Subquadratic Kronecker Regression with Applications to Tensor Decomposition

Matthew Fahrbach[2] Gang Fu[2] **Mehrdad Ghadiri**[1]

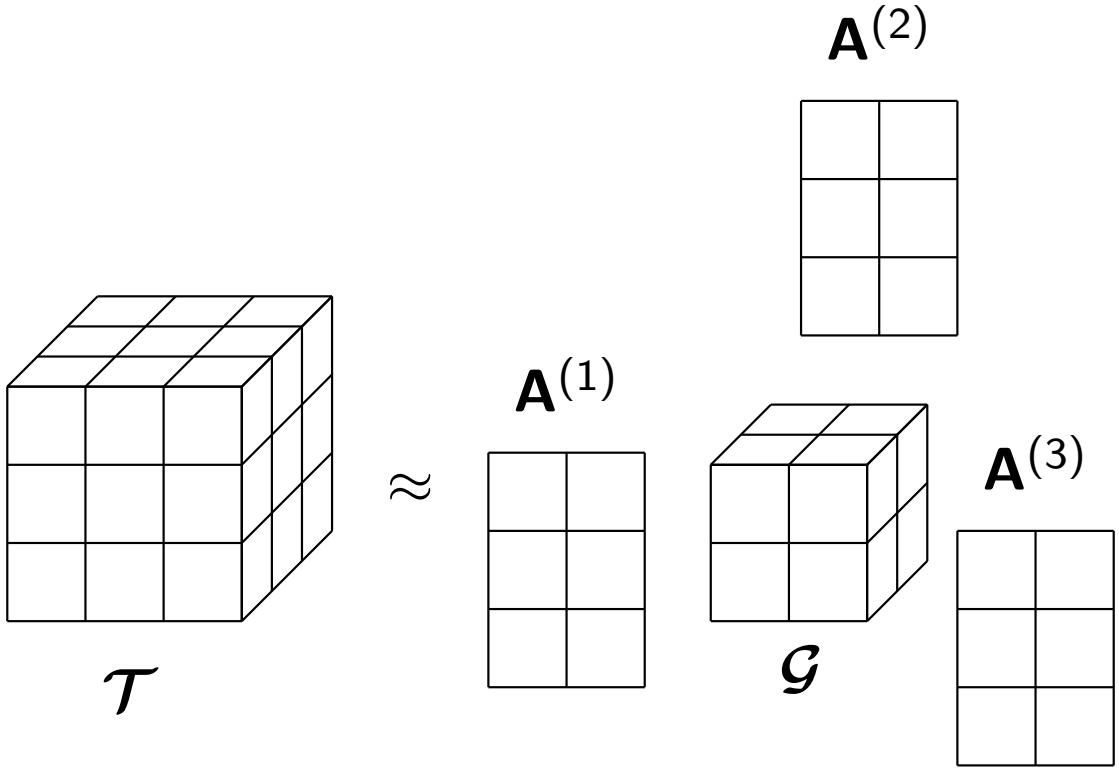
[1]Georgia Institute of Technology

[2]Google Research

Authors are listed alphabetically

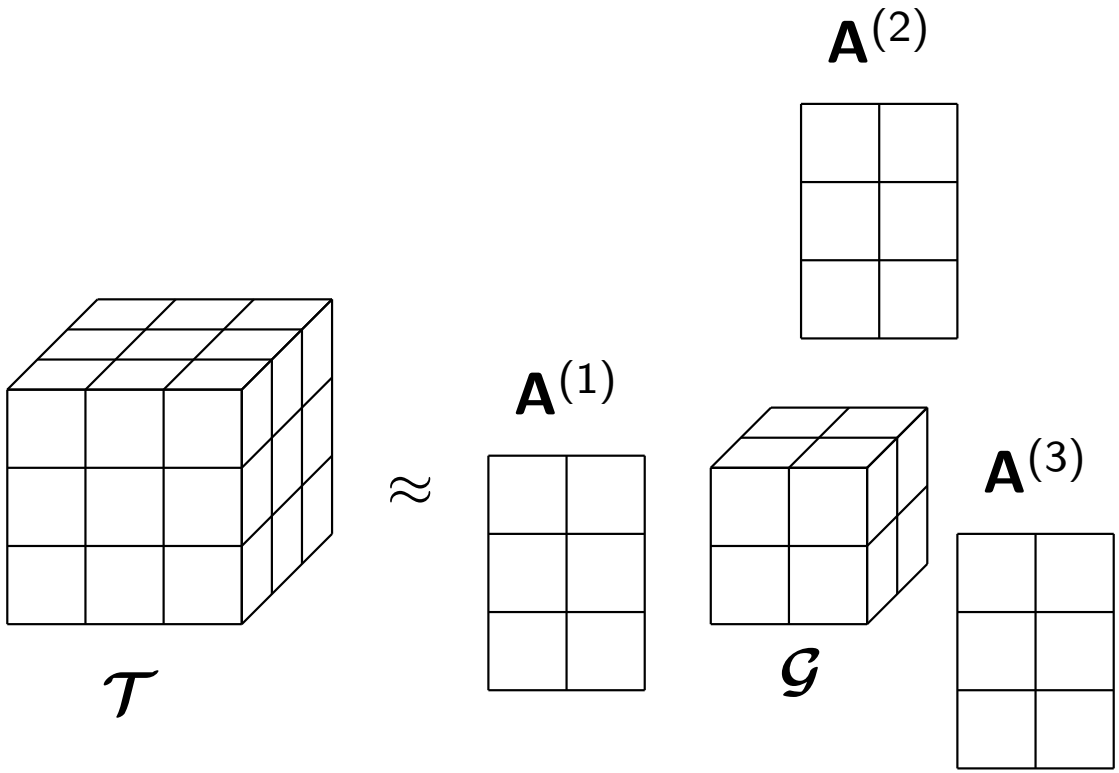
NeurIPS 2022

Our Problem: Tucker Decomposition



Tucker decomposition writes a tensor $\mathcal{T} \in \mathbb{R}^{I_1 \times \cdots \times I_N}$ of order N as product of N **factor matrices**, $\mathbf{A}^{(n)} \in \mathbb{R}^{I_n \times R_n}$ for $n \in [N]$, and a **core tensor** $\mathcal{G} \in \mathbb{R}^{R_1 \times \cdots \times R_N}$.

Our Problem: Tucker Decomposition



Tucker decomposition writes a tensor $\mathcal{T} \in \mathbb{R}^{I_1 \times \cdots \times I_N}$ of order N as product of N **factor matrices**, $\mathbf{A}^{(n)} \in \mathbb{R}^{I_n \times R_n}$ for $n \in [N]$, and a **core tensor** $\mathcal{G} \in \mathbb{R}^{R_1 \times \cdots \times R_N}$.

$$t_{i_1 i_2 \ldots i_N} \approx \widehat{t}_{i_1 i_2 \ldots i_N} \stackrel{\text{def}}{=} \sum_{r_1=1}^{R_1} \cdots \sum_{r_N=1}^{R_N} g_{r_1 r_2 \ldots r_N} a^{(1)}_{i_1 r_1} \cdots a^{(N)}_{i_N r_N}$$

Application: Faster Alternating Least Squares (ALS)

Tucker decomposition goal: $\min \|\mathcal{T} - \hat{\mathcal{T}}\|_F^2$

Application: Faster Alternating Least Squares (ALS)

Tucker decomposition goal: $\min \|\mathcal{T} - \hat{\mathcal{T}}\|_{\mathrm{F}}^2$

$\|\mathcal{T} - \hat{\mathcal{T}}\|_{\mathrm{F}}^2 = \sum_{i_1=1}^{I_1} \cdots \sum_{i_N=1}^{I_N} (t_{i_1 i_2 \ldots i_N} - \widehat{t}_{i_1 i_2 \ldots i_N})^2$

Application: Faster Alternating Least Squares (ALS)

Tucker decomposition goal: $\min \|\mathcal{T} - \hat{\mathcal{T}}\|_F^2$

ALS algorithm:

1. Fix $\mathbf{A}^{(1)}, \ldots, \mathbf{A}^{(N)}$ and optimize $\mathcal{G}$.
   - $\min_{\mathbf{g}} \|(\mathbf{A}^{(1)} \otimes \mathbf{A}^{(2)} \otimes \cdots \otimes \mathbf{A}^{(N)})\mathbf{g} - \text{vec}(\mathcal{T})\|_2^2$

Application: Faster Alternating Least Squares (ALS)

Tucker decomposition goal: $\min \|\mathcal{T} - \hat{\mathcal{T}}\|_F^2$

ALS algorithm:

1. Fix $\mathbf{A}^{(1)}, \ldots, \mathbf{A}^{(N)}$ and optimize $\mathcal{G}$.
   - $\min_{\mathbf{g}} \|(\mathbf{A}^{(1)} \otimes \mathbf{A}^{(2)} \otimes \cdots \otimes \mathbf{A}^{(N)})\mathbf{g} - \text{vec}(\mathcal{T})\|_2^2$

Kronecker product:

$$\begin{bmatrix} 1 & 2 \\ 6 & 1 \end{bmatrix} \otimes \begin{bmatrix} 4 & 3 & 1 \\ 5 & 2 & 3 \end{bmatrix} = \begin{bmatrix} 4 & 3 & 1 & & & \\ 5 & 2 & 3 & & & \\ & & & & & \\ & & & & & \end{bmatrix}$$

Application: Faster Alternating Least Squares (ALS)

Tucker decomposition goal: $\min \|\mathcal{T} - \hat{\mathcal{T}}\|_F^2$

ALS algorithm:

1. Fix $\mathbf{A}^{(1)}, \ldots, \mathbf{A}^{(N)}$ and optimize $\mathcal{G}$.
   - $\min_{\mathbf{g}} \|(\mathbf{A}^{(1)} \otimes \mathbf{A}^{(2)} \otimes \cdots \otimes \mathbf{A}^{(N)})\mathbf{g} - \text{vec}(\mathcal{T})\|_2^2$

Kronecker product:

1	2
6	1

$\otimes$

4	3	1
5	2	3

$=$

4	3	1	8	6	2
5	2	3	10	4	6

Application: Faster Alternating Least Squares (ALS)

Tucker decomposition goal: $\min \|\mathcal{T} - \hat{\mathcal{T}}\|_F^2$

ALS algorithm:

1. Fix $\mathbf{A}^{(1)}, \ldots, \mathbf{A}^{(N)}$ and optimize $\mathcal{G}$.
   - $\min_{\mathbf{g}} \|(\mathbf{A}^{(1)} \otimes \mathbf{A}^{(2)} \otimes \cdots \otimes \mathbf{A}^{(N)})\mathbf{g} - \text{vec}(\mathcal{T})\|_2^2$

Kronecker product:

1	2
6	1

$\otimes$

4	3	1
5	2	3

$=$

4	3	1	8	6	2
5	2	3	10	4	6
24	18	6			
30	12	18			

Application: Faster Alternating Least Squares (ALS)

Tucker decomposition goal: $\min \|\mathcal{T} - \hat{\mathcal{T}}\|_F^2$

ALS algorithm:

1. Fix $\mathbf{A}^{(1)}, \ldots, \mathbf{A}^{(N)}$ and optimize $\mathcal{G}$.
   - $\min_{\mathbf{g}} \|(\mathbf{A}^{(1)} \otimes \mathbf{A}^{(2)} \otimes \cdots \otimes \mathbf{A}^{(N)})\mathbf{g} - \text{vec}(\mathcal{T})\|_2^2$

Kronecker product:

$$\begin{bmatrix} 1 & 2 \\ 6 & 1 \end{bmatrix} \otimes \begin{bmatrix} 4 & 3 & 1 \\ 5 & 2 & 3 \end{bmatrix} = \begin{bmatrix} 4 & 3 & 1 & 8 & 6 & 2 \\ 5 & 2 & 3 & 10 & 4 & 6 \\ 24 & 18 & 6 & 4 & 3 & 1 \\ 30 & 12 & 18 & 5 & 2 & 3 \end{bmatrix}$$

Application: Faster Alternating Least Squares (ALS)

Tucker decomposition goal: $\min \|\mathcal{T} - \hat{\mathcal{T}}\|_F^2$

ALS algorithm:

1. Fix $\mathbf{A}^{(1)}, \ldots, \mathbf{A}^{(N)}$ and optimize $\mathcal{G}$.
   - $\min_{\mathbf{g}} \|(\mathbf{A}^{(1)} \otimes \mathbf{A}^{(2)} \otimes \cdots \otimes \mathbf{A}^{(N)})\mathbf{g} - \text{vec}(\mathcal{T})\|_2^2$
2. Fix $\mathbf{A}^{(1)}, \ldots, \mathbf{A}^{(i-1)}, \mathbf{A}^{(i+1)}, \ldots \mathbf{A}^{(N)}$, and $\mathcal{G}$; Optimize $\mathbf{A}^{(i)}$.
   - For each column j of $\mathbf{A}^{(i)}$ solve

     $\min_{\mathbf{y}} \|(\mathbf{A}^{(1)} \otimes \cdots \otimes \mathbf{A}^{(i-1)} \otimes \mathbf{A}^{(i+1)} \otimes \cdots \otimes \mathbf{A}^{(N)})\mathbf{G}_{(n)}^\top \mathbf{y} - (\mathbf{T}_{(n)}^\top)_{:j}\|_2^2$

Kronecker Product Regression

$$\arg\min_{\mathbf{x}} \|(\mathbf{A}^{(1)} \otimes \mathbf{A}^{(2)} \otimes \cdots \otimes \mathbf{A}^{(N)})\mathbf{x} - \mathbf{b}\|_2^2$$

Kronecker Product Regression

$$\arg\min_{\mathbf{x}} \|(\mathbf{A}^{(1)} \otimes \mathbf{A}^{(2)} \otimes \cdots \otimes \mathbf{A}^{(N)})\mathbf{x} - \mathbf{b}\|_2^2$$

Number of columns of $\mathbf{A}^{(i)}$: R_i

$R \stackrel{\text{def}}{=} R_1 R_2 \cdots R_N$

Goal: $(1+\epsilon)$-approximation.

Kronecker Product Regression

$$\arg\min_{\mathbf{x}} \|(\mathbf{A}^{(1)} \otimes \mathbf{A}^{(2)} \otimes \cdots \otimes \mathbf{A}^{(N)})\mathbf{x} - \mathbf{b}\|_2^2$$

Number of columns of $\mathbf{A}^{(i)}$: R_i

$R \stackrel{\text{def}}{=} R_1 R_2 \cdots R_N$

Goal: $(1+\epsilon)$-approximation.

Previous work: Diao-Jayaram-Song-Sun-Woodruff (NeurIPS'19):

Kronecker Product Regression

$$\arg\min_{\mathbf{x}} \|(\mathbf{A}^{(1)} \otimes \mathbf{A}^{(2)} \otimes \cdots \otimes \mathbf{A}^{(N)})\mathbf{x} - \mathbf{b}\|_2^2$$

Number of columns of $\mathbf{A}^{(i)}$: R_i

$R \stackrel{\text{def}}{=} R_1 R_2 \cdots R_N$

Goal: $(1+\epsilon)$-approximation.

Previous work: Diao-Jayaram-Song-Sun-Woodruff (NeurIPS'19):

- $\tilde{O}(\sum_{i=1}^{N}(\text{nnz}(\mathbf{A}^{(i)}) + R_i^{\omega}) + R^{\omega}\epsilon^{-1})$ by sampling $\tilde{O}(R\varepsilon^{-1})$ rows from the Kronecker product matrix.
- $\tilde{O}(\sum_{i=1}^{N}(\text{nnz}(\mathbf{A}^{(i)}) + R_i^{\omega}\epsilon^{-1}) + R\epsilon^{-N})$ by sampling $\tilde{O}(R_i\epsilon^{-1})$ rows from each factor matrix $\mathbf{A}^{(n)}$.

Kronecker Product Regression

$$\arg\min_{\mathbf{x}} \|(\mathbf{A}^{(1)} \otimes \mathbf{A}^{(2)} \otimes \cdots \otimes \mathbf{A}^{(N)})\mathbf{x} - \mathbf{b}\|_2^2$$

Number of columns of $\mathbf{A}^{(i)}$: R_i

$R \stackrel{\text{def}}{=} R_1 R_2 \cdots R_N$

Goal: $(1+\epsilon)$-approximation.

Previous work: Diao-Jayaram-Song-Sun-Woodruff (NeurIPS'19):

- $\tilde{O}(\sum_{i=1}^{N}(\text{nnz}(\mathbf{A}^{(i)}) + R_i^{\omega}) + R^{\omega}\epsilon^{-1})$ by sampling $\tilde{O}(R\varepsilon^{-1})$ rows from the Kronecker product matrix.
- $\tilde{O}(\sum_{i=1}^{N}(\text{nnz}(\mathbf{A}^{(i)}) + R_i^{\omega}\epsilon^{-1}) + R\epsilon^{-N})$ by sampling $\tilde{O}(R_i\epsilon^{-1})$ rows from each factor matrix $\mathbf{A}^{(n)}$.

Our Result: Faster Kronecker Product Regression

Diao-Jayaram-Song-Sun-Woodruff (NeurIPS'19):

- $\tilde{O}(\sum_{i=1}^{N}(\text{nnz}(\mathbf{A}^{(i)}) + R_i^{\omega}) + R^{\omega}\epsilon^{-1})$ by sampling $\tilde{O}(R\varepsilon^{-1})$ rows from the Kronecker product matrix.
- $\tilde{O}(\sum_{i=1}^{N}(\text{nnz}(\mathbf{A}^{(i)}) + R_i^{\omega}\epsilon^{-1}) + R\epsilon^{-N})$ by sampling $\tilde{O}(R_i\epsilon^{-1})$ rows from each factor matrix $\mathbf{A}^{(n)}$.

Theorem (this work)

$$\tilde{O}\left(\sum_{i=1}^{N}\left(\text{nnz}(\mathbf{A}^{(i)}) + R_i^{\omega}N^2\epsilon^{-2}\right) + \min_{S\subseteq[N]} \text{MM}\left(\prod_{i\in S} R_i, R\varepsilon^{-1}, \prod_{i\in[N]\setminus S} R_i\right)\right)$$

$\text{MM}(a, b, c)$ is the running time of multiplying an $a \times b$ matrix with a $b \times c$ matrix. $\text{MM}(a, a, a) = a^{\omega}$.

Our Result: Faster Kronecker Product Regression

Diao-Jayaram-Song-Sun-Woodruff (NeurIPS'19):

- $\tilde{O}(\sum_{i=1}^{N}(\text{nnz}(\mathbf{A}^{(i)}) + R_i^{\omega}) + R^{\omega}\epsilon^{-1})$ by sampling $\tilde{O}(R\varepsilon^{-1})$ rows from the Kronecker product matrix.
- $\tilde{O}(\sum_{i=1}^{N}(\text{nnz}(\mathbf{A}^{(i)}) + R_i^{\omega}\epsilon^{-1}) + R\epsilon^{-N})$ by sampling $\tilde{O}(R_i\epsilon^{-1})$ rows from each factor matrix $\mathbf{A}^{(n)}$.

Theorem (this work)

$$\tilde{O}\left(\sum_{i=1}^{N}\left(\text{nnz}(\mathbf{A}^{(i)}) + R_i^{\omega}N^2\epsilon^{-2}\right) + \min_{S\subseteq[N]} \text{MM}\left(\prod_{i\in S} R_i, R\varepsilon^{-1}, \prod_{i\in[N]\setminus S} R_i\right)\right)$$

Improvements:

- Practical: $O(R^3) \to O(R^2)$
- Theoretical: $O(R^{2.372}) \to O(R^{1.626})$

Our Ideas and Techniques

Key ideas in our algorithm:

- Use input Kronecker matrix as **preconditioner** for Richardson iterations $\rightarrow$ exploit fast Kronecker matrix-vector products
- **Faster** multiplication for **sparsified Kronecker products**

Our Ideas and Techniques

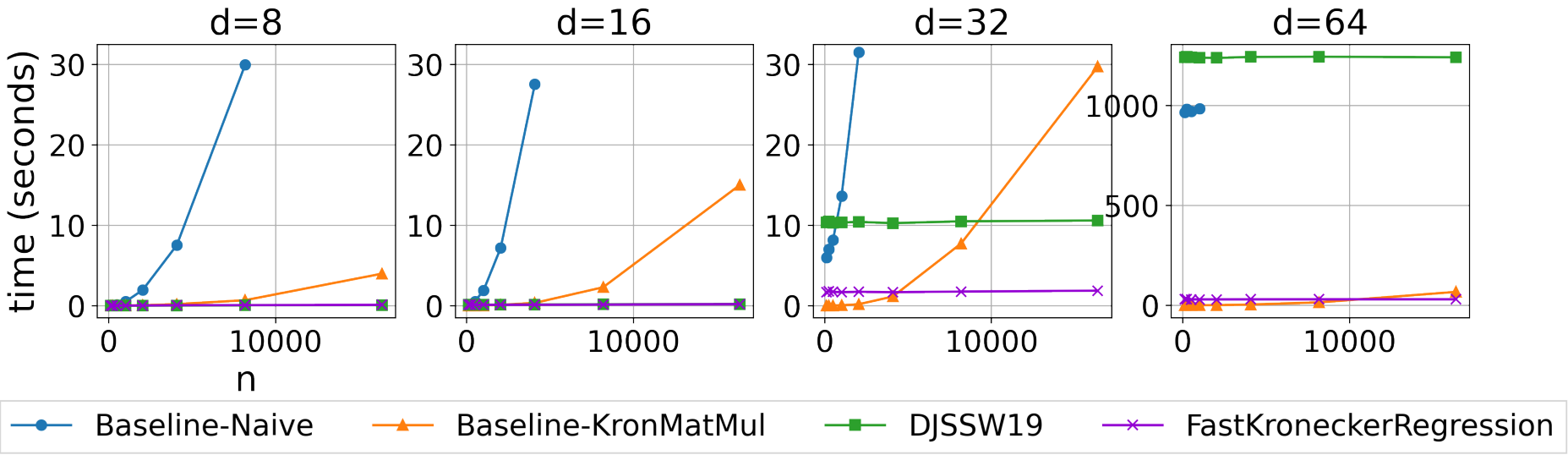
Key ideas in our algorithm:

- Use input Kronecker matrix as **preconditioner** for Richardson iterations $\rightarrow$ exploit fast Kronecker matrix-vector products
- **Faster** multiplication for **sparsified Kronecker products**

Extension to matrices with a Kronecker product block to solve **ridge regression** and **factor matrix updates**:

- Block leverage score sampling
- SVD of the inverse
- Low-rank perturbation of inverse (Sherman–Morrison-Woodbury identity)

Experiments



Running times of Kronecker regression algorithms with an order 2 design matrix of size $n^2 \times d^2$.

Experiments

Kronecker regression losses for $d = 64$

- OPT uses all rows
- DJSSW19 is Diao-Jayaram-Song-Sun-Woodruff (NeurIPS'19)
- Approx is the relative error

n	OPT	**Ours**	Approx	DJSSW19	Approx	% Rows sampled
1024	0.031	0.032	1.051	0.035	1.138	0.0370
2048	0.123	0.126	1.026	1.577	12.792	0.0093
4096	0.507	0.520	1.026	275.566	543.776	0.0023
8192	2.073	2.136	1.030	333.430	160.809	0.0006
16384	8.238	8.608	1.045	546391.728	66329.791	0.0001