SUPPLEMENTARY MATERIALS: LOW-RANK TUCKER APPROXIMATION OF A TENSOR FROM STREAMING DATA*

YIMING SUN†, YANG GUO‡, CHARLENE LUO§, JOEL TROPP¶, AND MADELEINE UDELL†

SM1. Time and Storage Complexity.

SM1.1. Comparison Between Algorithm 4.4 and T.-TS [8]. Here we compare the time and storage complexity of the two extant methods for streaming Tucker approximation: our one-pass method, and T.-TS [8].

To compare the storage and time costs of both T.-TS and the one-pass algorithm, we separate the cost into two parts: one for forming the sketch, the other for each iteration of ALS. Assume the tensor to approximate has equal side lengths $I_1 = \cdots = I_N = I$ and that the target rank for each mode is R.

The suggested default parameters for the sketch in [8] are $J_1 = 10R^{N-1}$ and $J_2 = 10R^N$. Our suggested default parameters are $k = 2r, s = 2k + 1$. Under the choice of the default parameter, we compare the the cost of storage and time in Table SM1 and Table SM2. In most problems with data that is not exactly low rank, i.e. $R > 4$, the suggested default setting of T.-TS typically leads to a higher storage cost. Moreover, our algorithm uses less storage and is faster to compute, particularly for tensors with many modes N.

However, the evaluation of the two algorithms should not be solely based on their default setups. If the memory constraint is set to be the same, our one-pass algorithm performs much better in the low-memory case, but slightly worse in the high-memory case (see Figure 3). The memory required by our default parameters is typically much smaller than that required with the default parameters of [8].

SM1.2. Computational Complexity of Algorithm 4.4. Here, we will calculate the computational complexity for our one-pass fixed-rank approximation algorithm.

In the sketching stage of the streaming algorithm, we first need to compute the factor sketches, $\mathbf{G}_n = \mathbf{X}\mathbf{\Omega}_n, n \in [N]$ with $kN\hat{I}$ flops in total. Then we need to compute the core tensor sketch $\mathcal{Z}$ by recursively multiplying $\mathcal{X}$ by $\mathbf{\Phi}_n, n \in [N]$. We can upper bound the number of flops by $\frac{s(1-\delta_1^N)}{1-\delta_1}\bar{I}$. Then in the approximation stage, we first perform "economy size" QR factorizations of $\mathbf{G}_1, \ldots, \mathbf{G}_N$ with $\mathcal{O}(k^2(\sum_{n=1}^N I_n))$ to find the orthonormal bases $\mathbf{Q}_1, \ldots, \mathbf{Q}_N$. To find the linkage tensor $\mathcal{W}$, we need to recursively solve linear square problems with $\frac{k^2 s^N (1-(k/s)^N)}{1-k/s}$ flops. Overall, the sketch computation dominates the total time complexity.

The HOSVD directly acts on $\mathcal{X}$ by first computing the SVD for each unfolding $(\mathcal{O}(kN\bar{I}))$ and then multiplying $\mathcal{X}$ by $\mathbf{U}_1^\top, \ldots, \mathbf{U}_N^\top$ $(\mathcal{O}(\frac{k(1-\delta_1^N)\bar{I}}{1-\delta_1}))$. The total time cost is less than the streaming algorithm with a constant factor. Note: we can use the randomized SVD in the first step of the HOSVD to improve the computational cost to $\bar{I}N \log k + \sum_{n=1}^N (I_n + I_{(-n)})k^2$ [7].

---

*Supplementary materials for SIMODS MS#M125771.

†Cornell University, Ithaca, NY 14850 (ys784@cornell.edu, udell@cornell.edu).

‡Computer Science, University of Wisconsin-Madison, Madison, WI 53711 (yguo@cs.wisc.edu).

§Columbia University, New York, NY 10027 (cl3788@columbia.edu).

¶Computing + Mathematical Sciences, California Institute of Technology, Pasadena, CA 91125-5000 (jtropp@cms.caltech.edu).

Algorithm		Storage Cost ($I = o(r^{2N})$)
T.-TS	Sketching	$\mathcal{O}(r^{2N})$
	Recovery	$\mathcal{O}(r^{2N})$
Algorithm 4.3 (One Pass)	Sketching	$\mathcal{O}(4^N r^N)$
	Recovery	$\mathcal{O}(4^N r^N)$

Table SM1: Storage complexity of Algorithm 4.3 and T.-TS on tensor $\mathcal{X} \in \mathbb{R}^{I \times \cdots \times I}$. Algorithm 4.3 uses parameters $(k, s) = (2r, 4r + 1)$ and uses a TRP composed of Gaussian DRMs inside the Tucker sketch. T.-TS uses default values for hyper-parameters: $J_1 = 10r^{N-1}, J_2 = 10r^N$.

Algorithm		Time Cost ($I = o(r^{2N})$)
T.-TS	Sketching	$\mathcal{O}(N\text{nnz}(\mathcal{X}))$
	Recovery	$\mathcal{O}(NIr^N + Nr^{2N-1} + r^{2N})$
Algorithm 4.3 (One Pass)	Sketching	$\mathcal{O}(Nr\ \text{nnz}(\mathcal{X})))$
	Recovery	$\mathcal{O}(Nr^{N+1})$

Table SM2: Time complexity of Algorithm 4.3 and T.-TS on tensor $\mathcal{X} \in \mathbb{R}^{I \times \cdots \times I}$. Algorithm 4.3 uses parameters $(k, s) = (2r, 4r + 1)$ and uses a TRP composed of Gaussian DRMs inside the Tucker sketch. T.-TS uses default values for hyper-parameters: $J_1 = 10r^{N-1}, J_2 = 10r^N$.

SM2. More Numerics. This section provides more numerical results on simulated datasets in Figure SM1, Figure SM2, Figure SM3, and Figure SM4.

We also provide more numerical results on real datasets in Figure SM5.

SM3. More Algorithms. This section provides detailed implementations.

Algorithm SM3.1 Higher order orthogonal iteration (HOOI) [5]

Given: tensor $\mathcal{X}$, target rank $\mathbf{r} = (r_1, \ldots, r_N)$
Initialize: compute $\mathcal{X} \approx [\![\mathcal{G}; \mathbf{U}_1, \ldots, \mathbf{U}_N]\!]$ using HOSVD
Repeat:

1. *Factors.* For each $n \in [N]$,

$$\mathbf{U}_n \leftarrow \arg\min_{\mathbf{U}_n} \|[\![\mathcal{G}; \mathbf{U}_1, \ldots, \mathbf{U}_N]\!] - \mathcal{X}\|_F^2, \tag{SM3.1}$$

2. *Core.*

$$\mathcal{G} \leftarrow \arg\min_{\mathcal{G}} \|[\![\mathcal{G}; \mathbf{U}_1, \ldots, \mathbf{U}_N]\!] - \mathcal{X}\|_F^2. \tag{SM3.2}$$
$$i.e.\ \ \mathcal{G} = \mathcal{X} \times_1 \mathbf{U}_1^\top \times_2 \cdots \times_N \mathbf{U}_N^\top$$

Return: Tucker approximation $\mathcal{X}_{\text{HOOI}} = [\![\mathcal{G}; \mathbf{U}_1, \ldots, \mathbf{U}_N]\!]$

Notice the core update (SM3.2) admits the closed form solution $\mathcal{G} \leftarrow \mathcal{X} \times_1 \mathbf{U}_1^\top \cdots \times_N \mathbf{U}_N^\top$, which motivates the second step of HOSVD for a linear sketch appropriate to a streaming setting (Algorithm SM3.2) or a distributed setting (Algorithm SM3.3).

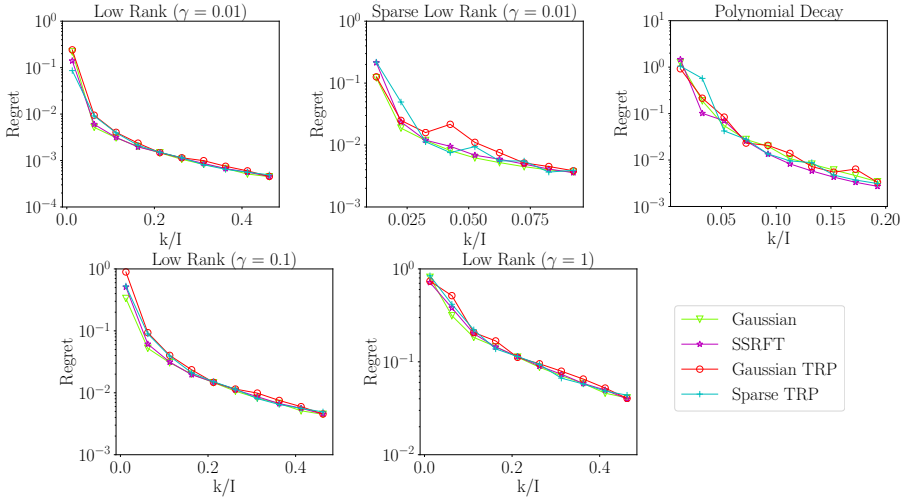

Fig. SM1: We approximate 3D synthetic tensors (see subsection 6.3) with $I = 400$, using our one-pass algorithm with $r = 5$ and varying k ($s = 2k + 1$), using a variety of DRMs in the Tucker sketch: Gaussian, SSRFT, Gaussian TRP, or Sparse TRP.

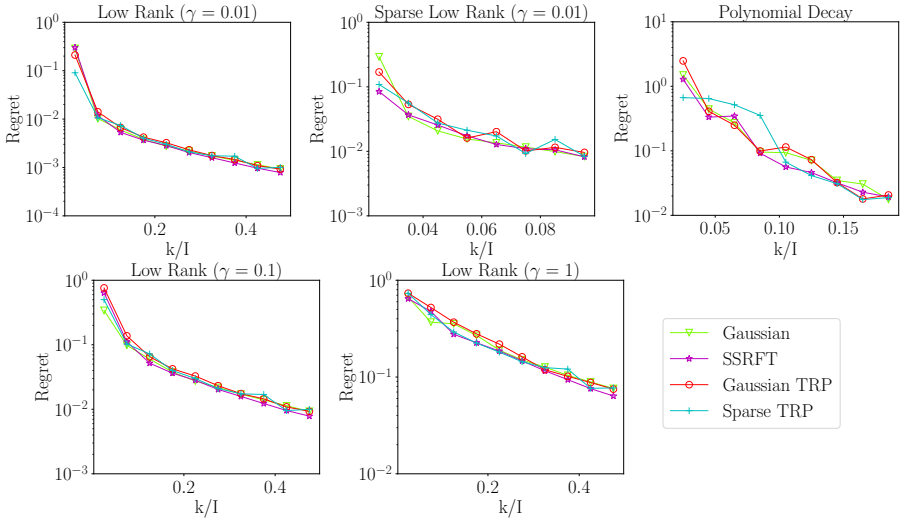

Fig. SM2: We approximate 3D synthetic tensors (see subsection 6.3) with $I = 200$, using our one-pass algorithm with $r = 5$ and varying k ($s = 2k + 1$), using a variety of DRMs in the Tucker sketch: Gaussian, SSRFT, Gaussian TRP, or Sparse TRP.

SM4. Scrambled Subsampled Randomized Fourier Transform. In order to reduce the cost of storing the test matrices, in particular, $\mathbf{\Omega}_1, \ldots, \mathbf{\Omega}_N$, we can use the Scrambled Subsampled Randomized Fourier Transform (SSRFT). To reduce the dimension of a matrix, $\mathbf{X} \in \mathbb{R}^{m \times n}$, along either the row or the column to size k, we

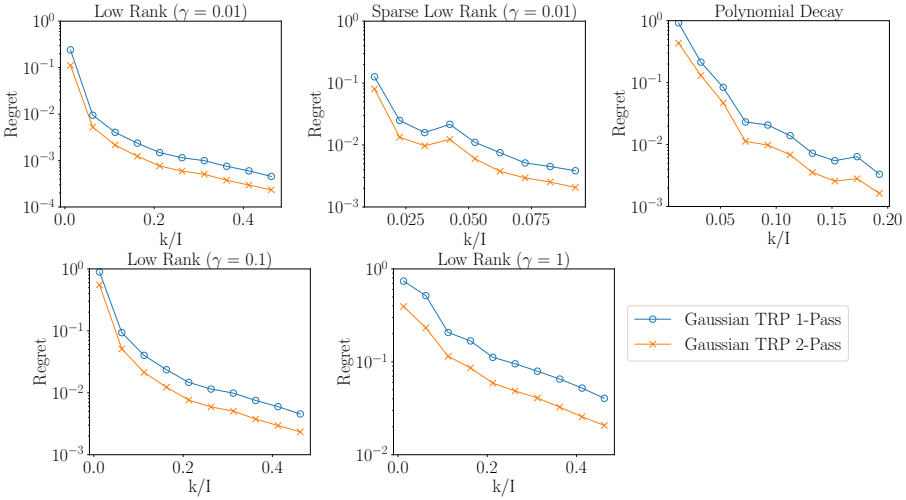

Fig. SM3: We approximate 3D synthetic tensors (see subsection 6.3) with $I = 400$, using our one-pass and two-pass algorithms with $r = 5$ and varying k ($s = 2k + 1$), using the Gaussian TRP in the Tucker sketch.

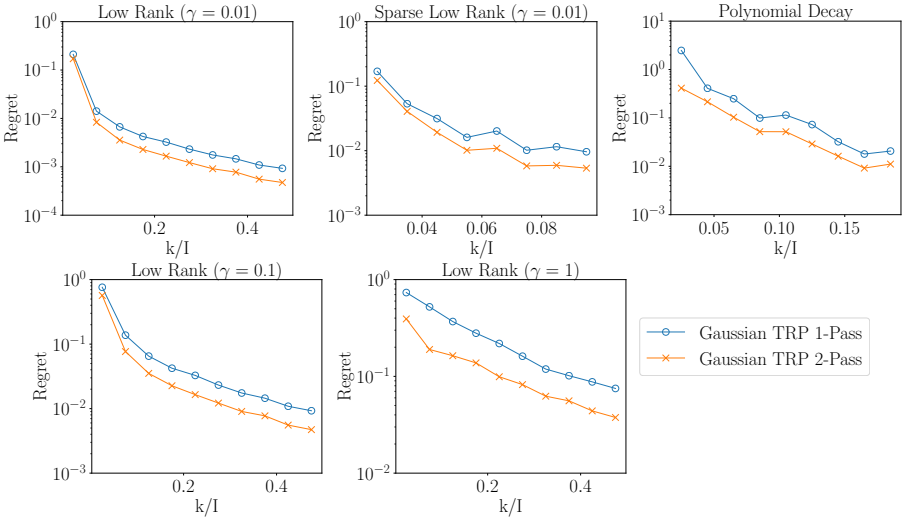

Fig. SM4: We approximate 3D synthetic tensors (see subsection 6.3) with $I = 200$, using our one-pass and two-pass algorithms with $r = 5$ and varying k ($s = 2k + 1$), using the Gaussian TRP in the Tucker sketch.

define the SSRFT map $\boldsymbol{\Xi}$ as:

$$\boldsymbol{\Xi} = \begin{cases} \mathbf{R}\mathbf{F}^\top \boldsymbol{\Pi}\mathbf{F}\boldsymbol{\Pi}^\top \in \mathbb{F}^{k\times m} & \text{(Row linear transform)} \\ (\bar{\mathbf{R}}\bar{\mathbf{F}}^\top \bar{\boldsymbol{\Pi}}\bar{\mathbf{F}}\bar{\boldsymbol{\Pi}}^\top)^\top \in \mathbb{F}^{n\times k} & \text{(Column linear transform)}, \end{cases}$$

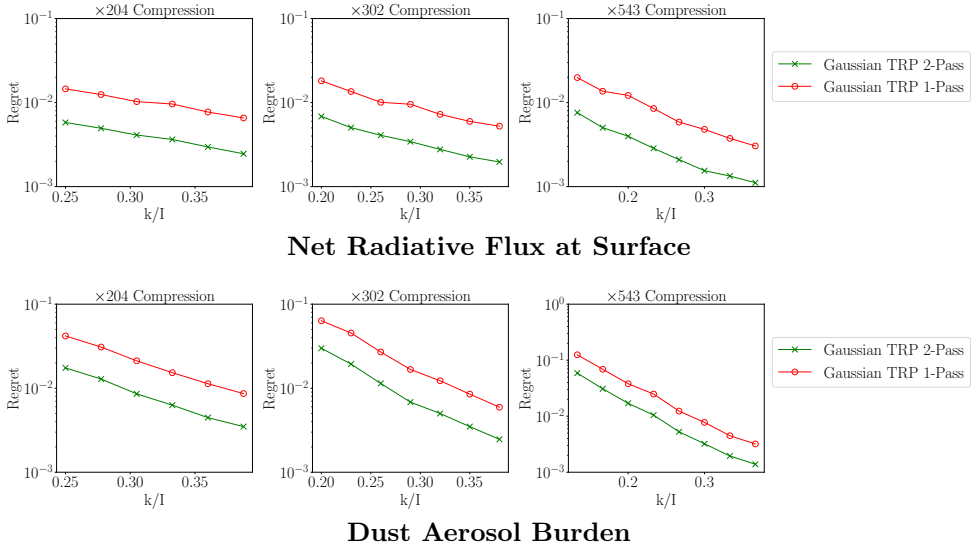

Fig. SM5: We approximate the net radiative flux and dust aerosol burden data using our one-pass and two-pass algorithms using Gaussian TRP. We compare the performance under different ranks ($r/I = 0.125, 0.2, 0.067$). The dataset comes from the CESM CAM. The dust aerosol burden measures the amount of aerosol contributed by the dust. The net radiative flux determines the energy received by the earth surface through radiation.

Algorithm SM3.2 Linear Update to Sketches

function SketchLinearUpdate($\mathcal{F}, \mathbf{V}_1, \ldots, \mathbf{V}_N, \mathcal{H}; \theta_1, \theta_2$)
  for $n = 1, \ldots, N$ **do**
    $\mathbf{V}_n \leftarrow \theta_1 \mathbf{V}_n + \theta_2 \mathbf{F}^{(n)} \mathbf{\Omega}_n$
  end for
  $\mathcal{H} \leftarrow \theta_1 \mathcal{H} + \theta_2 \mathcal{F} \times_1 \mathbf{\Phi}_1 \times \cdots \times_N \mathbf{\Phi}_N$
  return $(\mathbf{V}_1, \ldots, \mathbf{V}_N, \mathcal{H})$
end function

where $\mathbf{\Pi}, \mathbf{\Pi}' \in \mathbb{R}^{m \times m}, \bar{\mathbf{\Pi}}, \bar{\mathbf{\Pi}}' \in \mathbb{R}^{n \times n}$ are signed permutation matrices. That is, the matrix has exactly one non-zero entry, 1 or -1 with equal probability, in each row and column. $\mathbf{F} \in \mathbb{F}^{m \times m}, \mathbf{F} \in \mathbb{F}^{n \times n}$ denote the discrete cosine transform ($\mathbb{F} = \mathbb{R}$) or the discrete fourier transform ($\mathbb{F} = \mathbb{C}$). The matrix $\mathbf{R}, \bar{\mathbf{R}}$ is the restriction to k coordinates chosen uniformly at random.

In practice, we implement the SSRFT as in Algorithm SM4.1. It takes only $\mathcal{O}(m)$ or $\mathcal{O}(n)$ bits to store $\mathbf{\Xi}$, compared to $\mathcal{O}(km)$ or $\mathcal{O}(kn)$ for Gaussian or uniform random map. The cost of applying $\mathbf{\Xi}$ to a vector is $\mathcal{O}(n \log n)$ or $\mathcal{O}(m \log m)$ arithmetic operations for fast Fourier transform and $\mathcal{O}(n \log k)$ or $\mathcal{O}(m \log k)$ for fast cosine transform. Though in practice, SSRFT behaves similarly to the Gaussian random map, its analysis is less comprehensive [2, 9, 1] than the Gaussian case.

SM5. TensorSketch. Many authors have developed methods to perform dimension reduction efficiently. In particular, [6] proposed a method called *TensorSketch*

Algorithm SM3.3 Sketching in Distributed Setting

Require: $\mathcal{X}_i$ is the part of the tensor $\mathcal{X}$ at local machine i and $\mathcal{X} = \sum_{i=1}^{m} \mathcal{X}_i$.

function ComputeSketchDistributed($\mathcal{X}_1, \ldots, \mathcal{X}_m$)
  Send the same random generating environment to every local machine.
  Generate the same DRM at each local machine.
  for $i = 1 \ldots m$ **do**
    $(\mathbf{V}_1^{(i)}, \cdots, \mathbf{V}_n^{(i)}, \mathcal{H}^{(i)}) \leftarrow$ ComputeSketch($\mathcal{X}_i$)
  end for
  for $j = 1 \ldots n$ **do**
    $\mathbf{V}_j \leftarrow \sum_{i=1}^{m} \mathbf{V}_j^{(i)}$
  end for
  $\mathcal{H} \leftarrow \sum_{i=1}^{m} \mathcal{H}^{(i)}$
  return $(\mathbf{V}_1, \ldots, \mathbf{V}_n, \mathcal{H})$
end function

Algorithm SM4.1 Scrambled Subsampled Randomized Fourier Transform (Row Linear Transform)

Require: $\mathbf{X} \in \mathbb{R}^{m \times n}, \mathcal{F} = \mathbb{R}$, **randperm** creates a random permutaion vector, and **randsign** creates a random sign vector. **dct** denotes the discrete cosine transform.

function SSRFT($\mathbf{X}$)
  coords $\leftarrow$ **randperm**(m,k)
  $\mathbf{perm}_j \leftarrow$ **randperm**(m) for $j = 1, 2$
  $\mathbf{sgn}_j \leftarrow$ **randsign**(m) for $j = 1, 2$
  $\mathbf{X} \leftarrow$ **dct**$(\mathbf{sgn}_1 \cdot \mathbf{X}[\mathbf{perm}_1, :])$ ▷ elementwise product
  $\mathbf{X} \leftarrow$ **dct**$(\mathbf{sgn}_2 \cdot \mathbf{X}[\mathbf{perm}_2, :])$
  return $\mathbf{X}[\mathbf{coords}, :]$
end function

that aims to solve least squares problems for which the design matrix has a Kronecker product structure. [8] use this technique to compute a one-pass Tucker decomposition. Here we review the TensorSketch and how it is used in [8].

CountSketch. [4] proposed the CountSketch method. A comprehensive theoretical analysis in the context of low-rank approximation problems appears in [3]. To compute the sketch $\mathbf{X}\boldsymbol{\Omega} \in \mathbb{R}^{d \times k}$ for $\mathbf{X} \in \mathbb{R}^{m \times d}$, CountSketch defines $\boldsymbol{\Omega} = \mathbf{D}\boldsymbol{\Phi}$, where

1. $\mathbf{D} \in \mathbb{R}^{d \times d}$ is a diagonal matrix with each diagonal entry equal to $(-1, 1)$ with probability $(1/2, 1/2)$.
2. $\boldsymbol{\Phi} \in \mathbb{R}^{d \times k}$ is the matrix form of a hash function.

These two matrices have $2d$ non-zero entries in total and thus require much less storage than the standard kd entries. Furthermore, these two matrices can operate on each column of $\mathbf{X}$ at a cost of only $\mathcal{O}(kd)$ arithmetic operations.

TensorSketch. [8] proposes to use the CountSketch inside the HOOI method for Tucker decomposition They apply the sketch to solve least squares problems appearing in (SM3.1) and (SM3.2) in Algorithm SM3.1. They use J_1, J_2 to denote the reduced dimension. Using a standard random map, it would require a J_1-by-$I_{(-n)}$ random matrix to solve the problem in (SM3.1) and a J_2-by-$\prod_{n=1}^{N} I_n$ random matrix to solve the problem in (SM3.2). However, these problems have Kronecker problem structure:

as shown in [8], these two stages can be expressed as

(SM5.1)
$$\text{For } n = 1, \ldots, N, \text{update } \mathbf{U}^{(n)} = \underset{\mathbf{U} \in \mathbb{R}^{I_n \times R_n}}{\arg\min} \left\| \left(\bigotimes_{\substack{i=N \\ i \neq n}}^{1} \mathbf{U}^{(i)} \right) \mathbf{G}_{(n)}^{\top} \mathbf{U}^{\top} - \mathbf{Y}_{(n)}^{\top} \right\|_F^2 .$$

(SM5.2)
$$\text{Update } \mathcal{G} = \underset{\mathcal{Z} \in \mathbb{R}^{R_1 \times \cdots \times R_N}}{\arg\min} \left\| \left(\bigotimes_{i=N}^{1} \mathbf{U}^{(i)} \right) \mathbf{vec}\, \mathcal{Z} - \mathbf{vec}\, \mathcal{Y} \right\|_2^2 ,$$

where $\mathcal{Y}$ is the original data. Here $\forall i \in [n], \mathbf{U}_i$ is the factor matrix, and $\mathcal{G}$ is the core tensor. The target multilinear rank is $(R_1, \ldots, R_N)$.

Following [6], [8] proposes to apply TensorSketch to the Kronecker product structure of the input matrix in the sketch construction, i.e. $\otimes_{\substack{i=1 \\ i \neq n}}^{N} \mathbf{U}_i$ in (SM5.1) and $\otimes_{i=1}^{N} \mathbf{U}_i$ in (SM5.2). The TensorSketch method combines the CountSketch of each factor matrix via the Khatri-Rao product and Fast Fourier Transform. Consider sketching $\otimes_{i=1}^{N} \mathbf{U}_i$ in (SM5.2). TensorSketch is defined as

(SM5.3)
$$\mathbf{\Omega X} = \mathrm{FFT}^{-1}\left(\odot_{n=1}^{N} \left(\mathrm{FFT}\big(\mathrm{CountSketch}^{(n)}(\mathbf{U}^{(n)})\big)^{\top} \right)^{\top} \right)$$

By only storing $\mathrm{CountSketch}^{(1)}, \ldots, \mathrm{CountSketch}^{(N)}$, TensorSketch only requires storage $2\sum_{i=1}^{N} I_n$. Therefore, the storage cost of the sketch is dominated by the sketch size, $NR^{n-1}J_1 + J_2R^n \approx NKR^{2n-2} + KR^{2n}$, when $J_1 = KR^{n-1}, J_2 = KR^n$.

References.

[1] N. Ailon and B. Chazelle, *The fast Johnson–Lindenstrauss transform and approximate nearest neighbors*, SIAM Journal on computing, 39 (2009), pp. 302–322.

[2] C. Boutsidis and A. Gittens, *Improved matrix algorithms via the subsampled randomized hadamard transform*, SIAM Journal on Matrix Analysis and Applications, 34 (2013), pp. 1301–1340.

[3] K. L. Clarkson and D. P. Woodruff, *Low-rank approximation and regression in input sparsity time*, Journal of the ACM (JACM), 63 (2017), p. 54.

[4] G. Cormode and M. Hadjieleftheriou, *Finding frequent items in data streams*, Proceedings of the VLDB Endowment, 1 (2008), pp. 1530–1541.

[5] L. De Lathauwer, B. De Moor, and J. Vandewalle, *A multilinear singular value decomposition*, SIAM journal on Matrix Analysis and Applications, 21 (2000), pp. 1253–1278.

[6] H. Diao, Z. Song, W. Sun, and D. P. Woodruff, *Sketching for Kronecker Product Regression and P-splines*, arXiv e-prints, (2017), arXiv:1712.09473, p. arXiv:1712.09473, https://arxiv.org/abs/1712.09473.

[7] N. Halko, P.-G. Martinsson, and J. A. Tropp, *Finding structure with randomness: Probabilistic algorithms for constructing approximate matrix decompositions*, SIAM review, 53 (2011), pp. 217–288.

[8] O. A. Malik and S. Becker, *Low-rank tucker decomposition of large tensors using tensorsketch*, in Advances in Neural Information Processing Systems, 2018, pp. 10116–10126.

[9] J. A. Tropp, *Improved analysis of the subsampled randomized hadamard transform*, Advances in Adaptive Data Analysis, 3 (2011), pp. 115–126.