Tensor Decomposition and Its Applications in Machine Learning

Zhongkai Hao

Content

- Introduction to tensor decomposition
- Application-- NERF
- Application– Solving high dimensional PDE
- Application– Neural network compression

Definition of Tensor

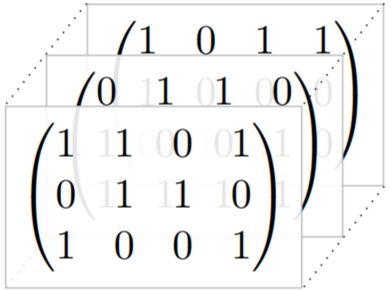
- In mathematics, tensor is a high dimensional array $\mathbf{T} \in \mathbb{R}^{n_1 \times \cdots \times n_d}$,
- Example of 1d, 2d, 3d tensor

$$\begin{pmatrix} 1 \\ 0 \\ 1 \\ 1 \\ 0 \end{pmatrix}$$

(a)

$$\begin{pmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 \\ 1 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \end{pmatrix}$$

(b)



(c)

- Storage consumption of a tensor $O(n^d)$

Tensor Calculus

- Addition $$(\mathbf{T}+\mathbf{U})_{x_1,\ldots,x_d} = \mathbf{T}_{x_1,\ldots,x_d} + \mathbf{U}_{x_1,\ldots,x_d}.$$
- Scalar multiply $$(\lambda \cdot \mathbf{T})_{x_1,\ldots,x_d} = \lambda \cdot \mathbf{T}_{x_1,\ldots,x_d}.$$
- Index contraction $$\mathbf{T} \in \mathbb{R}^{m_1\times\ldots\times m_d\times p_1\times\ldots\times p_f},$$ $$\mathbf{U} \in \mathbb{R}^{n_1\times\ldots\times n_e\times p_1\times\ldots\times p_f}.$$

$$\mathbf{V}_{x_1,\ldots,x_d,y_1,\ldots,y_e} = \sum_{z_1=1}^{p_1}\cdots\sum_{z_f=1}^{p_f} \mathbf{T}_{x_1,\ldots,x_d,z_1,\ldots,z_f} \cdot \mathbf{U}_{y_1,\ldots,y_e,z_1,\ldots,z_f}.$$

- Tensor multiplication

Definition 2.2.3. *For tensors* $\mathbf{G} \in \mathbb{R}^{M\times N}$ *and* $\mathbf{H} \in \mathbb{R}^{N\times P}$ *with index sets* $M = (m_1,\ldots,m_d)^T$, $N = (n_1,\ldots,n_d)^T$, *and* $P = (p_1,\ldots,p_d)^T$, *the product* $\mathbf{G}\cdot\mathbf{H} \in \mathbb{R}^{M\times P}$ *is defined as*

$$(\mathbf{G}\cdot\mathbf{H})_{x_1,y_1,\ldots,x_d,y_d} = \sum_{z_1=1}^{n_1}\cdots\sum_{z_d=1}^{n_d} \mathbf{G}_{x_1,z_1,\ldots,x_d,z_d} \cdot \mathbf{H}_{z_1,y_1,\ldots,z_d,y_d}, \tag{2.2.3}$$

Tensor Calculus

- Tensor product $(\mathbf{T} \otimes \mathbf{U})_{x_1,\ldots,x_d,y_1,\ldots,y_e} = \mathbf{T}_{x_1,\ldots,x_d} \cdot \mathbf{U}_{y_1,\ldots,y_e},$

- Property of tensor product

Theorem 2.2.6. *Let* $\mathbf{G} \in \mathbb{R}^{M \times N}$ *and* $\mathbf{H} \in \mathbb{R}^{N \times P}$ *with* $\mathbf{G} = \mathbf{G}_1 \otimes \mathbf{G}_2$ *and* $\mathbf{H} = \mathbf{H}_1 \otimes \mathbf{H}_2$, *where*

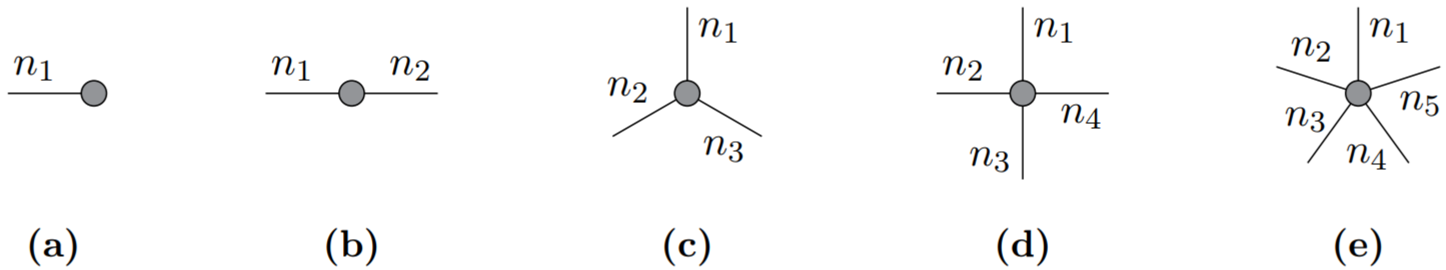
$$\mathbf{G}_1 \in \mathbb{R}^{(m_1 \times n_1) \times \ldots \times (m_e \times n_e)}, \quad \mathbf{G}_2 \in \mathbb{R}^{(m_{e+1} \times n_{e+1}) \times \ldots \times (m_d \times n_d)},$$
$$\mathbf{H}_1 \in \mathbb{R}^{(n_1 \times p_1) \times \ldots \times (n_e \times p_e)}, \quad \mathbf{H}_2 \in \mathbb{R}^{(n_{e+1} \times p_{e+1}) \times \ldots \times (n_d \times p_d)}.$$

Then, the product of $\mathbf{G}$ *and* $\mathbf{H}$ *is given by*

$$\mathbf{G} \cdot \mathbf{H} = (\mathbf{G}_1 \otimes \mathbf{G}_2) \cdot (\mathbf{H}_1 \otimes \mathbf{H}_2) = (\mathbf{G}_1 \cdot \mathbf{H}_1) \otimes (\mathbf{G}_2 \cdot \mathbf{H}_2).$$

Graphical representation of tensor

- Example



- Representing index contraction and tensor multiplication

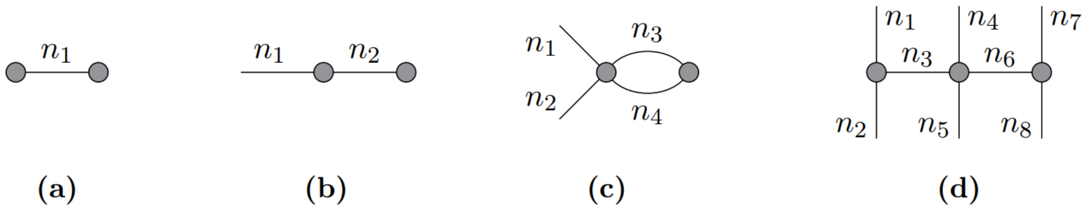


Figure 2.3: *Graphical representation of tensor contractions: (a) Inner product of two vectors. (b) Matrix-vector product. (c) Two-dimensional contraction of two tensors. (d) Contraction of three tensors.*

Matricization and Vectorization

- Define a bijection

$$\phi_N : \{1,\dots,n_1\} \times \cdots \times \{1,\dots,n_d\} \to \{1,\dots,\prod_{k=1}^{d} n_k\},$$

$$\phi_N(x_1,\dots,x_d) = 1 + (x_1 - 1) + \ldots + (x_d - 1) \cdot n_1 \cdot \ldots \cdot n_{d-1}$$
$$= 1 + \sum_{k=1}^{d} (x_k - 1) \prod_{l=1}^{k-1} n_l.$$

- Matricization
- Vectorization

Definition 2.4.2. *Let $N = (n_1,\dots,n_d)^T$ be an index set and $\mathbf{T} \in \mathbb{R}^N$ a tensor. For two ordered subsets $N' = (n_{k_1},\dots,n_{k_e})^T$ and $N'' = (n_{l_1},\dots,n_{l_f})^T$ of N which satisfy* (2.4.3), *the matricization of $\mathbf{T}$ with respect respect to N' and N'' is given by*

$$\left(\mathbf{T}\Big|_{N'}^{N''}\right)_{\overline{x_{k_1},\dots,x_{k_e}},\overline{x_{l_1},\dots,x_{l_f}}} = \mathbf{T}_{x_1,\dots,x_d}. \qquad (2.4.4)$$

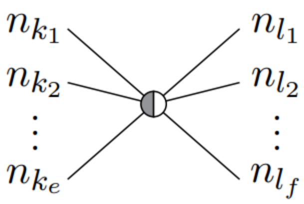
Definition 2.4.3. *Let $N = (n_1,\dots,n_d)^T$ be an index set and $\mathbf{T} \in \mathbb{R}^N$ a tensor. For a reordering $N' = (n_{k_1},\dots,n_{k_d})^T$, the vectorization of $\mathbf{T}$ is given by*

$$\left(\mathbf{T}\Big|_{N'}\right)_{\overline{x_{k_1},\dots,x_{k_d}}} = \mathbf{T}_{x_1,\dots,x_d}.$$

Orthonormality

- **Definition 2.6.1.** *Let* $\mathbf{T} \in \mathbb{R}^N$, $N = (n_1, \ldots, n_d)^T$, *be a tensor and* $N', N'' \subset N$ *a splitting of the modes with* $N' = (n_{k_1}, \ldots, n_{k_e})^T$ *and* $N'' = (n_{l_1}, \ldots, n_{l_f})^T$, $e+f = d$. $\mathbf{T}$ *is called orthonormal with respect to* N' *if the matricization of* $\mathbf{T}$ *with respect to the sets* N' *and* N'' (2.4.4) *satisfies*

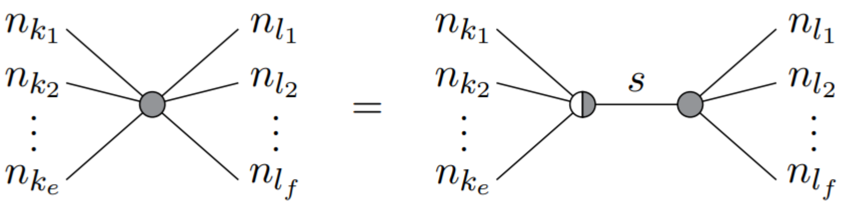
$$\mathbf{T}\Big|_{N'}^{N''} \cdot \left(\mathbf{T}\Big|_{N'}^{N''}\right)^T = \mathbf{T}\Big|_{N'}^{N''} \cdot \mathbf{T}\Big|_{N''}^{N'} = I \in \mathbb{R}^{N' \times N'}.$$



(a)

- QR decomposition

- $$\mathbf{T}\Big|_{N'}^{N''} = Q \cdot R = \mathbf{Q}\Big|_{N'}^{s} \cdot \mathbf{R}\Big|_{s}^{N'},$$



(a)

Tensor Decomposition

- Rank-one tensor

Definition 3.1.1. *A tensor* $\mathbf{T} \in \mathbb{R}^N$, $\mathbb{R}^N = \mathbb{R}^{n_1 \times \cdots \times n_d}$, *of order* d *is called* rank-one tensor *if it can be written as the tensor product of* d *vectors, i.e.*

$$\mathbf{T} = \bigotimes_{i=1}^{d} \mathbf{T}^{(i)} = \mathbf{T}^{(1)} \otimes \cdots \otimes \mathbf{T}^{(d)}, \tag{3.1.1}$$

where $\mathbf{T}^{(i)} \in \mathbb{R}^{n_i}$ *for* $i = 1, \ldots, d$.

- Storage consumption of rank-one tensor $O(nd)$
- Canonical format

$$\mathbf{T} = \sum_{k=1}^{r} \bigotimes_{i=1}^{d} \mathbf{T}_{k,:}^{(i)} = \sum_{k=1}^{r} \mathbf{T}_{k,:}^{(1)} \otimes \cdots \otimes \mathbf{T}_{k,:}^{(d)},$$

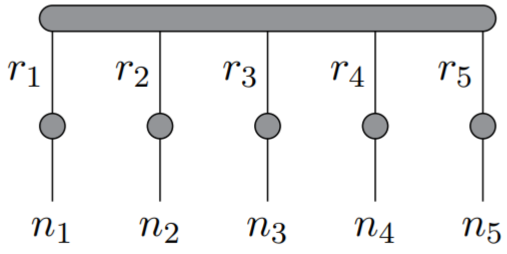
- Problems: best canonical format might not exist

Tensor Decomposition

- Tucker format

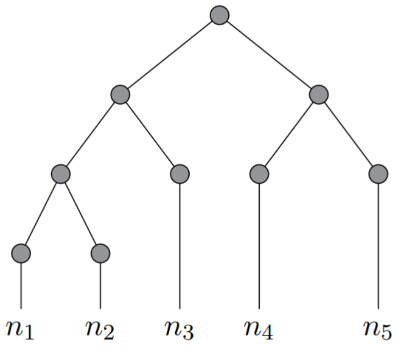
Definition 3.3.1. *A tensor* $\mathbf{T} \in \mathbb{R}^N$ *is said to be in the* Tucker format *if*

$$\mathbf{T} = \sum_{k_1=1}^{r_1} \cdots \sum_{k_d=1}^{r_d} \left(\mathbf{T}^{(1)}_{:,k_1} \otimes \cdots \otimes \mathbf{T}^{(d)}_{:,k_d} \right) \cdot \mathbf{U}_{k_1,\dots,k_d}$$
$$= \left(\mathbf{T}^{(1)} \otimes \cdots \otimes \mathbf{T}^{(d)} \right) \cdot \mathbf{U},$$



(a)

- Graphical illustration of Tucker format
- Storage consumption $O(rnd + r^d)$
- Improved: Hierarchical Tucker format



(b)

Tensor Decomposition—Tensor Train format

- Tensor Train decomposition
- **Definition 3.4.1.** *A tensor* $\mathbf{T} \in \mathbb{R}^N$ *is said to be in the* TT format *if*

$$\mathbf{T} = \sum_{k_0=1}^{r_0} \cdots \sum_{k_d=1}^{r_d} \bigotimes_{i=1}^{d} \mathbf{T}^{(i)}_{k_{i-1},:,k_i} = \sum_{k_0=1}^{r_0} \cdots \sum_{k_d=1}^{r_d} \mathbf{T}^{(1)}_{k_0,:,k_1} \otimes \cdots \otimes \mathbf{T}^{(d)}_{k_{d-1},:,k_d}.$$

- query element using tensor train format

$$\mathbf{T}_{x_1,\ldots,x_d} = \sum_{k_0=1}^{r_0} \cdots \sum_{k_d=1}^{r_d} \mathbf{T}^{(1)}_{k_0,x_1,k_1} \cdot \ldots \cdot \mathbf{T}^{(d)}_{k_{d-1},x_d,k_d} = \mathbf{T}^{(1)}_{:,x_1,:} \cdot \ldots \cdot \mathbf{T}^{(d)}_{:,x_d,:}.$$

- TT format for tensor operator

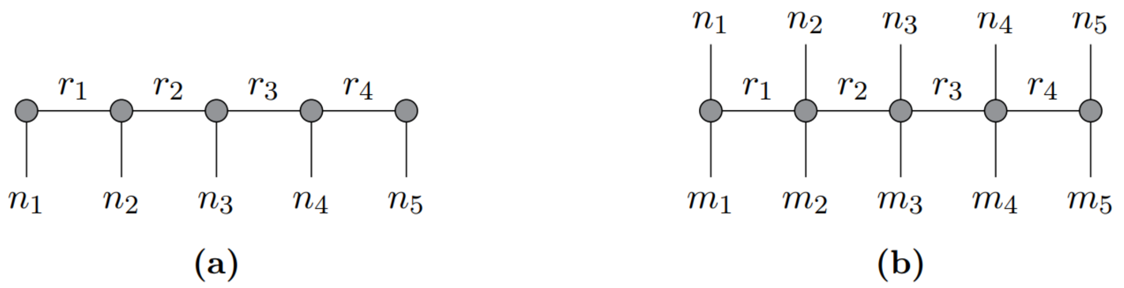
Definition 3.4.2. *A tensor operator* $\mathbf{G} \in \mathbb{R}^{M \times N}$ *is said to be in the* TT format *if*

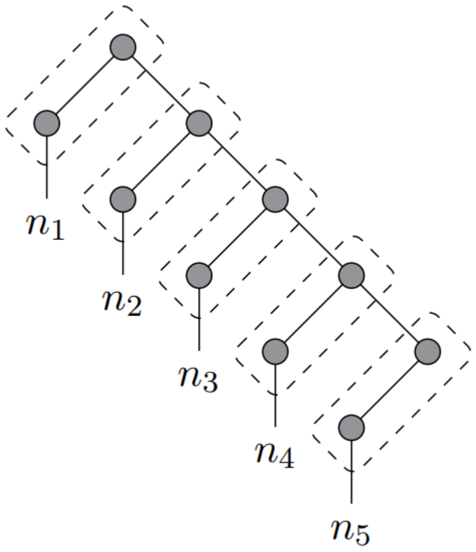
$$\mathbf{G} = \sum_{k_0=1}^{r_0} \cdots \sum_{k_d=1}^{r_d} \bigotimes_{i=1}^{d} \mathbf{G}^{(i)}_{k_{i-1},:,:,k_i} = \sum_{k_0=1}^{r_0} \cdots \sum_{k_d=1}^{r_d} \mathbf{G}^{(1)}_{k_0,:,:,k_1} \otimes \cdots \otimes \mathbf{G}^{(d)}_{k_{d-1},:,:,k_d},$$

Tensor Train decomposition

$$\mathbf{T} = \left[\mathbf{T}^{(1)}\right] \otimes \cdots \otimes \left[\mathbf{T}^{(d)}\right]$$

- Graphical illustration of tensor train format





- Memory consumption of tensor train formate $O(r^2nd)$
- Tensor train format as a special case of HT format

Addition/Multiplication for tensor train

- Addition

Theorem 3.4.4. *For tensor operators* $\mathbf{G}_1, \mathbf{G}_2 \in \mathbb{R}^{M \times N}$ *with TT representations*

$$\mathbf{G}_1 = \left[\mathbf{G}_1^{(1)}\right] \otimes \cdots \otimes \left[\mathbf{G}_1^{(d)}\right], \quad \mathbf{G}_2 = \left[\mathbf{G}_2^{(1)}\right] \otimes \cdots \otimes \left[\mathbf{G}_2^{(d)}\right],$$

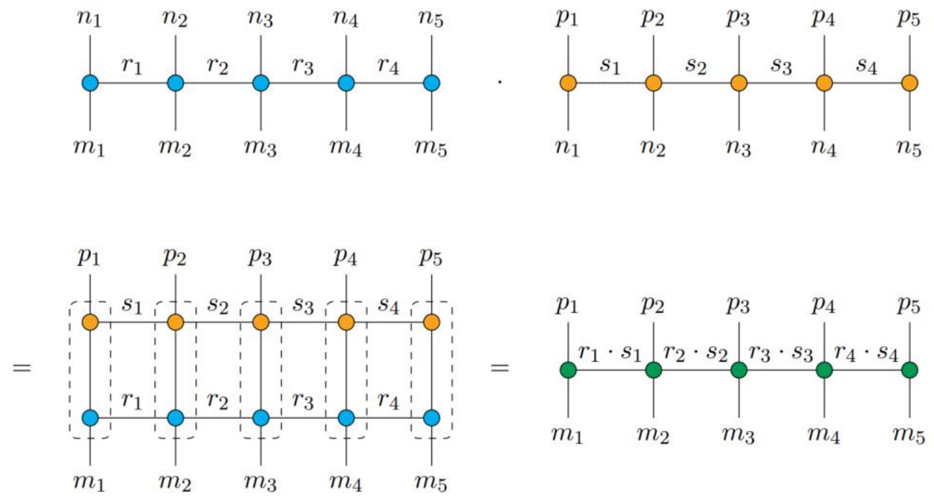
the sum $\mathbf{G} = \mathbf{G}_1 + \mathbf{G}_2$ *is given by*

$$\mathbf{G} = \left[\left[\mathbf{G}_1^{(1)}\right] \left[\mathbf{G}_2^{(1)}\right]\right] \otimes \begin{bmatrix} \left[\mathbf{G}_1^{(2)}\right] & 0 \\ 0 & \left[\mathbf{G}_2^{(2)}\right] \end{bmatrix} \otimes \cdots$$

$$\cdots \otimes \begin{bmatrix} \left[\mathbf{G}_1^{(d-1)}\right] & 0 \\ 0 & \left[\mathbf{G}_2^{(d-1)}\right] \end{bmatrix} \otimes \begin{bmatrix} \left[\mathbf{G}_1^{(d)}\right] \\ \left[\mathbf{G}_2^{(d)}\right] \end{bmatrix}.$$

- Multiplication

$$\mathbf{G}^{(i)}_{\overline{k_{i-1},l_{i-1}},:,:,\overline{k_i,l_i}} = \left(\mathbf{G}_1^{(i)}\right)_{k_{i-1},:,:,k_i} \cdot \left(\mathbf{G}_2^{(i)}\right)_{l_{i-1},:,:,l_i},$$

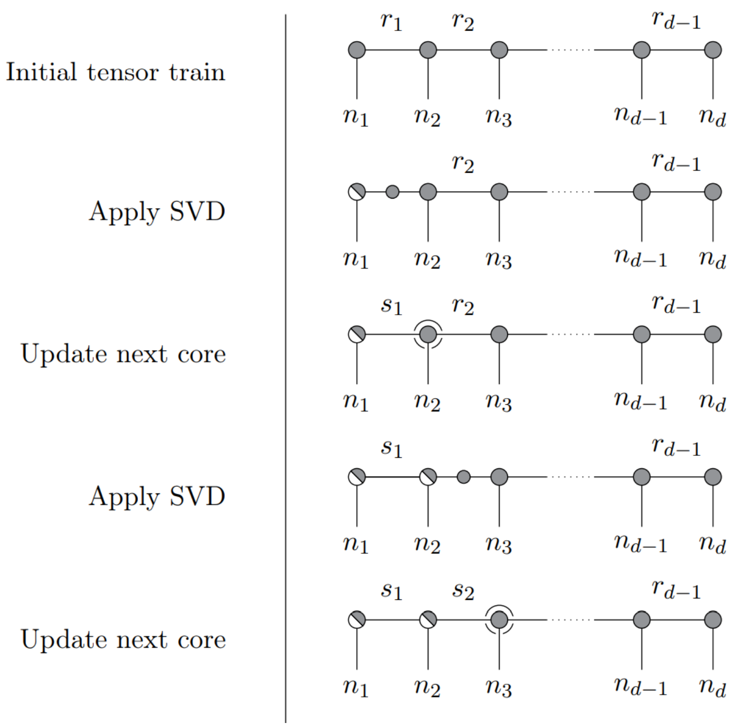


Orthonormalization

- Left unfolding $\mathbf{T}^{(i)} \in \mathbb{R}^{r_{i-1} \times n_i \times r_i}$ $\mathcal{L}\left(\mathbf{T}^{(i)}\right) = \mathbf{T}^{(i)}\Big|_{r_{i-1},n_i}^{r_i}$
- Left orthonormal

$$\left(\mathcal{L}\left(\mathbf{T}^{(i)}\right)\right)^T \cdot \mathcal{L}\left(\mathbf{T}^{(i)}\right) = \mathbf{T}^{(i)}\Big|_{r_i}^{r_{i-1},n_i} \cdot \mathbf{T}^{(i)}\Big|_{r_{i-1},n_i}^{r_i} = I \in \mathbb{R}^{r_i \times r_i}.$$

- SVD of tensor train one by one



Other tensor decomposition formats

- Quantized Tensor-Train Format
- For tensor $\mathbf{G} \in \mathbb{R}^{M \times N}$,we could decompose its dimension $m_i = m_{i,1} \cdot \ldots \cdot m_{i,c_i}$ and $n_i = n_{i,1} \cdot \ldots \cdot n_{i,c_i}$,
- $$\mathbf{G}' \in \mathbb{R}^{(m_{1,1} \times n_{1,1}) \times \cdots \times (m_{1,c_1} \times n_{1,c_1}) \times \cdots \times (m_{d,1} \times n_{d,1}) \times \cdots \times (m_{d,c_d} \times n_{1,c_d})}$$
- Then apply TT format to this tensor

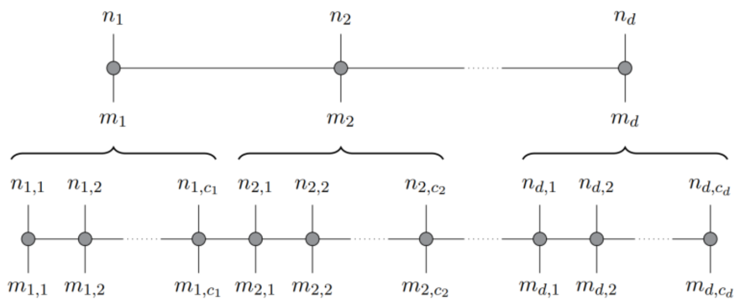
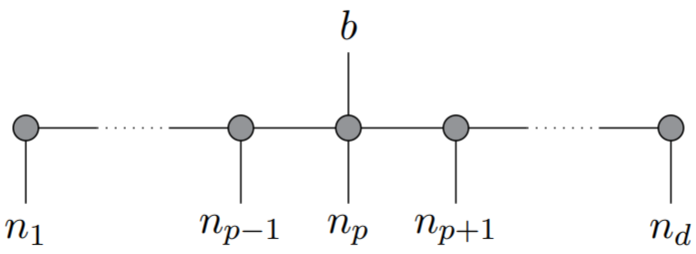
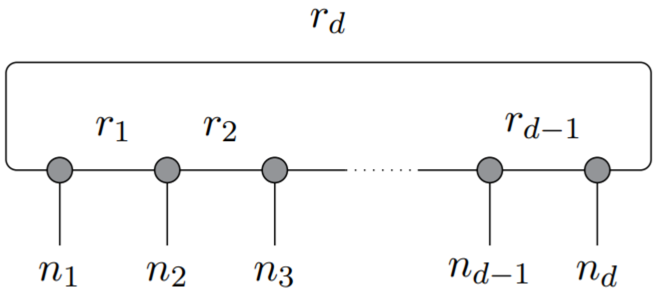


Figure 3.9: *Conversion from TT into QTT format: Each core is divided into several cores with smaller mode sizes.*

- block TT/ cycle TT



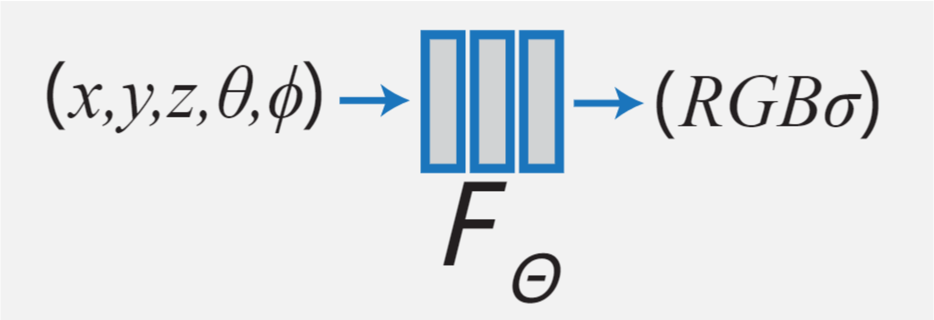


Applications of Tensor decomposition in machine learning

- Tensor decomposition offers a approximation for high dimensional data.
- Current applications of tensor decomposition

- Neural radiance field
- Solving high dimensional PDEs
- Second order optimizer
- Neural network/Data compression…

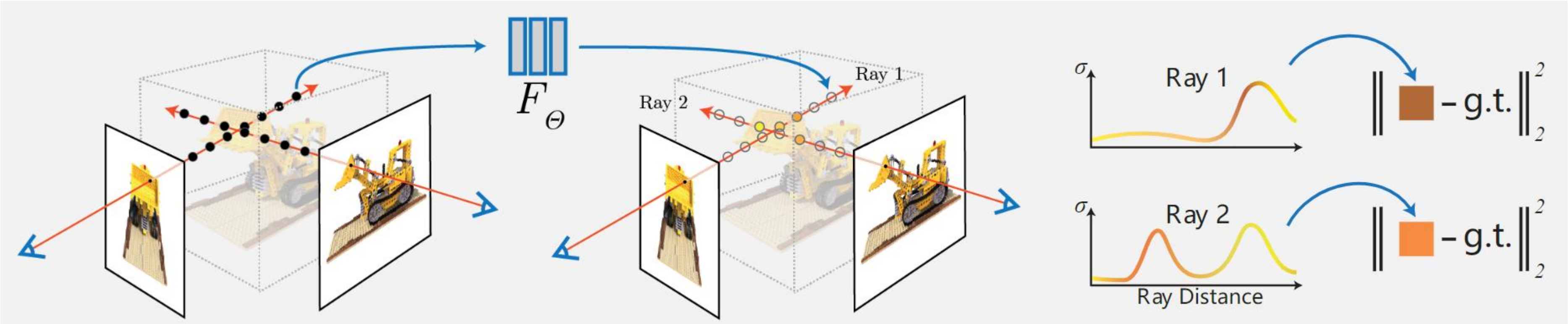
Neural Radiance Fields



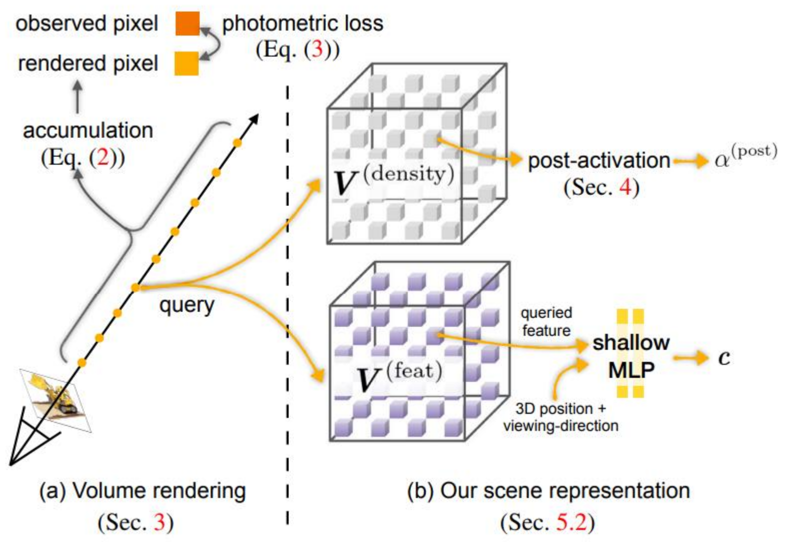
- Neural radiance field
- Problem: given a set of images with camera parameters, output images at other camera parameters

- Network io: xyz->rgb
- Loss function of NERF $\mathcal{L} = \sum_{r \in R} \left[\left\| \hat{C}_c(r) - C(r) \right\|_2^2 - \left\| \hat{C}_f(r) - C(r) \right\|_2^2 \right]$

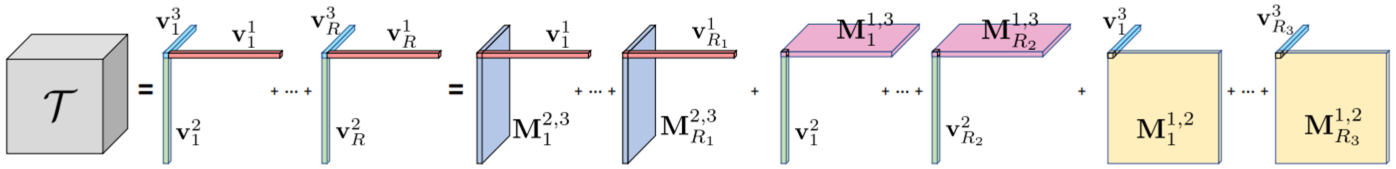
- Drawbacks of NerF: slow training



TensoRF (compressed voxel)

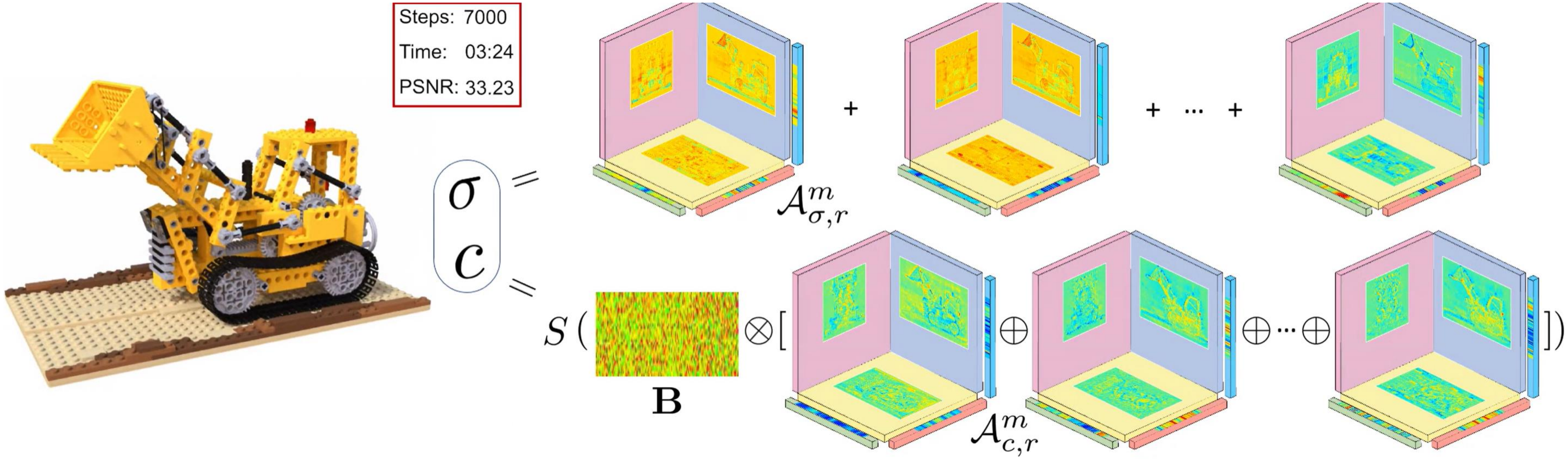


- Voxel based scene representation is much faster
- However, it requires $O(n^3)$ to store the voxels
- Tensor decomposition provides a promising compression scheme
- 1. TensorCP
- 2. TensorVM (vector-matrix)



TensoRF

- visualization result



TensoRF

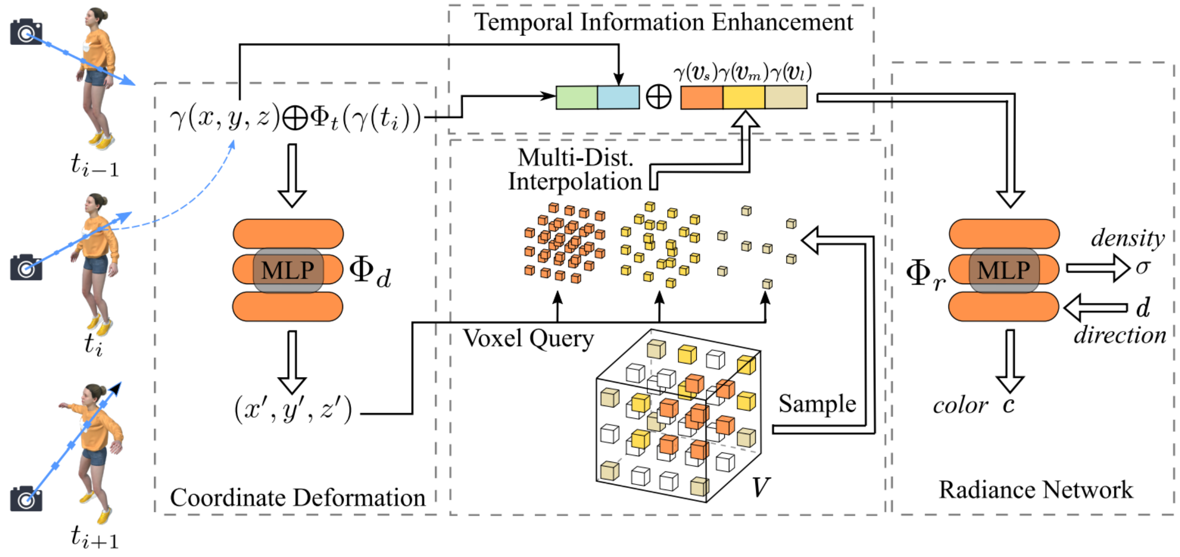
- Compression result for TensoRF
- Good trade-off performance between speed (~30min training time) and memory

Method	BatchSize	Steps	Time ↓	Size(MB)↓	PSNR↑	SSIM↑
			Synthetic-NeRF			
SRN [36]	-	-	>10h	-	22.26	0.846
NSVF [18]	8192	150k	>48*h	-	31.75	0.953
NeRF [24]	4096	300k	~35h	5.00	31.01	0.947
SNeRG [12]	8192	250k	~15h	1771.5	30.38	0.950
PlenOctrees [47]	1024	200k	~15h	1976.3	31.71	0.958
Plenoxels [46]	5000	128k	11.4m	778.1	31.71	0.958
DVGO [37]	5000	30k	15.0m	612.1	31.95	0.957
Ours-CP-384	4096	30k	25.2m	3.9	31.56	0.949
Our-VM-192-SH	4096	30k	16.8m	71.9	32.00	0.955
Ours-VM-48	4096	30k	13.8m	18.9	32.39	0.957
Ours-VM-192	4096	15k	8.1m	71.8	32.52	0.959
Ours-VM-192	4096	30k	17.4m	71.8	33.14	0.963

Dynamic NERF

- Four dimensional voxel (4d tensor) is unaffordable!
- Use multi-resolution 3d voxel + MLP for compressing

-
- 1. query voxel features from multiple resolution
- 2. query time features using MLP with fourier ebd
- 3. concat features and use another MLP(deformnet)



Dynamic NERF

Table 1: Comparisons about training/memory cost and rendering quality on synthetic scenes.

Method	w/ Time Enc.	w/ Explicit Rep.	Time	Storage	PSNR ↑	SSIM ↑	LPIPS ↓
NeRF [Mildenhall et al. 2020]	✗	✗	~ hours	5 MB	19.00	0.87	0.18
DirectVoxGO [Sun et al. 2022]	✗	✓	5 mins	205 MB	18.61	0.85	0.17
Plenoxels [Yu et al. 2022]	✗	✓	6 mins	717 MB	20.24	0.87	0.16
T-NeRF [Pumarola et al. 2021]	✓	✗	~ hours	–	29.51	0.95	0.08
D-NeRF [Pumarola et al. 2021]	✓	✗	20 hours	4 MB	30.50	0.95	0.07
TiNeuVox-S (ours)	✓	✓	8 mins	8 MB	30.75	0.96	0.07
TiNeuVox-B (ours)	✓	✓	28 mins	48 MB	**32.67**	**0.97**	**0.04**

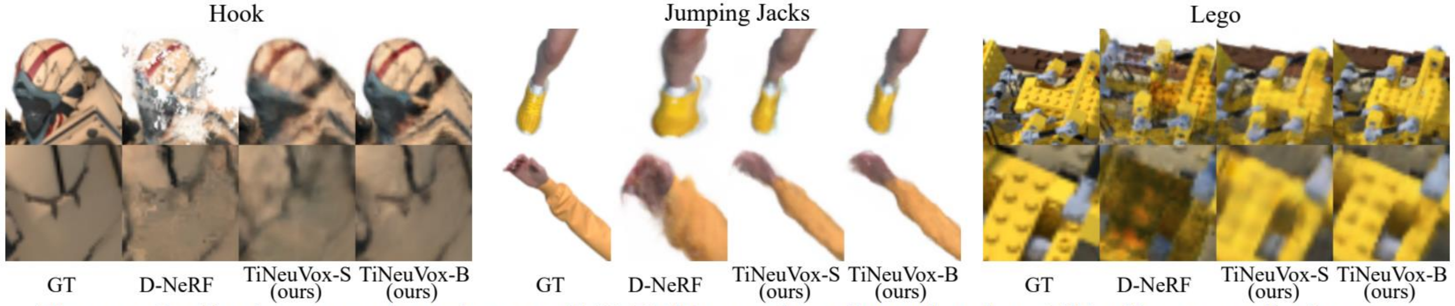


Figure 4: Qualitative comparisons between D-NeRF [Pumarola et al. 2021] and our TiNeuVox on synthetic scenes.

- Observations
- 1. voxel with tensor decomposition provides good explainability and speed-memory trade-off
- 2. combine neural methods with voxel methods are better (… need tune parameters)

Solving high dimensional PDEs

- High dimensional parabolic PDEs with the following terminal condition and vanishing boundary condition,

$$(\partial_t + L)V(x,t) + h(x,t,V(x,t),(\sigma^\top \nabla V)(x,t)) = 0 \quad (1)$$

$$L = \frac{1}{2}\sum_{i,j=1}^{d}(\sigma\sigma^\top)_{ij}(x,t)\partial_{x_i}\partial_{x_j} + \sum_{i=1}^{d} b_i(x,t)\partial_{x_i},$$

$$V(x,T) = g(x),$$

- For such PDEs, the solution is associated with the following stochastic process

$$\mathrm{d}X_s = b(X_s,s)\,\mathrm{d}s + \sigma(X_s,s)\,\mathrm{d}W_s, \quad X_0 = x_0,$$

$$Y_s = V(X_s,s), \qquad Z_s = (\sigma^\top \nabla V)(X_s,s)$$

$$\mathrm{d}Y_s = -h(X_s,s,Y_s,Z_s)\,\mathrm{d}s + Z_s \cdot \mathrm{d}W_s,$$

- We could simulate the stochastic process as

$$\widehat{X}_{n+1} = \widehat{X}_n + b(\widehat{X}_n,t_n)\Delta t + \sigma(\widehat{X}_n,t_n)\xi_{n+1}\sqrt{\Delta t},$$
$$\widehat{Y}_{n+1} = \widehat{Y}_n - h_n\Delta t + \widehat{Z}_n \cdot \xi_{n+1}\sqrt{\Delta t},$$

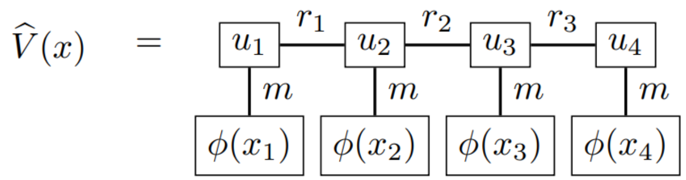
Solving high dimensional PDEs

- In each time step, we solve the following least square problem

$$\mathbb{E}\left[\left(\widehat{V}_n(\widehat{X}_n) - h_{n+1}\Delta t - \widehat{V}_{n+1}(\widehat{X}_{n+1})\right)^2\right]$$

- Representing solution with tensor train format

$$\widehat{V}(x_1, \ldots, x_d) = \sum_{i_1=1}^{m} \cdots \sum_{i_d=1}^{m} c_{i_1,\ldots,i_d} \phi_{i_1}(x_1) \cdots \phi_{i_d}(x_d),$$

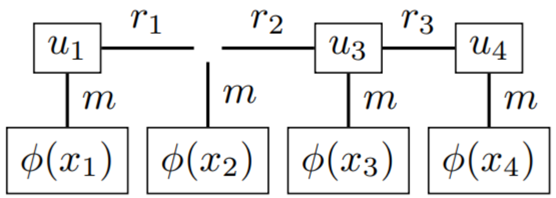


- We choose polynomial functions as basis functions for $\varphi(x)$.

Solving high dimensional PDEs

$$\arg\min_{\widehat{V}\in\mathcal{U}} \sum_{j=1}^{J} |\widehat{V}(x_j) - R(x_j)|^2,$$

- Solving the regression problem using SALSA (rank adaptive stable alternating least s
- Simple LS algorithm
- Simulate the PDE



Algorithm 1 simple ALS algorithm

Input: initial guess $u_1 \circ u_2 \circ \cdots \circ u_d$.
Output: result $u_1 \circ u_2 \circ \cdots \circ u_d$.
repeat
  for $i = 1$ **to** d **do**
    identify the local basis functions (19), parametrized by u_k, $k \neq j$
    optimize u_i using the local basis by solving the local least squares problem
  end for
until $noChange$ is $true$

Algorithm 2 PDE approximation

Input: initial parametric choice for the functions $\widehat{V}_n$ for $n \in \{0, \ldots, N-1\}$
Output: approximation of $V(\cdot, t_n) \approx \widehat{V}_n$ along the trajectories for $n \in \{0, \ldots, N-1\}$
Simulate K samples of the discretized SDE (7).
Choose $\widehat{V}_N = g$.
for $n = N-1$ **to** 0 **do**
  approximate either (10) or (11) (both depending on $\widehat{V}_{n+1}$) using Monte Carlo
  minimize this quantity (explicitly or by iterative schemes)
  set $\widehat{V}_n$ to be the minimizer
end for

Examples

- Hamilton-Jacobi-Bellman equation\

$$(\partial_t + \Delta) V(x,t) - |\nabla V(x,t)|^2 = 0,$$
$$V(x,T) = g(x), \qquad g(x) = \log\left(\tfrac{1}{2} + \tfrac{1}{2}|x|^2\right)$$

$$b = \mathbf{0}, \quad \sigma = \sqrt{2}\,\mathrm{Id}_{d\times d}, \quad h(x,s,y,z) = -\frac{1}{2}|z|^2$$

- Reference solution

$$V(x,t) = -\log \mathbb{E}\left[e^{-g(x+\sqrt{T-t}\sigma\xi)}\right],$$

- Comparision between TT approximation and NN approximation

	$\mathrm{TT}_{\mathrm{impl}}$	$\mathrm{TT}_{\mathrm{expl}}$	$\mathrm{NN}_{\mathrm{impl}}$	$\mathrm{NN}_{\mathrm{expl}}$
$\widehat{V}_0(x_0)$	4.5903	4.5909	4.5822	4.4961
relative error	$5.90\mathrm{e}^{-5}$	$3.17\mathrm{e}^{-4}$	$1.71\mathrm{e}^{-3}$	$2.05\mathrm{e}^{-2}$
reference loss	$3.55\mathrm{e}^{-4}$	$5.74\mathrm{e}^{-4}$	$4.23\mathrm{e}^{-3}$	$1.91\mathrm{e}^{-2}$
PDE loss	$1.99\mathrm{e}^{-3}$	$3.61\mathrm{e}^{-3}$	90.89	91.12
comp. time	41	25	44712	25178

Table 1. Comparison of approximation results for the HJB equation in $d = 100$.

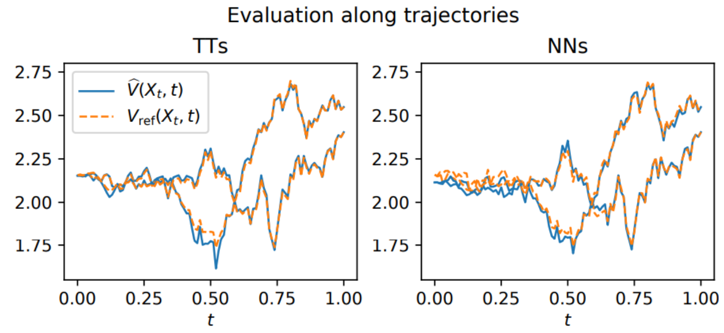


Figure 5. Reference solutions compared with implicit TT and NN approximations along two trajectories in $d = 10$.

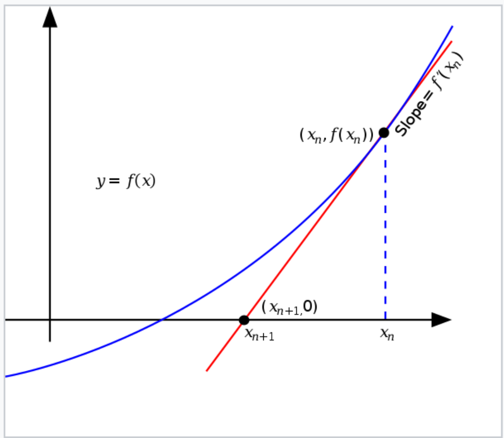
Examples

- HJB equation
- Performance with different degree for basis function

	Polynom. degree			
	0	1	2	3
$\widehat{V}_0(x_0)$	0.294	0.312	0.312	0.312
PDE loss	9.04e^{-2}	7.80e^{-4}	1.05e^{-3}	5.06e^{-4}
comp. time	110	3609	4219	5281

- We find that low polynomial degree (linear) is more efficient and are easier to optimize.
- Observation (blessing of dimensionality):
- The required polynomial degree decreases with increasing dimension. (almost constant in high dimensional space)

Second order optimizer– Low rank approximation of Hessian matrix



x_{n+1} is a better approximation than x_n for the root x of the function f (blue curve)

- Shampoo optimizer is a preconditioned gradient descent
- A general framework of quasi-newton method

$$\Delta x = -B^{-1} \nabla f(x_k).$$

- For accurate Newton's method, B should be the Hessian matrix $\frac{\partial^2 f}{\partial x_i \partial x_j}$
- For quasi- Newton's method, B could be (low rank/block/…)approximation of Hessian matrix
- For NN with parameters N, the size of Hessian matrix is N^2, inversion of Hessian is intractable

Second order optimizer of NN

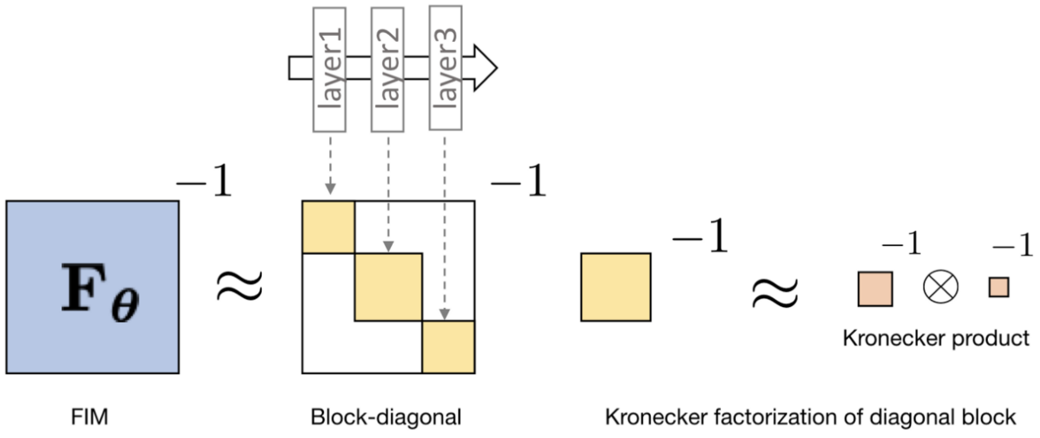
- Use approximation of Hessian matrix rather than the direct inversion
- Natural gradient descent

$$\mathbf{F}_{\boldsymbol{\theta}} = \mathbb{E}_{(\mathbf{x},\mathbf{y})}\left[\nabla \log p(\mathbf{y}|\mathbf{x};\boldsymbol{\theta}) \nabla \log p(\mathbf{y}|\mathbf{x};\boldsymbol{\theta})^{\mathrm{T}}\right]$$

- Update of NGD

$$\boldsymbol{\theta}^{(t+1)} = \boldsymbol{\theta}^{(t)} - \epsilon \boxed{\underbrace{\boxed{\mathbf{F}^{-1}_{\boldsymbol{\theta}^{(t)}}}}_{①} \nabla E(\boldsymbol{\theta}^{(t)})}$$

- Under layer independence simplication, we have



Approximation of (inverse of) FIM by K-FAC

Second order optimizer of DNN

- Every submatrix is a Kronecker product (tensor product)
- Its inversion is

$$\mathbf{A} \otimes \mathbf{B} := \begin{pmatrix} [\mathbf{A}]_{1,1}\mathbf{B} & \cdots & [\mathbf{A}]_{1,n}\mathbf{B} \\ \vdots & \ddots & \vdots \\ [\mathbf{A}]_{m,1}\mathbf{B} & \cdots & [\mathbf{A}]_{m,n}\mathbf{B} \end{pmatrix} \in \mathbb{R}^{ma \times nb}$$

$$(\mathbf{A} \otimes \mathbf{B})^{-1} = \mathbf{A}^{-1} \otimes \mathbf{B}^{-1}$$

$\mathbf{A} \in \mathbb{R}^{m \times n}, \mathbf{B} \in \mathbb{R}^{a \times b}$: Kronecker factors

- Specifically, each block is calculated by

$$\mathbf{F}_i = \mathbb{E}\left[\nabla_i \log p(\mathbf{y}|\mathbf{x};\boldsymbol{\theta}) \nabla_i \log p(\mathbf{y}|\mathbf{x};\boldsymbol{\theta})^{\mathrm{T}}\right]$$

$$\nabla_i \log p(\mathbf{y}|\mathbf{x};\boldsymbol{\theta}) = \mathbf{g}_i \otimes \mathbf{a}_{i-1} \in \mathbb{R}^{d_{i-1} \cdot d_i}$$

$\mathbf{a}_{i-1} \in \mathbb{R}^{d_{i-1}}$: the input to i-th layer (activation of (i-1)-th layer)

$\mathbf{g}_i = \dfrac{\partial \log p(\mathbf{y}|\mathbf{x};\boldsymbol{\theta})}{\partial \mathbf{s}_i} \in \mathbb{R}^{d_i}$: the gradient for the output of i-th layer

- Finally,

$$\begin{aligned} \mathbf{F}_i &\approx \mathbb{E}\left[\mathbf{g}_i \mathbf{g}_i^{\mathrm{T}}\right] \otimes \mathbb{E}\left[\mathbf{a}_{i-1} \mathbf{a}_{i-1}^{\mathrm{T}}\right] \\ &= \mathbf{G}_i \otimes \mathbf{A}_{i-1} \end{aligned}$$

Second order optimizer

- k-FAC:

Fisher information matrix

$$\mathbf{F}_{\boldsymbol{\theta}} \in \mathbb{R}^{60{,}000{,}000 \times 60{,}000{,}000} \qquad \square^{-1}$$

Fisher block

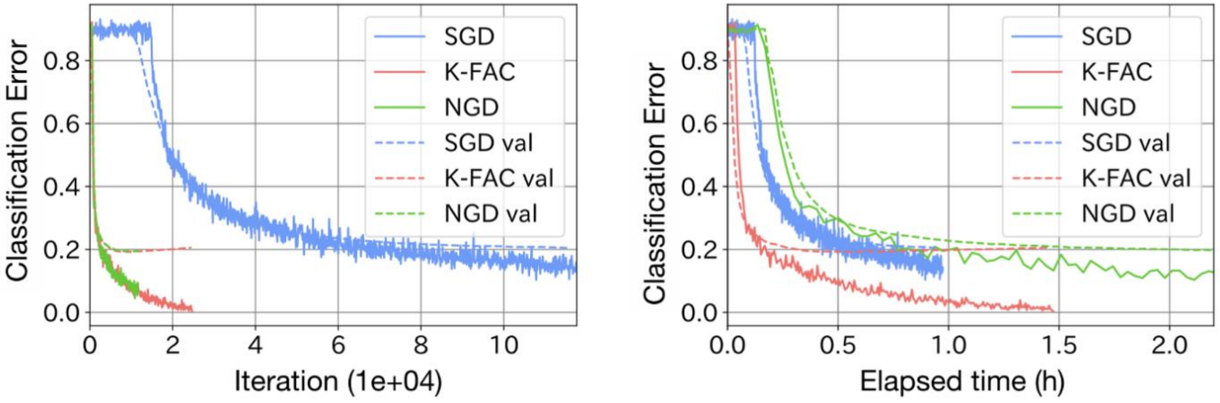
$$\mathbf{F}_i \in \mathbb{R}^{4{,}096{,}000 \times 4{,}096{,}000} \qquad \square^{-1}$$

Kronecker factors

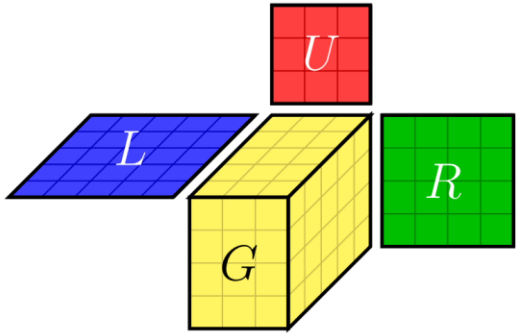
$$\mathbf{A}_{i-1} \in \mathbb{R}^{4{,}096 \times 4{,}096} \qquad \square^{-1} \otimes \square^{-1}$$

$$\mathbf{G}_i \in \mathbb{R}^{1{,}000 \times 1{,}000}$$

- Performance



Second order optimizer



- Shampoo: quasi-newton optimizer for matrix (left) and tensor (right)

Initialize $W_1 = \mathbf{0}_{m \times n}$; $L_0 = \epsilon I_m$; $R_0 = \epsilon I_n$
for $t = 1, \ldots, T$ **do**
  Receive loss function $f_t : \mathbb{R}^{m \times n} \mapsto \mathbb{R}$
  Compute gradient $G_t = \nabla f_t(W_t)$ $\{G_t \in \mathbb{R}^{m \times n}\}$
  Update preconditioners:

$$L_t = L_{t-1} + G_t G_t^{\mathsf{T}}$$

$$R_t = R_{t-1} + G_t^{\mathsf{T}} G_t$$

  Update parameters:

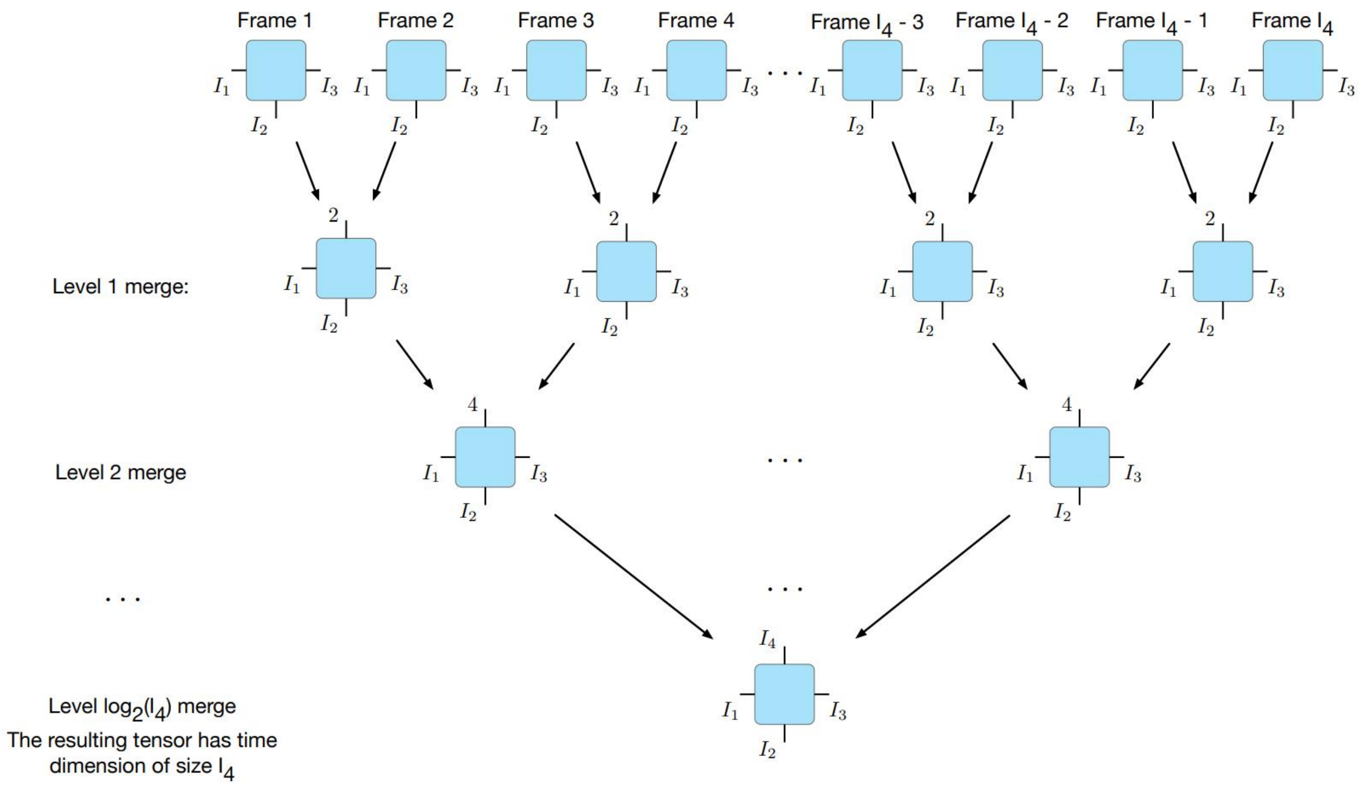
$$W_{t+1} = W_t - \eta L_t^{-1/4} G_t R_t^{-1/4}$$

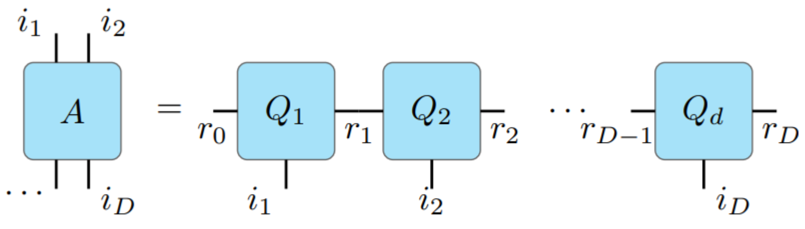
Initialize: $W_1 = \mathbf{0}_{n_1 \times \cdots \times n_k}$; $\forall i \in [k] : H_0^i = \epsilon I_{n_i}$
for $t = 1, \ldots, T$ **do**
  Receive loss function $f_t : \mathbb{R}^{n_1 \times \cdots \times n_k} \mapsto \mathbb{R}$
  Compute gradient $G_t = \nabla f_t(W_t)$ $\{G_t \in \mathbb{R}^{n_1 \times \cdots \times n_k}\}$
  $\widetilde{G}_t \leftarrow G_t$ $\{\widetilde{G}_t$ is preconditioned gradient$\}$
  for $i = 1, \ldots, k$ **do**
    $H_t^i = H_{t-1}^i + G_t^{(i)}$
    $\widetilde{G}_t \leftarrow \widetilde{G}_t \times_i (H_t^i)^{-1/2k}$
  Update: $W_{t+1} = W_t - \eta \widetilde{G}_t$

- The inversion of Lt using Schur-Newton method with cache

Data compression with tensor train format

- A multi-level tree like tensor train decomposition

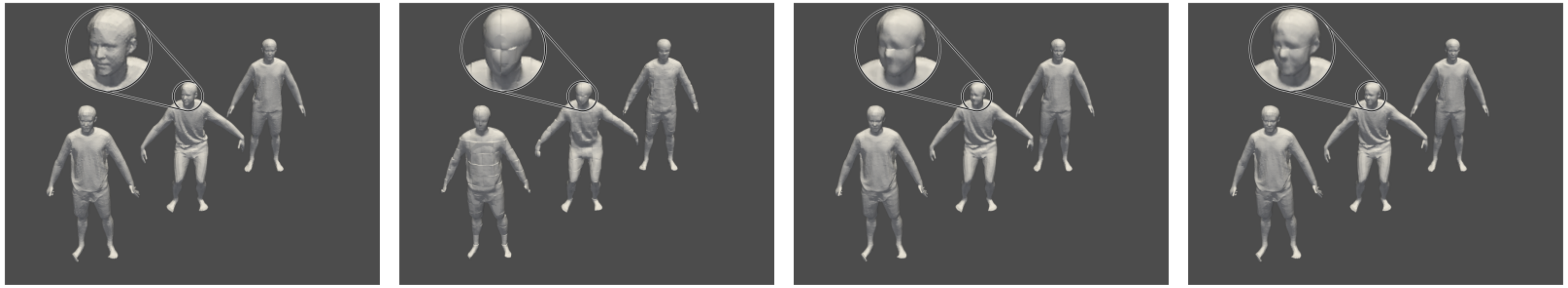




(b) TT

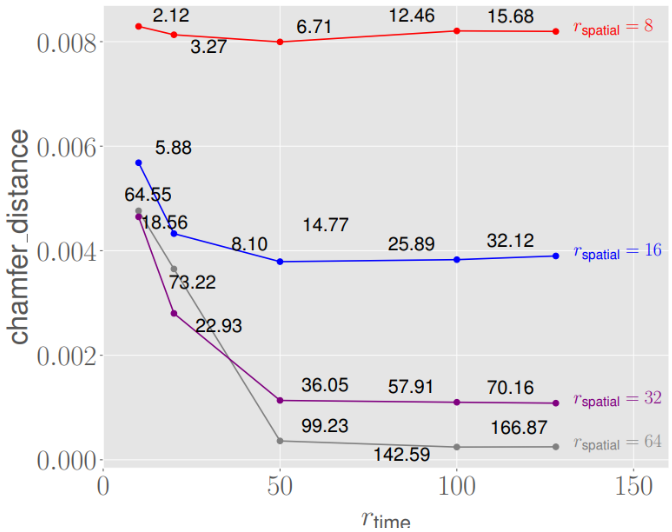
Data compression with tensor train

- Performance



(a) Original, 284 GB (b) 1:12345, 24 MB (c) 1:793, 368 MB (d) 1:227, 1.24 GB

Figure 1: T4DT with different compression levels in OQTT format for a *longshort-flying-eagle* scene of resolution 512^3 with 284 time frames. Only frames 1, 142, and 284 are depicted. The compression ratio is different from the actual memory consumption due to the padding of the time dimension to 512. High compression is achieved with $r_{\max} = 400$, MSDM2 $= 0.45$ in Fig. 1b, medium compression with $r_{\max} = 1800$, MSDM2 $= 0.32$ in Fig. 1c, and low compression / high quality with $r_{\max} = 4000$, MSDM2 $= 0.29$ in Fig. 1d.



- Metric: Chamfer distance

$$d_{\mathrm{CD}}(\mathbf{A}, \mathbf{B}) = \sum_{a \in \mathbf{A}} \min_{b \in \mathbf{B}} ||x - y||^2 + \sum_{b \in \mathbf{B}} \min_{a \in \mathbf{A}} ||x - y||^2.$$

Summary

- Tensor decomposition is a basic data compression technique and function representer
- Pros
- Mature algorithms and strong interpretability
- Promising (time/memory) efficiency for high dimensional problems

- Cons
- Linear decomposition without prior

- Future works
- Better utilize tensor decomposition as tools in machine learning
- Combination of neural based methods and tensor based methods