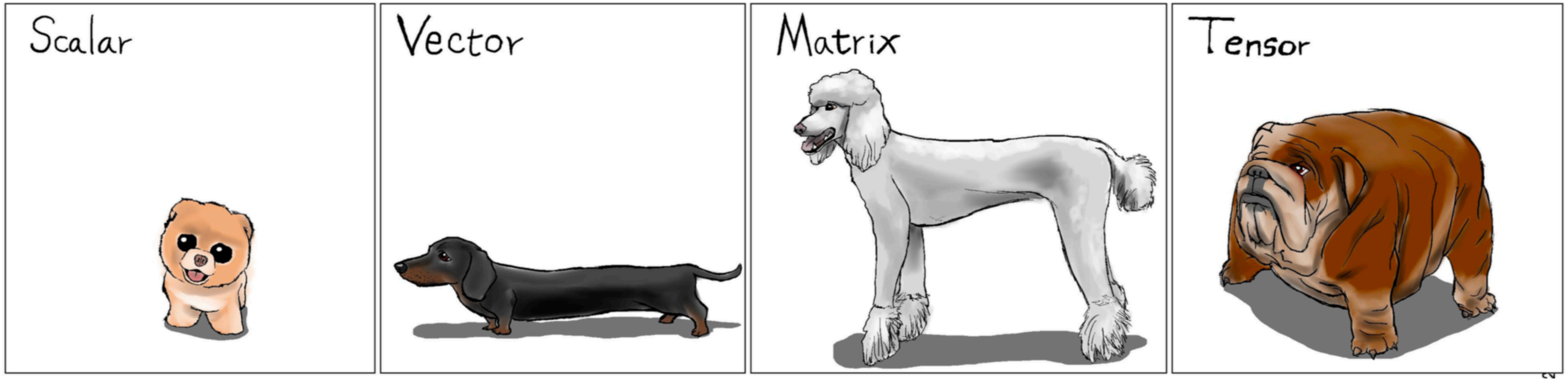
- Tensors
- Basic tensor decompositions
- Advanced tensor decompositions.

WHAT IS A TENSOR

D = 1: Vector

D = 2: Matrix

D > 2: Tensor



WHY STUDY TENSORS?

Data is multidimensional.

Can you come up with examples of multidimensional data in practice?

WHY STUDY TENSORS?

Data is multidimensional.

Can you come up with examples of multidimensional data in practice?

Images: width x height x colors (i.e., 512 x 512 x 3)

Videos: width x height x colors x time (i.e. 512 x 512 x 3 x 100)

Multispectral images

Tomography/MRI images

Many, many more..

LITERATURE ON TENSORS

1) Brett Bader, Tammy Kolda, Tensor decomposition, SIREV, 2009

2) Cichocki A, Lee N, Oseledets I, Phan AH, Zhao Q, Mandic DP. Tensor networks for dimensionality reduction and large-scale optimization: Part 1&2 Low-rank tensor decompositions. Foundations and Trends® in Machine Learning. 2016 Dec 18;9(4-5):249-429.

TENSOR AND MULTIVARIATE FUNCTIONS

Consider d-variate function: $f(x_1, \ldots, x_d)$

Sample each point on a grid $x_{i_k}, i_k = 1, \ldots, n_k$.

You get $n_1 \times n_2 \times \ldots \times n_d$ d-dimensional tensor!

WHERE TENSORS COME FROM

D-dimensional Partial Differential Equations (PDE)

$\Delta u = f$

Parametric equations:

$A(p)u(p) = f(p), p = (p_1, \ldots, p_M)$

Data: images, videos, hyperspectral images

Factor models

Weight tensors in deep neural networks…

WHERE TENSORS COME FROM

D-dimensional Partial Differential Equations (PDE) $\Delta u = f$

Parametric equations: $A(p)u(p) = f(p), p = (p_1, \ldots, p_M)$

Data: images, videos, hyperspectral images

Factor models

Weight tensors in deep neural networks…

DEFINITION

A tensor is a d-dimensional array:

$$A(i_1, \ldots, i_d), \quad 1 \leq i_k \leq n_k$$

Mathematically more correct definition: polylinear form

DEFINITIONS

Tensors (as matrices) they form a linear space.

The natural norm for tensors is Frobenius norm:

$$\|A\|_F = \sqrt{\sum_{i_1,\ldots,i_d} |A(i_1, \ldots, i_d)|}$$

CURSE OF DIMENSIONALITY

Curse of dimensionality: Storage of a d-dimensional tensor requires n^d

Grows exponentially with the dimensionality

BASIC QUESTIONS

How to break the curse of dimensionality?

How to perform multidimensional sampling

How to do everything efficiently and in a robust way?

REAL LIFE PROBLEMS

If you need to compute something high-dimensional, people typically do the following:

- Monte-Carlo integration
- Special basis sets (radial basis functions)
- Best N-term approximations (wavelets, sparse grids)
- Neural networks

But we want algebraic techniques…

SEPARATION OF VARIABLES

One of the few fruitful ideas is the idea of **separation of variables.**

SEPARATION OF VARIABLES

We have seen separation of variables in two dimensions:

$$A(i_1, i_2) \approx \sum_{\alpha=1}^{r} U_1(i_1, \alpha) U_2(i_2, \alpha)$$

Or in the matrix form, $A = U_1 U_2^\top$

IDEAL SEPARATION

Rank-1:

$$A(i_1, i_2) = U_1(i_1)U_2(i_2)$$

How we can generalize it to d dimensions?

IDEAL SEPARATION

D dimension, rank=1

$$A(i_1, i_2, \ldots, i_d) = U_1(i_1)U_2(i_2)\ldots U_d(i_d)$$

However, not many tensors, can be represented in this format

CANONICAL POLYADIC (CP) FORMAT

A tensor is said to be in the **canonical polyadic (CP) format**, if it can be represented as

$$A(i_1, \ldots, i_d) = \sum_{\alpha=1}^{r} U_1(i_1, \alpha) \ldots U_d(i_d, \alpha)$$

The minimal number r such that the equality is achieved is called **CP-rank**

CP-FORMAT

$$A(i_1, \ldots, i_d) = \sum_{\alpha=1}^{r} U_1(i_1, \alpha) \ldots U_d(i_d, \alpha)$$

What are the properties of the CP-format and CP-rank, and are they different from the matrix rank?

The answer is big yes! There is a big difference between matrix rank and tensor rank.

CP-FORMAT

$$A(i_1, \ldots, i_d) = \sum_{\alpha=1}^{r} U_1(i_1, \alpha) \ldots U_d(i_d, \alpha)$$

The answer is big yes! There is a big difference between matrix rank and tensor rank.

Computation of the CP-rank an NP-complete problem, i.e. it can not be done in polynomial time.

CP-FORMAT

$$A(i_1, \ldots, i_d) = \sum_{\alpha=1}^{r} U_1(i_1, \alpha) \ldots U_d(i_d, \alpha)$$

The answer is big yes! There is a big difference between matrix rank and tensor rank.

There exists a $9 \times 9 \times 9$ tensor for which the value of the CP rank is not known!

CP-DECOMPOSITION

There are good properties of the CP decomposition!

- The CP decomposition is unique (Kruskal theorem)
- The number of parameters is very low

CP-DECOMPOSITION

There are bad properties of the CP decomposition!

- The best approximation may not exist
- The computation of the rank can be a hard problem
- Algorithms may converge very slow.

BAD EXAMPLE (1)

Example $f(x_1, \ldots, x_d) = x_1 + \ldots + x_d$

The CP-rank is d.

This tensor can be approximated with rank with any accuracy!!

BAD EXAMPLE (2)

Example $f(x_1, \ldots, x_d) = x_1 + \ldots + x_d$

The CP-rank is d.

$$g = (1 + x_1 t)(1 + x_2 t)\ldots(1 + x_d t)$$

$$\frac{dg}{dt}\Big|_{t=0} = f$$

$\displaystyle \frac{dg}{dt} \approx \frac{g(h) - g(0)}{h} + \mathcal{O}(h)$ — rank 2 with any accuracy.

But what does it mean from the view point of algorithms?

ANOTHER EXAMPLE:

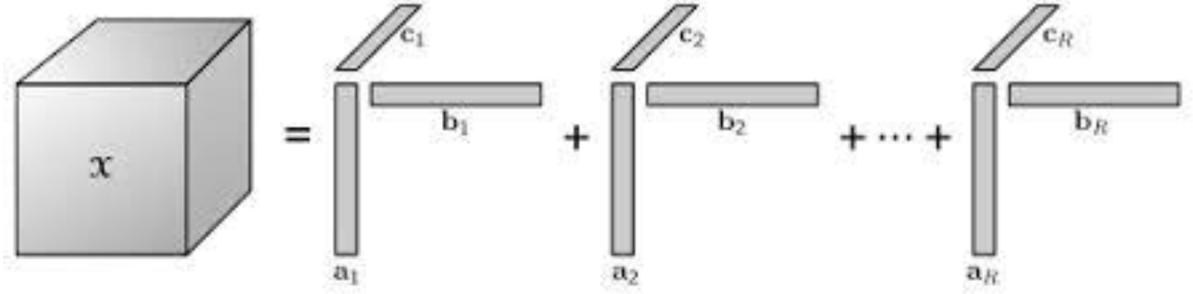
$$f(x_1, \ldots, x_d) = \sin(x_1 + \ldots + x_d)$$

For the complex field, the CP rank is 2.

For the real field, the CP rank is d (non-trivial!)

APPLICATIONS OF THE CP

Work on the CP decomposition has been started in **multiway factor analysis**



Each factor corresponds to pure mixture

COMPUTATION OF THE CP DECOMPOSITION

The main workhorse for the computation of the CP decomposition is **the alternating least squares** (ALS)

Each substep is a linear least squares!!

$$A = (U, V, W)$$

Fix V, W compute U

Fix U, W compute V

Fix U, V compute W

ALS: SOME INSIGHTS

Convergence of the ALS can be very slow.

Many improvements have been developed.

Best code: TensorLab (De Lathauwer laboratory, KU Leuven)

CP & ALS: SUMMARY

Useful in many factor models

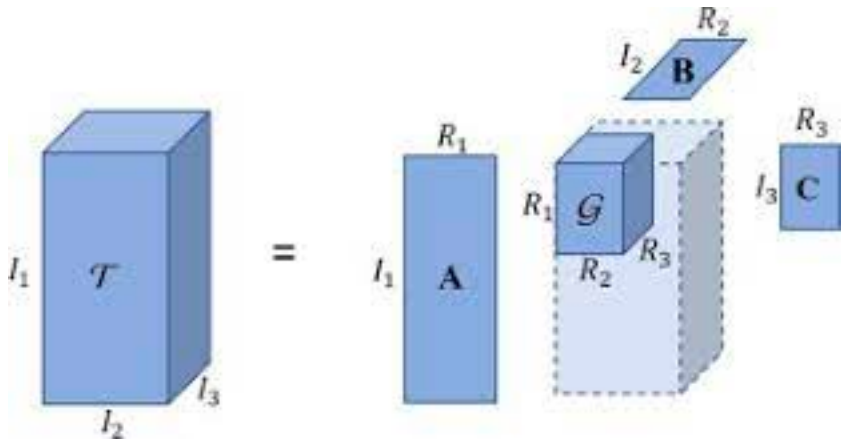
Sometimes difficult to compute

Not the only tensor decomposition around!

TUCKER DECOMPOSITION

Another attempt is the Tucker decomposition or Higher-Order SVD: HOSVD

It was proposed by Tucker (1966) and brought to mathematics by Lieven De Lathauwer in 2000.



TUCKER DECOMPOSITION

$$A(i_1, i_2, \ldots, i_d) = \sum_{\alpha_1,\ldots,\alpha_d} G(\alpha_1, \ldots, \alpha_d) U_1(i_1, \alpha_1) U_2(i_2, \alpha_2) \ldots U_d(i_d, \alpha_d)$$

It can be computed using SVD!

It has exponential dependence on d.

It is very good for small d.

HIGHER-ORDER SVD

$$A(i_1, i_2, \ldots, i_d) = \sum_{\alpha_1, \ldots, \alpha_d} G(\alpha_1, \ldots, \alpha_d) U_1(i_1, \alpha_1) U_2(i_2, \alpha_2) \ldots U_d(i_d, \alpha_d)$$

To compute U_k we need to compute the **unfolding** of the tensor into a matrix of size $n \times n^{d-1}$ and compute its left singular vectors.

One can use alternating least squares method.

It has much faster convergence!

CROSS APPROXIMATION

$$A(i_1, i_2, \ldots, i_d) = \sum_{\alpha_1,\ldots,\alpha_d} G(\alpha_1, \ldots, \alpha_d) U_1(i_1, \alpha_1) U_2(i_2, \alpha_2) \ldots U_d(i_d, \alpha_d)$$

You can generalize cross approximation to Tucker decomposition

(Oseledets, Savostyanov, Tucker dimensionality reduction of three-dimensional arrays in linear time).

We managed to compress $10^6 \times 10^6 \times 10^6$ arrays on a very old workstation in 2005.

LETS SUMMARIZE

CP decomposition: no curse of dimensionality, difficult to compute

Tucker decomposition: easy to compute, curse of dimensionality

Is there anything in between those formats?

Yes, and these are novel SVD based tensor formats: **tensor train, H-Tucker**

And more complicated representations called **tensor networks**

REMINDER

Canonical decomposition:

$$A(i_1, \dots, i_d) = \sum_{\alpha=1}^{r} U_1(i_1, \alpha) \dots U_d(i_d, \alpha_d)$$

Tucker decomposition

$$A(i_1, \dots, i_d) = \sum_{\alpha_1=1,\dots,\alpha_d}^{r} G(\alpha_1, \dots, \alpha_d) U_1(i_1, \alpha_1) \dots U_d(i_d, \alpha_d)$$

Exponential complexity!

Is there anything in between?

FIRST ATTEMPT

First attempt was just reshaping tensors into matrices.

Take a tensor, reshape it into $n^{d_1} \times n^{d_2}$ matrix

Compute SVD of the matrix:

$$A(\mathcal{I}, \mathcal{J}) \approx \sum_{\alpha=1}^{r} U_1(\mathcal{I}, \alpha) U_2(\mathcal{J}, \alpha), \quad \mathcal{I} \cup \mathcal{J} = (i_1, \ldots, i_d)$$

FIRST ATTEMPT

Compute SVD of the matrix:

$$A(\mathcal{I}, \mathcal{J}) \approx \sum_{\alpha=1}^{r} U_1(\mathcal{I}, \alpha) U_2(\mathcal{J}, \alpha), \quad \mathcal{I} \cup \mathcal{J} = (i_1, \ldots, i_d)$$

Now we can run it recursively.

If you do in naive way, you get r tensors with d/2 indices, leading to large complexity (which one)?

TREE TUCKER

Compute SVD of the matrix:

$$A(\mathcal{I}, \mathcal{J}) \approx \sum_{\alpha=1}^{r} U_1(\mathcal{I}, \alpha) U_2(\mathcal{J}, \alpha), \quad \mathcal{I} \cup \mathcal{J} = (i_1, \ldots, i_d)$$

Example: $\mathcal{I} = (i_1, i_4), \quad \mathcal{J} = (i_2, i_3, i_5)$

A smarter idea: consider $U_1(\mathcal{I}, \alpha)$ as $d_1 + 1$ dimensional tensor and compress.

Now we get real «dimensionality reduction» !

TREE TUCKER

Lemma: If A has canonical rank r, the new tensors have rank not higher than r

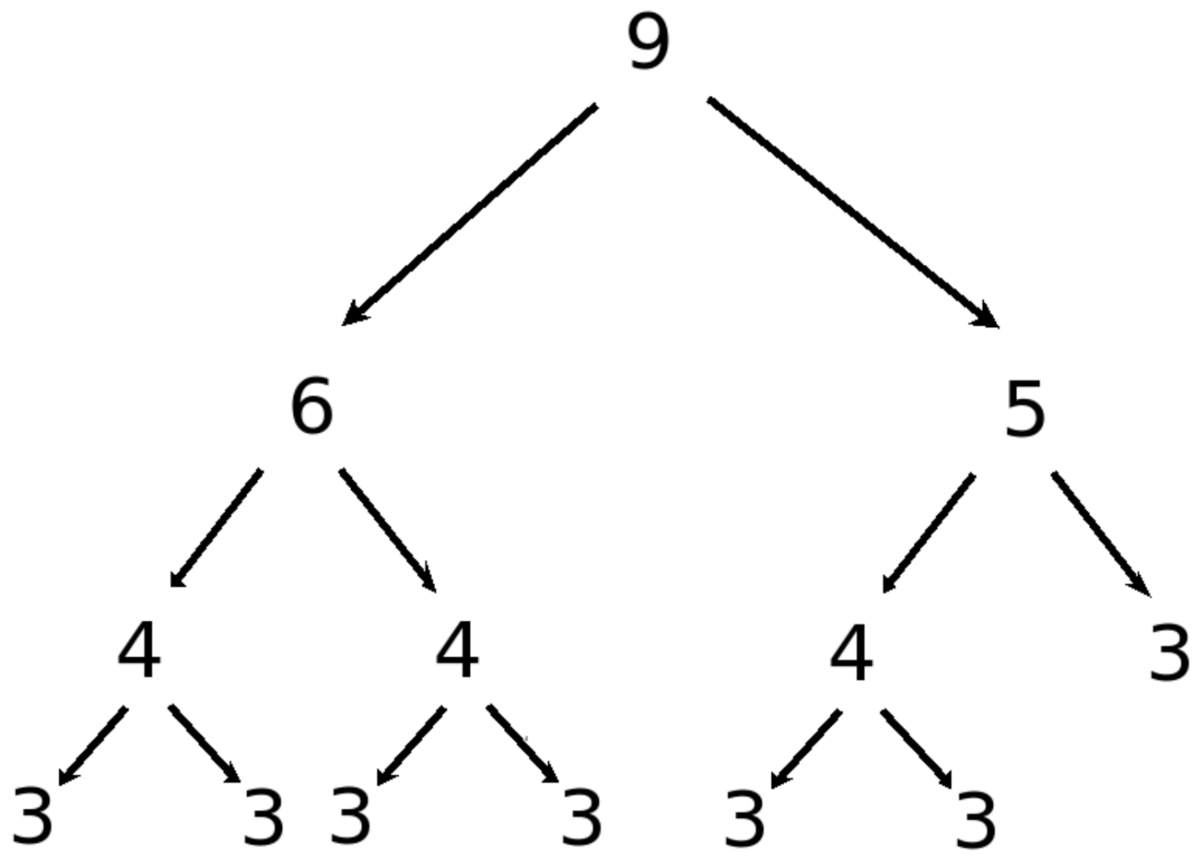
I. V. Oseledets, E.E. Tyrtyshnikov Breaking curse of dimensionality, or how to use SVD in many dimension. SISC, 2009.

TREE TUCKER

The process is then applied recursively:

We had a 9 dimensional tensor of canonical rank r, we split it into 4 and 5 indices, replace it by 6 = 5 + 1 and 5 = 4 + 1 dimensional tensors of canonical rank r. We can go on…

TREE TUCKER



TREE TUCKER: COMPLEXITY

The number of leafs in the tree is exactly $d - 2$

And we get $\mathcal{O}(dnr + (d-2)r^3)$ parameters!

I.e., no curse of dimensionality, but SVD-based algorithm!

EQUIVALENCE TO A SIMPLER MODEL

We quickly realized (March 2009) that this representation is equivalent to a much simpler one:

If we reorder indices in a tensor, we get the following representation:

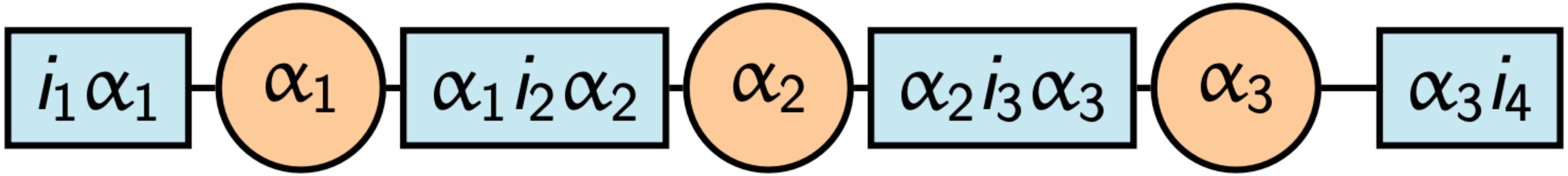
$$A(i_1, \ldots, i_d) = \sum_{\alpha_1, \ldots, \alpha_{d-1}} G_1(i_1, \alpha_1) G_2(\alpha_1, i_2, \alpha_2) \ldots\ldots G_d(\alpha_{d-1}, i_d)$$

which is now now known as **tensor train decomposition**

TENSOR TRAIN DECOMPOSITION

Tensor train

$$A(i_1, \ldots, i_d) = \sum_{\alpha_1,\ldots,\alpha_{d-1}} G_1(i_1, \alpha_1) G_2(\alpha_1, i_2, \alpha_2) \ldots G_d(\alpha_{d-1}, i_d)$$



MATRIX FORM OF TT-DECOMPOSITION

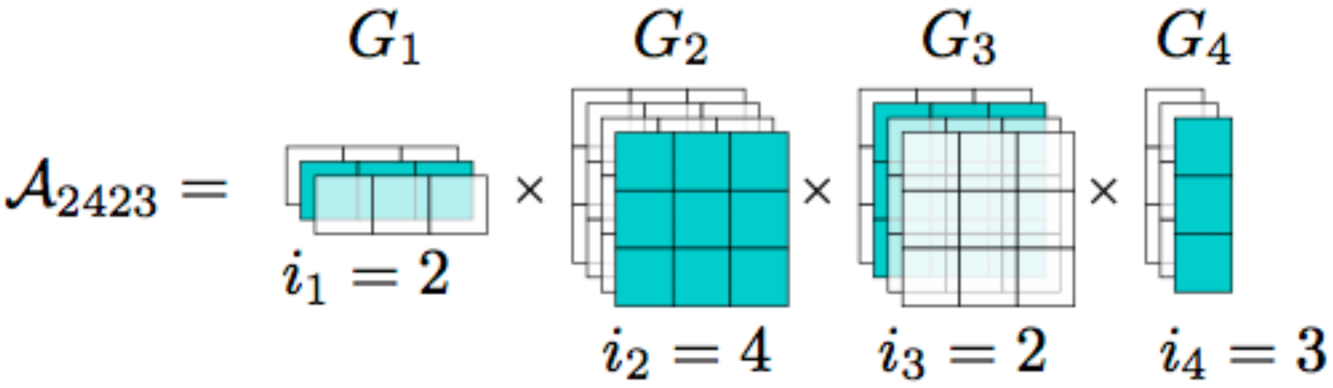
$$A(i_1, \ldots, i_d) = G_1(i_1) \ldots G_d(i_d),$$

Where $G_k(i_k)$ is a matrix of size $r_{k-1} \times r_k$ and $r_0 = r_d = 1$

VISUALIZATION OF TT-DECOMPOSITION

$A(i_1, \ldots, i_d) = G_1(i_1)\ldots G_d(i_d),$

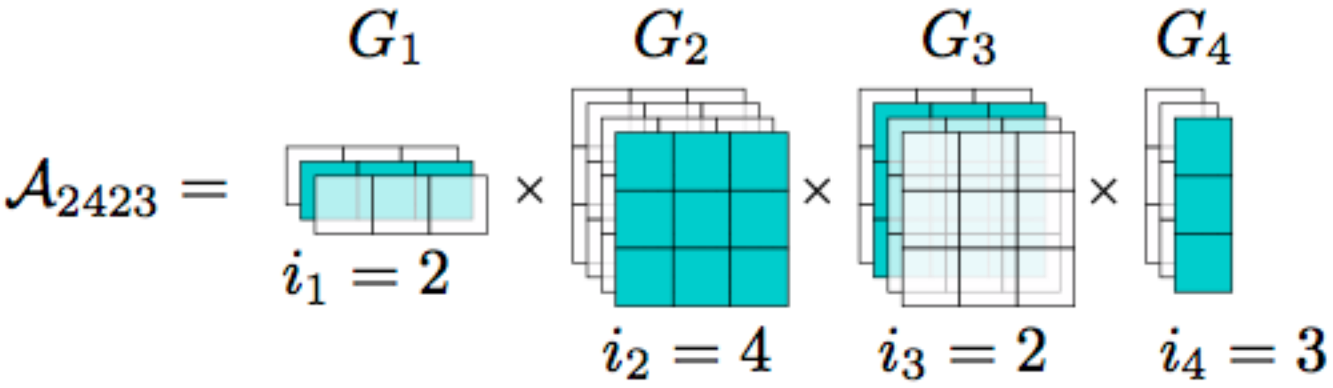
Where $G_k(i_k)$ is a matrix of size $r_{k-1} \times r_k$ and $r_0 = r_d = 1$



VISUALIZATION OF TT-DECOMPOSITION

$$A(i_1, \ldots, i_d) = G_1(i_1) \ldots G_d(i_d),$$

Matrices $G_k(i_k)$are called TT-cores, and r_k are called TT-ranks



H-TUCKER DECOMPOSITION

H-Tucker is another successful SVD-based tensor format.

In this representation, you apply Tucker decomposition **recursively**

I.e. we first reshape the tensor into $r^2 \times r^2 \times \ldots \times r^2$ array and compute Tucker decomposition,

Get an $r \times r \times \ldots \times r$ but with twice smaller dimension.

H-TUCKER AND TT-FORMAT

H-Tucker and TT are different formats, with H-Tucker more general,

but TT-format typically is simpler to implement and analyze.

There has been a line of research for comparing them.

TT-RANKS ARE MATRIX RANKS

Theorem (Oseledets, Tensor-Train decomposition, 2011)

$$r_k = \mathrm{rank} A_k, A_k = A(i_1 \ldots i_k; i_{k+1} \ldots i_d)$$

I.e. the ranks are matrix ranks of unfoldings !!

PROOF

Theorem (Oseledets, Tensor-Train decomposition, 2011)

$$r_k = \operatorname{rank} A_k, A_k = A(i_1 \ldots i_k; i_{k+1} \ldots i_d)$$

Proof is constructive and gives the TT-SVD algorithm

It takes only 50 lines of code to implement

PROPERTIES OF TT-FORMAT

- If the tensor has small canonical rank, then $r_k \leq r$
- TT-ranks are matrix ranks, **TT-SVD**
- All basic arithmetic, linear in d, polynomial in r
- Fast **TENSOR ROUNDING**
- TT-cross methods, exact interpolation formula
- Much more advanced stuff (tomorrow)

TT-RANKS ARE MATRIX RANKS

Theorem (Oseledets, Tensor-Train decomposition, 2011)

$$r_k = \text{rank} A_k, A_k = A(i_1 \ldots i_k; i_{k+1} \ldots i_d)$$

I.e. the ranks are matrix ranks of unfoldings !!

NO EXACT RANKS IN PRACTICE

Theorem (Oseledets, Tensor-Train decomposition, 2011)

If $A_k = R_k + E_k, \quad \mathrm{rank}(A_k) = r_k, \quad \|E_k\| \leq \varepsilon_k$

then there exists a TT-decomposition with

$$\|A - \mathbf{TT}\|_F \leq \sqrt{\sum_{k=1}^{d-1} \varepsilon_k^2}$$

TT-SVD

- A_1 is $n_1 \times (n_2 n_3 n_4)$
- $U_1, S_1, V_1 = \mathrm{SVD}(A_1)$, U_1 is $n_1 \times r_1$ — first core
- $A_2 = S_1 V_1,\ A_2$ is $r_1 \times (n_2 n_3 n_4)$
- Reshape it into $(r_1 n_2) \times (n_3 n_4)$ matrix A_2'
- Compute its SVD: $A_2' = G_2 S_2 V_2^\top$, U_2 is $(r_1 n_2) \times r_2$ — second core
- $A_3 = S_2 V_2^\top$ reshape into $(r_2 n_3) \times n_4$ and compute its SVD, resulting in the third and fourth core.

FAST AND SIMPLE LINEAR ALGEBRA

- Addition and Hadamard product, scalar product, matrix-by-vector product
- All scale linear in d

ADDITION

$$A(i_1, \ldots, i_d) = A_1(i_1) \ldots A_d(i_d), \quad B(i_1, \ldots, i_d) = B_1(i_1) \ldots B_d(i_d)$$

How the tensor $C = A + B$ look like?

MULTIPLICATION

$$A(i_1, \dots, i_d) = A_1(i_1)\dots A_d(i_d), \quad B(i_1, \dots, i_d) = B_1(i_1)\dots B_d(i_d)$$

How the tensor $C = A \circ B$ look like?

Answer: $C_k(i_k) = A_k(i_k) \otimes B_k(i_k)$, where $\otimes$ is a **Kronecker product of matrices**

TENSOR ROUNDING

Suppose we made an operation with tensors.

We got suboptimal tensor representation.

Can we find optimal ranks?

TENSOR ROUNDING (2)

Can we find optimal ranks?

Yes, we can.

The answer is given by **rounding**

ROUNDING: ILLUSTRATION IN 2D

Let us illustrate the idea in 2D.

In fact, there is a trick: if something works for matrices, it can be generalized to TT format (but it many be not very easy).

So, lets assume $A = UV^\top$

How to compute the SVD?

ROUNDING: ILLUSTRATION IN 2D …

So, lets assume $A = UV^\top$

How to compute the SVD?

Compute QR factorization (Lecture 1)

$$U = Q_U R_U, \quad V = Q_V R_V$$

Then, $A = Q_U M Q_V^\top$. It is enough to compute SVD of a small matrix!!

$M = \hat{U} S \hat{V}^\top$, then $A = (Q_U \hat{U}) S (Q_V \hat{V})^\top$ is the SVD of the matrix (why??)

TT-ROUNDING

In tensor rounding, you have to orthogonalize the cores from the left
(I will illustrate on a picture) and then start computing the SVD from the right.

This non-uniqueness of TT-format allows to orthogonalize representations!

TT-ROUNDING

Rounding can be done in $\mathcal{O}(dnr^3)$ operations

Now we do not have curse of dimensionality, but we have **curse of the rank**

TT-CROSS

Where do we get these 100-dimensional tensors from?

We can only know their elements or set them up implicitly
(as a solution of a certain minimization problem)

TT-CROSS

Recall, that for matrices we can recover a rank-r matrix from r columns and r rows using **skeleton decomposition**

$$A = C\hat{A}^{-1}R$$

Can we generalize it to TT-format?

Yes, we can

TT-CROSS

Oseledets, Tyrtyshnikov, TT-cross approximation of multidimensional arrays, 2010

You can exactly recover a rank-r tensor from $\mathcal{O}(dnr^3)$ elements.

Quite a lot of heuristics have been proposed (we are still working on this), but algorithms still work.

FOKKER PLANCK USING CROSS METHOD

$$dx = f(x,t)dt + S(x,t)d\beta, \quad d\beta d\beta^\top = Q(t)dt, \quad x = x(t) \in \mathbb{R}^d$$

Evolution is described by the Fokker-Planck equation for $\rho(x,t)$

$$\frac{\partial \rho}{\partial t} = \sum_{i=1}^{d}\sum_{j=1}^{d} \frac{\partial}{\partial x_i}\frac{\partial}{\partial x_j}[D_{ij}\rho] - \sum_{i=1}^{d}\frac{\partial}{\partial x_i}[f_i\rho]$$

We consider case $D = \gamma I$

KEY IDEAS

1: Parametrize density with TT-decomposition

2: Use splitting between diffusion and advection

3: Each splitting step can be integrated «almost explicitly»

DIFFUSION STEP

$\frac{\partial \rho}{\partial t} = \Delta \rho, \quad \rho_{k+1} = e^{\Delta \tau} \rho_k$, does not change the TT-rank (exponential of the Laplacian has TT-rank 1!)

CONVECTION STEP

$$\frac{\partial \rho}{\partial t} = \nabla \cdot (f\rho)$$

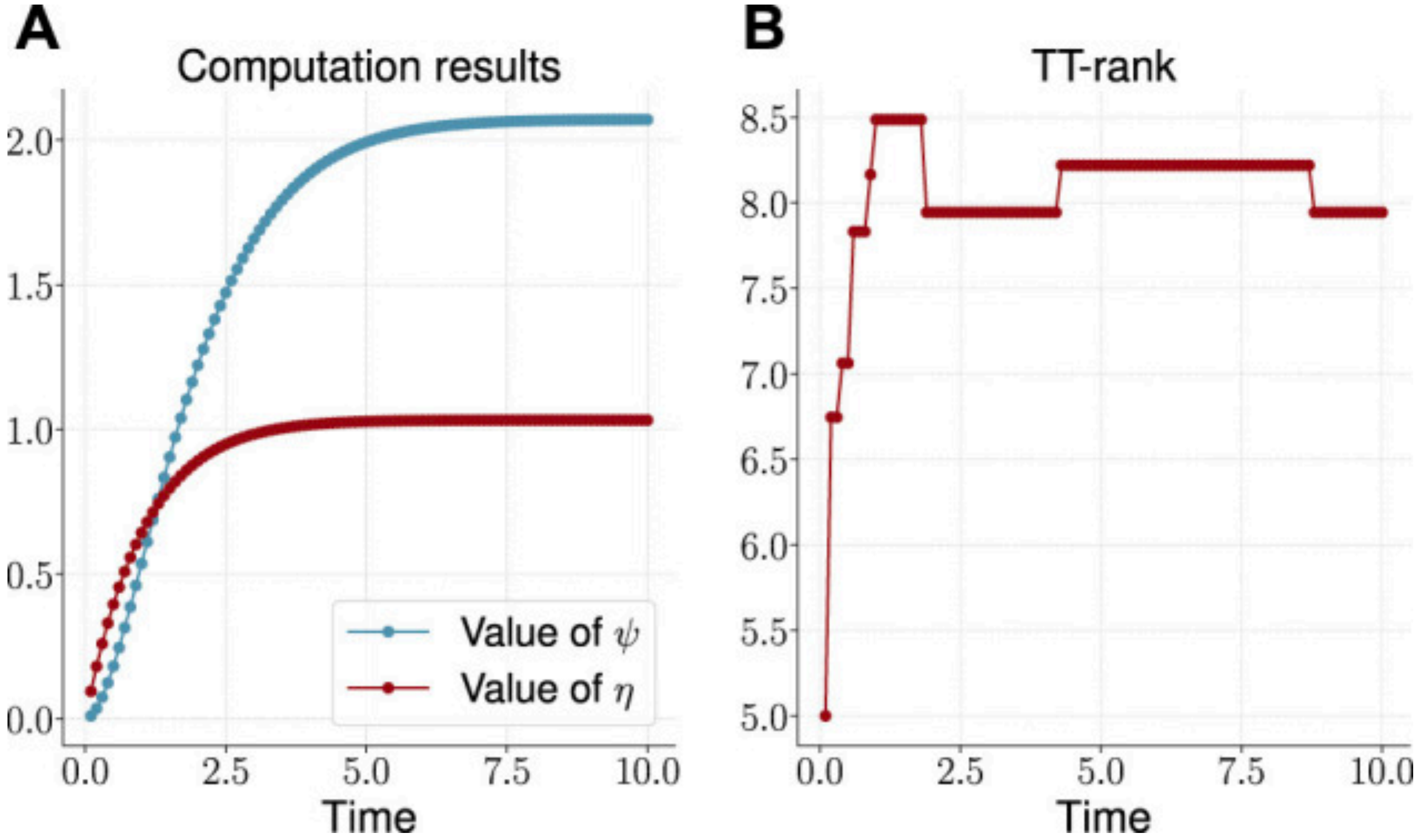
In Lagrangian coordinates,

$$\frac{dx}{dt} = f(x, t)$$

$$\frac{d\rho}{dt} = -\operatorname{Tr}(\nabla f)\rho$$

I.e. to evaluate the density in the next time step at any point, we integrate this system of ODE back in time. Thus, we can evaluate the density at any point, and can use TT-cross method to build the approximation!

NUMERICAL ILLUSTRATION



REFERENCE

A. Chertkov, I. Oseledets, Solution of the Fokker–Planck Equation by Cross Approximation Method in the Tensor Train Format, Front. Artif. Intelligence. 2021

APPROXIMATION OF PROBABILITY DISTRIBUTIONS FROM SAMPLES USING TENSORS

We are given samples $x_1, \ldots, x_N$ from the probability distribution

$$p(x) \approx q_\theta(x)$$

$$q_\theta(x) = \langle \alpha_\theta, \Phi(x) \rangle = \sum_{k=1}^{K} \alpha_{\theta,k} f_k(x)$$

Tensor-product basis: $\Phi(x) = f(x_1) \otimes \ldots \otimes f_d(x_d), \quad f_k(x) \in \mathbb{R}^{m_k}$

We put tensor-train constraints on α, which is a d-dimensional tensor!

LOSS FUNCTION

As a loss function, we use

$$\mathscr{L}(p(x) - q_\theta(x))^2 dx = \int q_\theta^2 dx - 2E_{x\sim p} q_\theta(x) + \text{const}$$

All these terms are computable.

SQUARED TT-MODEL

TT-format for the density is not positive;

We also propose to use squared TT model

$\hat{q} = q_\theta^2(x)$

It happens, that the complexity of the basic operations (sampling, loss evaluation, etc.) does not grow significantly with respect to the ranks.

WHY TT IS GOOD?

- Sampling is cheap
- Likelihood is available
- Optimization can be done by Riemannian optimization

Table 1: Comparison of the capabilities of different density estimation models. *FFJORD does not use true log-likelihood i
the training process and instead uses its unbiased estimate.

Method	Exact Sampling	Tractable LL	No middle-man Training	Computation of CDF
FFJORD	✓	✓*	✓*	✗
Normalizing Flows	✓	✓	✓	✗
GANs	✓	✗	✗	✗
VAEs	✓	✗	✓	✗
Autoregressive	✓	✓	✓	✗
Energy-based	✗	✗	✗	✗
TTDE (ours)	✓	✓	✓	✓

EXPERIMENTS

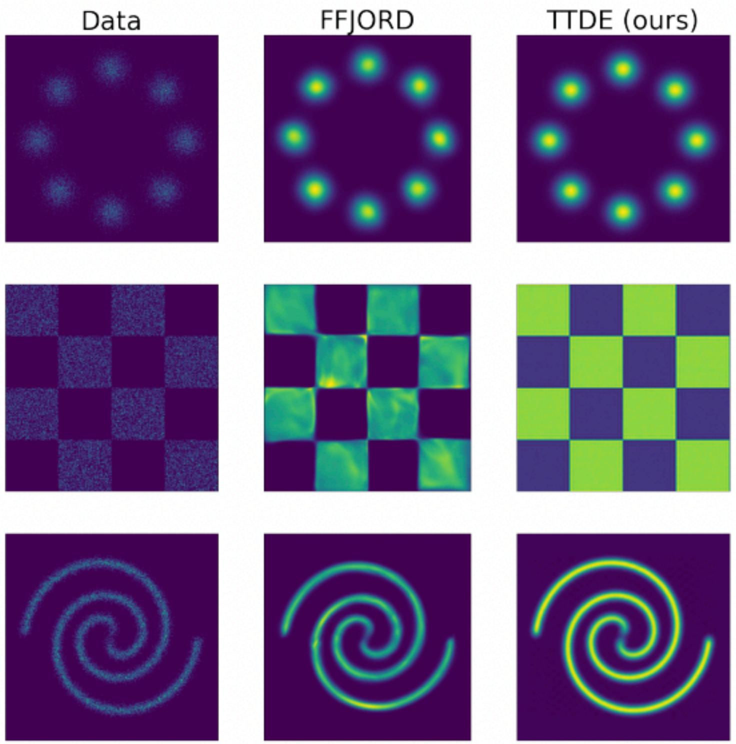


Figure 1: Comparison of TTDE and FFJORD models on 2-dimensional toy distributions.

	Random init.	Rank-1 init.
Adam	5	11
Riemannian	12	32

Table 2: Experiment with mixture of 7 Gaussians in 3D with additional dimensions containing only noise. We report the maximum dimensionality for which approximation of the density converges to the true one for different initialization settings and optimization methods used.

EXPERIMENTS

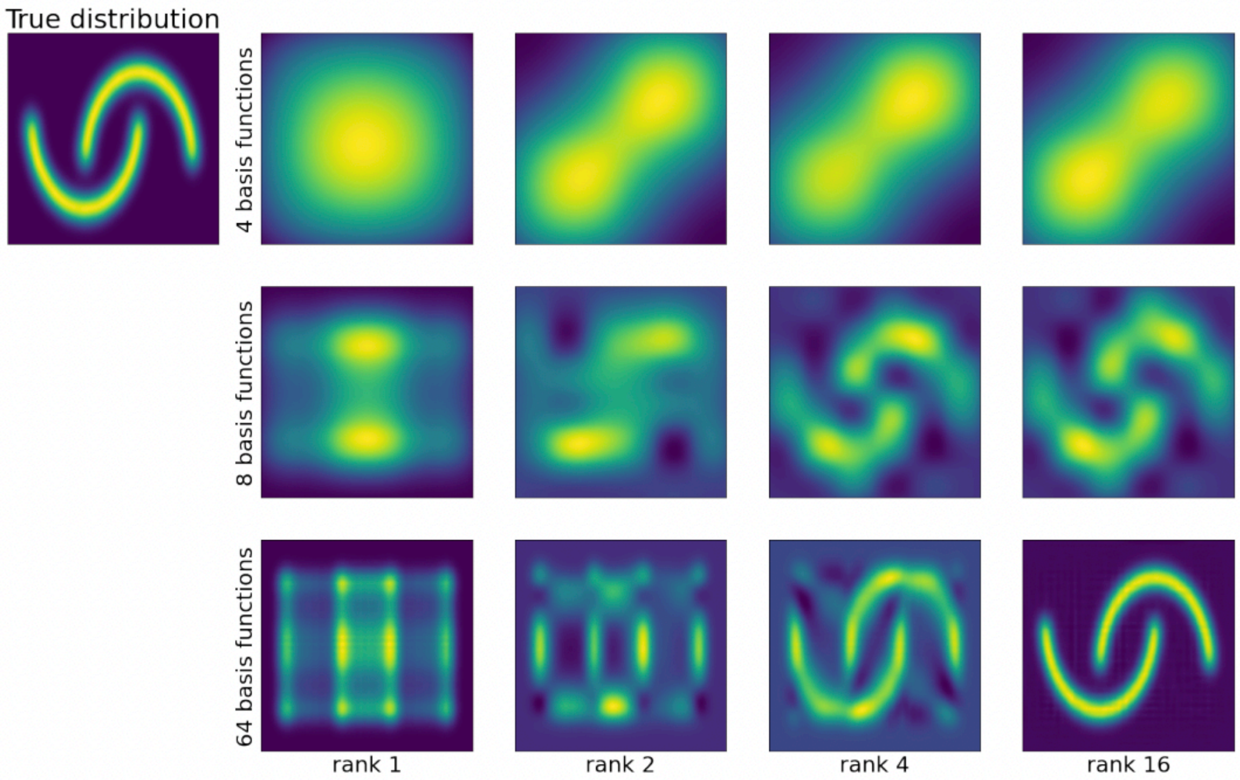


Figure 2: Approximations of “two moons” distribution by TTDE for different basis function set sizes and TT-ranks.

EXPERIMENTS

	POWER	GAS	HEPMASS	MINIBOONE	BSDS300
Dataset dimensionality	6	8	21	43	64
Gaussians	-7.74	-3.58	-27.93	-37.24	96.67
MADE	-3.08	3.56	-20.98	-15.59	148.85
Real NVP	0.17	8.33	-18.71	-13.84	153.28
Glow	0.17	8.15	-18.92	-11.35	155.07
FFJORD	0.46	8.59	**-14.92**	**-10.43**	**157.40**
Squared TTDE (ours)	**0.46**	**8.93**	−21.34*	−28.77*	143.30

EXPERIMENTS

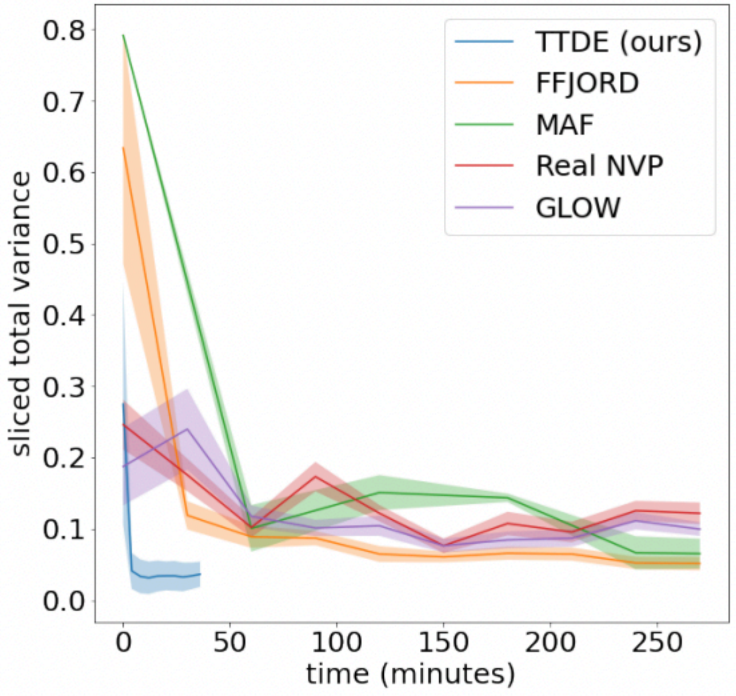


Figure 4: Dependence of the sliced total variation w.r.t. the training time for models trained on 6-dimensional UCI POWER dataset.

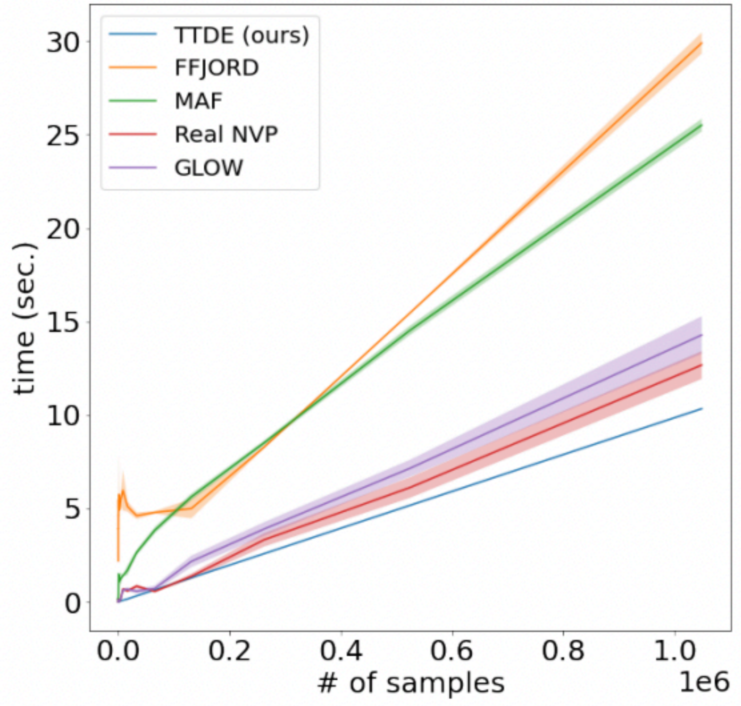


Figure 5: Dependence of the sampling time w.r.t. the number of samples to be generated for 6-dimensional space for models trained on UCI POWER dataset. Our model outperforms its competitors and shows 2.6, 2.5, 1.4 and 1.2 times speedups compared to FFJORD, MAF, GLOW and Real NVP respectively.

EXPERIMENTS

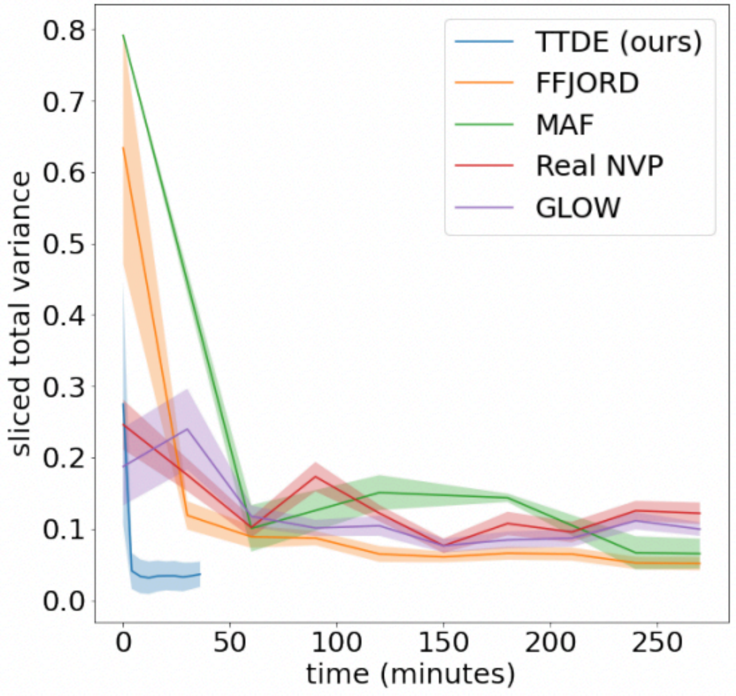


Figure 4: Dependence of the sliced total variation w.r.t. the training time for models trained on 6-dimensional UCI POWER dataset.

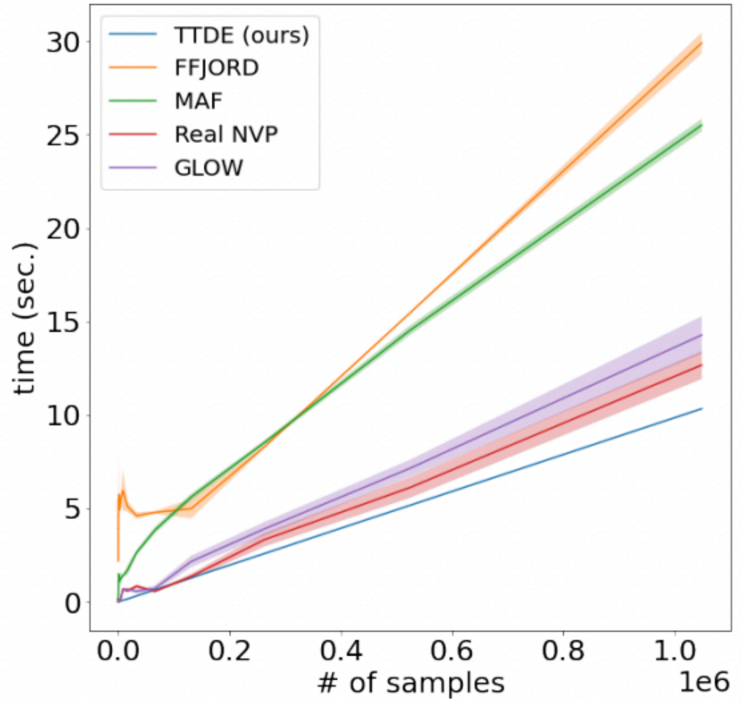


Figure 5: Dependence of the sampling time w.r.t. the number of samples to be generated for 6-dimensional space for models trained on UCI POWER dataset. Our model outperforms its competitors and shows 2.6, 2.5, 1.4 and 1.2 times speedups compared to FFJORD, MAF, GLOW and Real NVP respectively.

APPLICATIONS…

Tensor train neighborhood preserving **embedding** [PDF] ieee.org
W Wang, V Aggarwal, S Aeron - IEEE Transactions on Signal …, 2018 - ieeexplore.ieee.org
In this paper, we propose a **tensor train** neighborhood preserving **embedding** (TTNPE) to embed multidimensional **tensor** data into low-dimensional **tensor** subspace. Novel …
Save Cite Cited by 27 Related articles All 5 versions

Tt-rec: **Tensor train** compression for deep learning recommendation models [PDF] mlsys.org
C Yin, B Acun, CJ Wu, X Liu - Proceedings of Machine …, 2021 - proceedings.mlsys.org
… Component Analysis (PCA) (Wold et al., 1987), **Tensor**-**Train** (TT) is an approach to **tensor** decomposition by decomposing multidimensional data into product of smaller **tensors** …
Save Cite Cited by 13 Related articles All 4 versions

TTH-RNN: **Tensor-train** hierarchical recurrent neural network for video summarization [PDF] ieee.org
B Zhao, X Li, X Lu - IEEE Transactions on Industrial Electronics, 2020 - ieeexplore.ieee.org
… the out- put xe, and the mapping matrix We can be reshaped into **tensors** x ∊ Rdf1 … Besides, according to [22], the total computation complexity of the **tensor-train embedding** layer is O(d …
Save Cite Cited by 26 Related articles All 2 versions

Nimble GNN **Embedding** with **Tensor-Train** Decomposition [PDF] arxiv.org
C Yin, D Zheng, I Nisa, C Faloutos, G Karypis… - arXiv preprint arXiv …, 2022 - arxiv.org
This paper describes a new method for representing **embedding** tables of graph neural networks (GNNs) more compactly via **tensor-train** (TT) decomposition. We consider the …
Save Cite All 3 versions

Scalable gaussian processes with billions of inducing inputs via **tensor train** decomposition [PDF] mlr.press
P Izmailov, A Novikov… - … Conference on Artificial …, 2018 - proceedings.mlr.press
… allows deep kernels that produce **embeddings** of dimensionality up … **Tensor Train** (TT) decomposition, proposed in Os- eledets (2011 … allows to efficiently store **tensors** (multi- dimensional …
Save Cite Cited by 46 Related articles All 10 versions

[HTML] Tensorized **embedding** layers [HTML] aclanthology.org
O Hrinchuk, V Khrulkov, L Mirvakhabova… - Findings of the …, 2020 - aclanthology.org
… We introduce a novel way of parameterizing **embedding** layers based on the **Tensor Train** decomposition, which allows compressing the model significantly at the cost of a negligible …
Save Cite Cited by 10 Related articles All 3 versions

APPLICATIONS…

Learning a low **tensor-train** rank representation for hyperspectral image super-resolution [PDF] ieee.org

R Dian, S Li, L Fang - … on neural networks and learning systems, 2019 - ieeexplore.ieee.org

… In this paper, a novel low **tensor**- **train** (TT) rank (LTTR)-based HSI … each other and can constitute a 4-D **tensor**, whose four … impose the LTTR constraint on these 4-D **tensors**, which can …

Save Cite Cited by 146 Related articles All 5 versions

Multiscale feature **tensor train** rank minimization for multidimensional image recovery [PDF] ieee.org

H Zhang, XL Zhao, TX Jiang, MK Ng… - IEEE Transactions on …, 2021 - ieeexplore.ieee.org

… ZHANG et al.: MULTISCALE FEATURE **TENSOR TRAIN** RANK MINIMIZATION … is especially suitable for high- dimensional **tensors** [21], [28 … the resulting high-dimensional MSF **tensor** XW …

Save Cite Cited by 15 Related articles All 3 versions

TIE: Energy-efficient **tensor train**-based inference engine for deep neural network [PDF] acm.org

C Deng, F Sun, X Qian, J Lin, Z Wang… - Proceedings of the 46th …, 2019 - dl.acm.org

… Among various DNN compression techniques, **tensor-train** (TT) decomposition [52 … via decomposing the weight **tensors** of CONV … From the perspective of **tensor** theory, the impressive …

Save Cite Cited by 44 Related articles All 5 versions

Fast and accurate **tensor** completion with total variation regularized **tensor trains** [PDF] ieee.org

CY Ko, K Batselier, L Daniel, W Yu… - IEEE Transactions on …, 2020 - ieeexplore.ieee.org

… 21]–[23], which are intrin- sically defined on three-way **tensors** such as … [26] adopted **tensor trains** (TTs) and … **tensor** completion problem formulation and adopted the **tensor train** format as …

Save Cite Cited by 25 Related articles All 12 versions

SOFTWARE FOR TT-FORMAT

- tntoch https://github.com/rballester/tntorch (PyTorch)
- Teneva https://github.com/AndreiChertkov/teneva (Python)
- TT-Toolbox https://github.com/oseledets/TT-Toolbox (MATLAB)
- ttpy https://github.com/oseledets/ttpy (Python + numpy)
- T3f https://github.com/Bihaqo/t3f (Tensorflow)

RECAP OF LECTURE 3

- Tensors
- Basic tensor decompositions
- Advanced tensor decompositions.

NEXT LECTURE

- Multivariate function approximation
- Cross approximation
- Approximation of smooth functions