Subquadratic Kronecker Regression with Applications to Tensor Decomposition

Anonymous Author(s)
Affiliation
Address
email

Abstract

Kronecker regression is a highly-structured least squares problem $\min_{\mathbf{x}} \|\mathbf{K}\mathbf{x} - \mathbf{b}\|_2^2$, where the design matrix $\mathbf{K} = \mathbf{A}^{(1)} \otimes \cdots \otimes \mathbf{A}^{(N)}$ is a Kronecker product of factor matrices. This regression problem arises in each step of the widely-used alternating least squares (ALS) algorithm for computing the Tucker decomposition of a tensor. We present the first *subquadratic-time* algorithm for solving Kronecker regression to a $(1+\varepsilon)$-approximation that avoids the exponential term $O(\varepsilon^{-N})$ in the running time. Our techniques combine leverage score sampling and iterative methods. By extending our approach to block-design matrices where one block is a Kronecker product, we also achieve subquadratic-time algorithms for (1) Kronecker ridge regression and (2) updating the factor matrix of a Tucker decomposition in ALS, which is not a pure Kronecker regression problem, thereby improving the running time of all steps of Tucker ALS. We demonstrate the speed and accuracy of this Kronecker regression algorithm on synthetic data and real-world image tensors.

1 Introduction

Tensor decomposition has a rich multidisciplinary history with countless applications in data mining, machine learning, and signal processing [31, 51, 53, 28]. The most widely-used tensor decompositions are the CP decomposition and the Tucker decomposition. Similar to the singular value decomposition of a matrix, both decompositions have natural analogs of *low-rank* structure. Unlike matrix factorization, however, computing the rank of a tensor is NP-hard [24]. Therefore, most low-rank tensor decomposition algorithms decide on the rank structure in advance, and then optimize the variables of the decomposition to fit the data. While conceptually simple, this approach is extremely effective in many applications.

The *alternating least squares* (ALS) algorithm is the main workhorse for low-rank tensor decomposition (e.g., it is the first algorithm mentioned in the MATLAB Tensor Toolbox [5]). For both CP and Tucker decompositions, ALS cyclically optimizes disjoint blocks of variables while keeping all others fixed. As the name suggests, each step solves a linear regression problem. The *core tensor* update step in ALS for Tucker decompositions is notoriously expensive but highly structured. In fact, the design matrix of this regression problem is the Kronecker product of the factor matrices of the Tucker decomposition $\mathbf{K} = \mathbf{A}^{(1)} \otimes \cdots \otimes \mathbf{A}^{(N)}$. Our work builds on a line of Kronecker regression algorithms [15, 16, 43] to give the first *subquadratic-time* algorithm for solving Kronecker regression to a $(1+\varepsilon)$-approximation while avoiding an exponential term of $O(\varepsilon^{-N})$ in the running time.

We combine leverage score sampling with iterative methods to fully exploit the Kronecker structure of the design matrix. We also extend our approach to block-design matrices where one block is a Kronecker product, achieving subquadratic-time algorithms for (1) Kronecker ridge regression and (2) updating the factor matrix of a Tucker decomposition in ALS, which is not a pure Kronecker regression problem. Putting everything together, this work improves the running time of all steps of

Submitted to 36th Conference on Neural Information Processing Systems (NeurIPS 2022). Do not distribute.

ALS for Tucker decompositions and runs in time that is sublinear in the size of the input tensor, linear in the error parameter ε^{-1}, and subquadratic in the number of columns of the design matrix in each step. Our algorithms support L2 regularization in the Tucker loss function, so the decompositions can readily be used in downstream learning tasks (e.g., using the factor matrix rows as embeddings for clustering [62]). Regularization also plays a critical role in the more general tensor completion problem to prevent overfitting when data is missing and has applications in differential privacy [8, 6].

The current fastest Kronecker regression algorithm of Diao et al. [16] solves this problem by leverage score sampling and achieves the following running times for $\mathbf{A}^{(n)} \in \mathbb{R}^{I_n \times R_n}$ with $I_n \geq R_n$, for all $n \in [N]$, and $R = \prod_{n=1}^{N} R_n$:

1. $\tilde{O}(\sum_{n=1}^{N}(\text{nnz}(\mathbf{A}^{(n)}) + R_n^{\omega}) + R^{\omega}\varepsilon^{-1})$ by sampling $\tilde{O}(R\varepsilon^{-1})$ rows from $\mathbf{K}$ by their leverage scores.
2. $\tilde{O}(\sum_{n=1}^{N}(\text{nnz}(\mathbf{A}^{(n)}) + R_n^{\omega}\varepsilon^{-1}) + R\varepsilon^{-N})$ by sampling $\tilde{O}(R_n\varepsilon^{-1})$ rows from each factor matrix $\mathbf{A}^{(n)}$ and taking the Kronecker product of the sampled factor matrices.

Note that the second approach is linear in R, but we have to pay an exponential cost in the number of factor matrices for the error parameter. In this paper, we show that the running time of the first approach can be improved to subquadratic in R without increasing the running time dependence on ε, simultaneously improving on both approaches.

Theorem 1.1. *For $n \in [N]$, let $\mathbf{A}^{(n)} \in \mathbb{R}^{I_n \times R_n}$, $I_n \geq R_n$, and $\mathbf{b} \in \mathbb{R}^{I_1 \cdots I_n}$. There is a $(1+\varepsilon)$-approximation algorithm for solving $\arg\min_{\mathbf{x}} \|(\mathbf{A}^{(1)} \otimes \cdots \otimes \mathbf{A}^{(N)})\mathbf{x} - \mathbf{b}\|_2^2$ that runs in time*

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

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

Without using the fast matrix multiplication of Gall and Urrutia [21] and Alman and Williams [4], the last term in (1) is $\tilde{O}(R^2\varepsilon^{-1})$, which is *already an improvement* over the normal $\tilde{O}(R^3\varepsilon^{-1})$ running time. With fast matrix multiplication, $\text{MM}(\prod_{n \in S} R_n, R\varepsilon^{-1}, \prod_{n \in [N] \setminus S} R_n)$ is subquadratic in R, for any nontrivial subset $S \notin \{\emptyset, [N]\}$, which is an improvement over $\tilde{O}(R^{\omega}\varepsilon^{-1}) \approx \tilde{O}(R^{2.373}\varepsilon^{-1})$. In cases when there exists a "balanced" subset S such that $\prod_{n \in S} R_n = \sqrt{R}$, our running time goes as low as $\tilde{O}(R^{1.626}\varepsilon^{-1})$ using [21]. For ease of notation, we denote the subquadratic improvement by the constant $\theta^* > 0$, where $R^{2-\theta^*} = \min_{S \subseteq [N]} \text{MM}(\prod_{n \in S} R_n, R, \prod_{n \in [N] \setminus S} R_n)$.

Although updating the core tensor in the ALS algorithm for Tucker decomposition is a pure Kronecker product regression as described in Theorem 1.1, updating the factor matrices is a regression problem of the form $\arg\min_{\mathbf{x}} \|\mathbf{K}\mathbf{M}\mathbf{x} - \mathbf{b}\|_2^2$, where $\mathbf{K}$ is a Kronecker product and $\mathbf{M}$ is a matrix without any particular structure. We show that such problems can be converted to block regression problems where one of the blocks is the Kronecker product $\mathbf{K}$. We then design sublinear-time leverage score sampling techniques for these block matrices, which leads to the following theorem about accelerating all ALS steps.

Theorem 1.2. *There is an ALS algorithm for L2-regularized Tucker decompositions that takes a tensor $\mathcal{X} \in \mathbb{R}^{I_1 \times \cdots \times I_N}$ and returns N factor matrices $\mathbf{A}^{(n)} \in \mathbb{R}^{I_n \times R_n}$ and a core tensor $\mathcal{G} \in \mathbb{R}^{R_1 \times \cdots \times R_n}$ such that each factor matrix and core update is a $(1+\varepsilon)$-approximation to the optimum with high probability. The running times of each step are:*

- *Core tensor: $\tilde{O}(\sum_{n=1}^{N}(\text{nnz}(\mathbf{A}^{(n)}) + R_n^{\omega}) + R^{2-\theta^*}\varepsilon^{-1})$,*
- *Factor matrix: $\tilde{O}(\sum_{n=1}^{N}(\text{nnz}(\mathbf{A}^{(n)}) + R_n^{\omega}) + I_k R_{\neq k}^{2-\theta^*}\varepsilon^{-1} + I_k R \sum_{n=1}^{N} R_n + R_k^{\omega})$,*

where $R = \prod_{n=1}^{N} R_n$, $R_{\neq k} = R/R_k$, and $\theta^* > 0$ is a constant derived from fast rectangular matrix multiplication.

Note that for tensors of relatively large order, the superlinear term in R is the bottleneck in many applications since R is exponential in the order of the tensor. Thus, our improvements are significant in both theory and practice as illustrated in our experiments in Section 6.

1.1 Our Contributions and Techniques

We present several new results about approximate Kronecker regression and the ALS algorithm for Tucker decompositions. Below is a summary of our contributions:

1. Our main technical contribution is the algorithm `FastKroneckerRegression` in Section 4. This Kronecker regression algorithm builds on the block-sketching tools introduced in Section 3, and combines iterative methods with fast rectangular matrix multiplication to achieve a running time that is *subquadratic* in the number of columns in the Kronecker matrix. Our key insight is to use the original (non-sketched) Kronecker product as the preconditioner in the Richardson iterations when solving the sketched problem. This allows us to fully exploit the Kronecker structure through fast Kronecker matrix-vector multiplications (e.g., the `KronMatMul` algorithm in Lemma 4.4), instead of computing the pseduoinverse of the sketched normal matrix $((\mathbf{SK})^\intercal \mathbf{SK} + \lambda \mathbf{I})^+$ as in Diao et al. [16].
2. We generalize our Kronecker regression techniques to work for Kronecker ridge regression and the factor matrix updates in ALS for Tucker decomposition. We show that a factor matrix update is equivalent to solving an equality-constrained Kronecker regression problem with a low-rank update to the preconditioner in the Richardson iterations. We can implement these new matrix-vector products nearly as fast by using the Woodbury matrix identity. Thus, we provably speed up each step of Tucker ALS (i.e., the core tensor and factor matrices).
3. We give a block-sketching toolkit in Section 3 that states we can sketch blocks of a matrix by their leverage scores (i.e., their leverage scores in isolation, not with respect to the entire block matrix). This is one of the ways we exploit the Kronecker product structure of the design matrix. This approach can be useful for constructing spectral approximations and for approximately solving block regression problems. One corollary is that we can use the "sketch-and-solve" method for any ridge regression problem (Corollary 3.5).
4. We compare `FastKroneckerRegression` with Diao et al. [16, Algorithm 1] on a synthetic Kronecker regression task studied in [15, 16] and as a subroutine in ALS for computing the Tucker decomposition of the image tensors [40, 46, 47]. Our results highlight the importance of reducing the runtime dependency on the number of columns in the Kronecker product.

1.2 Related Work

Kronecker Regression. Diao et al. [15] recently gave the first Kronecker regression algorithm based on `TensorSketch` [49] that is faster than forming the Kronecker product. Diao et al. [16] improved this by removing the dependence on $O(\text{nnz}(\mathbf{b}))$ from the running time, where $\mathbf{b} \in \mathbb{R}^{I_1 \cdots I_N}$ is the response vector. Marco et al. [43] have studied the *generalized* Kronecker regression problem.

Ridge Leverage Scores. Alaoui and Mahoney [3] extended the notion of statistical leverage scores to account for L2 regularization. Sampling from approximate ridge leverage score distributions has since played a key role in sparse low-rank matrix approximation [14], the Nyström method [45], bounding statistical risk in ridge regression [44], and ridge regression [12, 44, 37, 29]. Fast recursive algorithms for computing approximate leverage scores [13] and for solving overconstrained least squares [36] are also closely related.

Tensor Decomposition. Cheng et al. [11] and Larsen and Kolda [34] used leverage score sampling to speed up ALS for CP decomposition.[1] Song et al. [54] gave a polynomial-time, relative-error approximation algorithm for several low-rank tensor decompositions, which include CP and Tucker. Frandsen and Ge [20] showed that if the tensor has an exact Tucker decomposition, then all local minima are globally optimal. Randomized low-rank Tucker decompositions based on sketching have become increasingly popular, especially in streaming applications: [41, 56, 9, 55, 28, 42, 40, 2]. The more general problem of low-rank tensor completion is also a fundamental approach for estimating the values of missing data [1, 39, 26, 25, 19]. Fundamental algorithms for tensor completion are based on ALS [63, 22, 38], Riemannian optimization [33, 30, 48], or projected gradient methods [60].

[1] The design matrix in each step of ALS for CP decomposition is a Khatri–Rao product, not a Kronecker product. CP decomposition does not suffer from a bottleneck step like ALS for Tucker decomposition since it is a sparser decomposition (i.e., CP decomposition does not have a core tensor—just factor matrices).

2 Preliminaries

Notation. The *order* of a tensor is the number of its dimensions. We denote scalars by normal lowercase letters $x \in \mathbb{R}$, vectors by boldface lowercase letters $\mathbf{x} \in \mathbb{R}^n$, matrices by boldface uppercase letters $\mathbf{X} \in \mathbb{R}^{m \times n}$, and higher-order tensors by boldface script letters $\mathcal{X} \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_N}$. We use normal uppercase letters to denote the size of an index set (e.g., $[N] = \{1, 2, \ldots, N\}$). The i-th entry of a vector $\mathbf{x}$ is denoted by x_i, the (i, j)-th entry of a matrix $\mathbf{X}$ by x_{ij}, and the (i, j, k)-th entry of a third-order tensor $\mathcal{X}$ by x_{ijk}.

Algorithm 1 `TuckerALS`

Input: $\mathcal{X} \in \mathbb{R}^{I_1 \times \cdots \times I_N}$, $(R_1, R_2, \ldots, R_N)$, λ

Initialize core tensor $\mathcal{G} \in \mathbb{R}^{R_1 \times R_2 \times \cdots \times R_n}$
Initialize factors $\mathbf{A}^{(n)} \in \mathbb{R}^{I_n \times R_n}$ for $n \in [N]$
repeat
  for $n = 1$ to N **do**
    $\mathbf{K} \leftarrow \mathbf{A}^{(1)} \otimes \cdots \otimes \mathbf{A}^{(n-1)} \otimes \mathbf{A}^{(n+1)} \otimes \cdots \otimes \mathbf{A}^{(N)}$
    $\mathbf{B} \leftarrow \mathbf{X}_{(n)}$
    for $i = 1$ to I_n **do**
      $\mathbf{y}^* \leftarrow \arg\min_{\mathbf{y}} \|\mathbf{K}\mathbf{G}_{(n)}^\intercal \mathbf{y} - \mathbf{b}_{i:}^\intercal\|_2^2 + \lambda\|\mathbf{y}\|_2^2$
      Update factor row $\mathbf{a}_{i:}^{(n)} \leftarrow \mathbf{y}^{*\intercal}$
  $\mathbf{K} \leftarrow \mathbf{A}^{(1)} \otimes \mathbf{A}^{(2)} \otimes \cdots \otimes \mathbf{A}^{(N)}$
  $\mathbf{g}^* \leftarrow \arg\min_{\mathbf{g}} \|\mathbf{K}\mathbf{g} - \mathrm{vec}(\mathcal{X})\|_2^2 + \lambda\|\mathbf{g}\|_2^2$
  Update core tensor $\mathcal{G} \leftarrow \mathrm{vec}^{-1}(\mathbf{g}^*)$
until convergence
return $\mathcal{G}, \mathbf{A}^{(1)}, \mathbf{A}^{(2)}, \ldots, \mathbf{A}^{(N)}$

Linear Algebra. Let $\mathbf{I}_n$ denote the $n \times n$ identity matrix and $\mathbf{0}_{m \times n}$ denote the $m \times n$ zero matrix. The transpose of a matrix $\mathbf{A} \in \mathbb{R}^{m \times n}$ is $\mathbf{A}^\intercal$ and the Moore–Penrose inverse is $\mathbf{A}^+$. The singular value decomposition (SVD) of $\mathbf{A}$ is a factorization of the form $\mathbf{U}\mathbf{\Sigma}\mathbf{V}^\intercal$, where $\mathbf{U} \in \mathbb{R}^{m \times m}$ and $\mathbf{V} \in \mathbb{R}^{n \times n}$ are orthogonal matrices, and $\mathbf{\Sigma} \in \mathbb{R}^{m \times n}$ is a diagonal matrix with non-negative real numbers on its diagonal. The entries $\sigma_i(\mathbf{A})$ of $\mathbf{\Sigma}$ are the singular values of $\mathbf{A}$, and the number of non-zero singular values is equal to $r = \mathrm{rank}(\mathbf{A})$. The *compact SVD* is a related decomposition where $\mathbf{\Sigma} \in \mathbb{R}^{r \times r}$ is a diagonal matrix containing the non-zero singular values. The Kronecker product of two matrices $\mathbf{A} \in \mathbb{R}^{m \times n}$ and $\mathbf{B} \in \mathbb{R}^{p \times q}$ is denoted by $\mathbf{A} \otimes \mathbf{B} \in \mathbb{R}^{(mp) \times (nq)}$.

Tensor Products. *Fibers* of a tensor are the vectors we get by fixing all but one index. If $\mathcal{X}$ is a third-order tensor, we denote the column, row, and tube fibers by $\mathbf{x}_{:jk}$, $\mathbf{x}_{i:k}$, and $\mathbf{x}_{ij:}$, respectively. The *mode-n unfolding* of a tensor $\mathcal{X} \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_N}$ is the matrix $\mathbf{X}_{(n)} \in \mathbb{R}^{I_n \times (I_1 \cdots I_{n-1} I_{n+1} \cdots I_N)}$ that arranges the mode-n fibers of $\mathcal{X}$ as columns of $\mathbf{X}_{(n)}$ ordered lexicographically by index. The *vectorization* of $\mathcal{X} \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_N}$ is the vector $\mathrm{vec}(\mathcal{X}) \in \mathbb{R}^{I_1 I_2 \cdots I_N}$ formed by vertically stacking the entries of $\mathcal{X}$ ordered lexicographically by index. For example, this transforms $\mathbf{X} \in \mathbb{R}^{m \times n}$ into a tall vector $\mathrm{vec}(\mathbf{X})$ by stacking its columns. We use $\mathrm{vec}^{-1}(\mathbf{x})$ to undo this operation when it is clear from context what the shape of the output tensor should be.

The *n-mode product* of tensor $\mathcal{X} \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_N}$ and matrix $\mathbf{A} \in \mathbb{R}^{J \times I_n}$ is denoted by $\mathcal{Y} = \mathcal{X} \times_n \mathbf{A}$ where $\mathcal{Y} \in \mathbb{R}^{I_1 \times \cdots \times I_{n-1} \times J \times I_{n+1} \times \cdots \times I_N}$. This operation multiplies each mode-n fiber of $\mathcal{X}$ by the matrix $\mathbf{A}$. This operation is expressed elementwise as

$$(\mathcal{X} \times_n \mathbf{A})_{i_1 \ldots i_{n-1} j i_{n+1} \ldots i_N} = \sum_{i_n=1}^{I_n} x_{i_1 i_2 \ldots i_N} a_{j i_n}.$$

The Frobenius norm $\|\mathcal{X}\|_{\mathrm{F}}$ of a tensor $\mathcal{X}$ is the square root of the sum of the squares of its entries.

Tucker Decomposition. The *Tucker decomposition* decomposes tensor $\mathcal{X} \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_N}$ into a *core tensor* $\mathcal{G} \in \mathbb{R}^{R_1 \times R_2 \times \cdots \times R_N}$ and N *factor matrices* $\mathbf{A}^{(n)} \in \mathbb{R}^{I_n \times R_n}$. We compute a Tucker decomposition by minimizing the nonconvex loss function

$$L(\mathcal{G}, \mathbf{A}^{(1)}, \ldots, \mathbf{A}^{(N)}; \mathcal{X}) = \|\mathcal{X} - \mathcal{G} \times_1 \mathbf{A}^{(1)} \cdots \times_N \mathbf{A}^{(N)}\|_{\mathrm{F}}^2 + \lambda \left(\|\mathcal{G}\|_{\mathrm{F}}^2 + \sum_{n=1}^{N} \|\mathbf{A}^{(n)}\|_{\mathrm{F}}^2 \right),$$

where λ is the L2 regularization. Elements in the reconstructed tensor $\widehat{\mathcal{X}} \stackrel{\mathrm{def}}{=} \mathcal{G} \times_1 \mathbf{A}^{(1)} \times_2 \cdots \times_N \mathbf{A}^{(N)}$ are

$$\widehat{x}_{i_1 i_2 \ldots i_N} = \sum_{r_1=1}^{R_1} \cdots \sum_{r_N=1}^{R_N} g_{r_1 r_2 \ldots r_N} a_{i_1 r_1}^{(1)} \cdots a_{i_N r_N}^{(N)}. \tag{2}$$

Equation (2) demonstrates that $\widehat{\mathcal{X}}$ is the sum of $R_1 \cdots R_N$ rank-1 tensors. The tuple $(R_1, R_2, \ldots, R_N)$ is the *multilinear rank* of the decomposition. The multilinear rank is typically chosen in advance and much smaller than the dimensions of $\mathcal{X}$.

Alternating Least Squares. We present `TuckerALS` in Algorithm 1 and highlight its connections to Kronecker regression. The core tensor update (Lines 10–12) is a ridge regression problem where the design matrix $\mathbf{K}_{\text{core}} \in \mathbb{R}^{I_1 \cdots I_N \times R_1 \cdots R_N}$ is a Kronecker product of the factor matrices. Each factor matrix update (Lines 5-9) also has Kronecker product structure, but there are additional subspace constraints we must account for. We describe these constraints in more detail in Section 5.

3 Row Sampling and Approximate Regression

Here we establish our sketching toolkit. The λ-*ridge leverage score* of the i-th row of $\mathbf{A} \in \mathbb{R}^{n \times d}$ is

$$\ell_i^\lambda(\mathbf{A}) \stackrel{\text{def}}{=} \mathbf{a}_{i:}(\mathbf{A}^\intercal \mathbf{A} + \lambda \mathbf{I})^+ \mathbf{a}_{i:}^\intercal. \tag{3}$$

The matrix of *cross* λ-*ridge leverage scores* is $\mathbf{A}(\mathbf{A}^\intercal \mathbf{A} + \lambda \mathbf{I})^+ \mathbf{A}^\intercal$. We denote its diagonal by $\boldsymbol{\ell}^\lambda(\mathbf{A})$ because it contains the λ-ridge leverage scores of $\mathbf{A}$. Ridge leverage scores generalize *statistical leverage scores* in that setting $\lambda = 0$ gives the leverage scores of $\mathbf{A}$. We denote the vector of statistical leverage scores by $\boldsymbol{\ell}(\mathbf{A})$. If $\mathbf{A} = \mathbf{U}\boldsymbol{\Sigma}\mathbf{V}^\intercal$ is the compact SVD of $\mathbf{A}$, then for all $i \in [n]$, we have

$$\ell_i^\lambda(\mathbf{A}) = \sum_{k=1}^{r} \frac{\sigma_k^2(\mathbf{A})}{\sigma_k^2(\mathbf{A}) + \lambda} u_{ik}^2, \tag{4}$$

where $r = \text{rank}(\mathbf{A})$. It follows that every $\ell_i^\lambda(\mathbf{A}) \leq 1$ since $\mathbf{U}$ is an orthogonal matrix. We direct the reader to Alaoui and Mahoney [3] or Cohen et al. [13] for further details.

The main results in this paper build on approximate leverage score sampling for block matrices. The λ-ridge leverage scores of $\mathbf{A} \in \mathbb{R}^{n \times d}$ can be computed by appending $\sqrt{\lambda}\mathbf{I}_d$ to the bottom of $\mathbf{A}$ to get $\overline{\mathbf{A}} \in \mathbb{R}^{(n+d) \times d}$ and considering the leverage scores of $\overline{\mathbf{A}}$, so we state the following results in terms of statistical leverage scores without loss of generality.

Definition 3.1. For any $\mathbf{A} \in \mathbb{R}^{n \times d}$, the vector $\hat{\boldsymbol{\ell}}(\mathbf{A}) \in \mathbb{R}^n$ is a β-*overestimate* for the leverage score distribution of $\mathbf{A}$ if, for all $i \in [n]$, it satisfies

$$\frac{\hat{\ell}_i(\mathbf{A})}{\|\hat{\boldsymbol{\ell}}(\mathbf{A})\|_1} \geq \beta \frac{\ell_i(\mathbf{A})}{\|\boldsymbol{\ell}(\mathbf{A})\|_1} = \beta \frac{\ell_i(\mathbf{A})}{\text{rank}(\mathbf{A})}.$$

Next we describe the approximate leverage score sampling algorithm in Woodruff [59, Section 2.4]. The core idea here is that if we sample $\tilde{O}(d/\beta)$ rows and reweight them appropriately, this smaller *sketched* matrix can be used instead of $\mathbf{A}$ to give provable guarantees for many problems.

Definition 3.2 (Leverage score sampling)**.** Let $\mathbf{A} \in \mathbb{R}^{n \times d}$ and $\mathbf{p} \in [0,1]^n$ be a β-overestimate for the leverage score distribution of $\mathbf{A}$. $\texttt{SampleRows}(\mathbf{A}, s, \mathbf{p})$ denotes the following procedure. Initialize sketch matrix $\mathbf{S} = \mathbf{0}_{s \times n}$. For each row i of $\mathbf{S}$, independently and with replacement, select an index $j \in [n]$ with probability p_j and set $s_{ij} = 1/\sqrt{p_j s}$. Return sketch $\mathbf{S}$.

The main result in this section is that we can choose to sketch a single block of a matrix by the leverage scores of that block in isolation. This sketched submatrix can then be used with the other (non-sketched) block to give a spectral approximation to the original matrix or for approximate linear regression. The notation $\mathbf{A} \preccurlyeq \mathbf{B}$ is the Loewner order and means $\mathbf{B} - \mathbf{A}$ is positive semidefinite.

Lemma 3.3. *Let* $\mathbf{A} = [\mathbf{A}_1; \mathbf{A}_2]$ *be vertically stacked with* $\mathbf{A}_1 \in \mathbb{R}^{n_1 \times d}$ *and* $\mathbf{A}_2 \in \mathbb{R}^{n_2 \times d}$*. Let* $\mathbf{p} \in [0,1]^{n_1}$ *be a* β*-overestimate for the leverage score distribution of* $\mathbf{A}_1$*. If* $s > 144d\ln(2d/\delta)/(\beta\varepsilon^2)$*, the sketch* $\mathbf{S}$ *returned by* $\texttt{SampleRows}(\mathbf{A}_1, s, \mathbf{p})$ *guarantees, with probability at least* $1 - \delta$*, that*

$$(1-\varepsilon)\mathbf{A}^\intercal \mathbf{A} \preccurlyeq (\mathbf{S}\mathbf{A}_1)^\intercal \mathbf{S}\mathbf{A}_1 + \mathbf{A}_2^\intercal \mathbf{A}_2 \preccurlyeq (1+\varepsilon)\mathbf{A}^\intercal \mathbf{A}.$$

Lemma 3.4 (Approximate block regression)**.** *Consider the problem* $\arg\min_{\mathbf{x} \in \mathbb{R}^d} \|\mathbf{A}\mathbf{x} - \mathbf{b}\|_2^2$ *where* $\mathbf{A} = [\mathbf{A}_1; \mathbf{A}_2]$ *and* $\mathbf{b} = [\mathbf{b}_1; \mathbf{b}_2]$ *are vertically stacked and* $\mathbf{A}_1 \in \mathbb{R}^{n_1 \times d}$*,* $\mathbf{A}_2 \in \mathbb{R}^{n_2 \times d}$*,* $\mathbf{b}_1 \in$

$\mathbb{R}^{n_1}$, $\mathbf{b}_2 \in \mathbb{R}^{n_2}$. *Let* $\mathbf{p} \in [0,1]^{n_1}$ *be a* β*-overestimate for the leverage score distribution of* $\mathbf{A}_1$. *Let* $s \geq 1680d\ln(40d)/(\beta\varepsilon)$ *and let* $\mathbf{S}$ *be the output of* $\texttt{SampleRows}(\mathbf{A}_1, s, \mathbf{p})$. *If*

$$\tilde{\mathbf{x}}^* = \arg\min_{\mathbf{x}\in\mathbb{R}^d}\Big(\|\mathbf{S}(\mathbf{A}_1\mathbf{x} - \mathbf{b}_1)\|_2^2 + \|\mathbf{A}_2\mathbf{x} - \mathbf{b}_2\|_2^2\Big),$$

then, with probability at least $9/10$*, we have* $\|\mathbf{A}\tilde{\mathbf{x}}^* - \mathbf{b}\|_2^2 \leq (1+\varepsilon)\min_{\mathbf{x}\in\mathbb{R}^d}\|\mathbf{A}\mathbf{x} - \mathbf{b}\|_2^2$.

We defer the proofs of these results to Appendix A. The key idea behind Lemma 3.4 is that leverage scores do not increase if rows are appended to the matrix. This then allows us to prove a sketched submatrix version of Drineas et al. [17, Lemma 8] for approximate matrix multiplication and satisfy the structural conditions for approximate least squares in Drineas et al. [18]. One consequence is that we can "sketch and solve" ridge regression problems, which was recently shown in [58, Theorem 1].

Corollary 3.5. *For any* $\mathbf{A} \in \mathbb{R}^{n\times d}$, $\mathbf{b} \in \mathbb{R}^d$, $\lambda \geq 0$, *consider* $\arg\min_{\mathbf{x}\in\mathbb{R}^d}(\|\mathbf{A}\mathbf{x} - \mathbf{b}\|_2^2 + \lambda\mathbf{x}_2^2)$. *Let* $\mathbf{p} \in [0,1]^{n_1}$ *be a* β*-overestimate for the leverage scores of* $\mathbf{A}$ *and* $s \geq 1680d\ln(40d)/(\beta\varepsilon)$. *If* $\mathbf{S}$ *is the output of* $\texttt{SampleRows}(\mathbf{A}, s, \mathbf{p})$*, then, with probability at least* $9/10$*, the sketched solution* $\tilde{\mathbf{x}}^* = \arg\min_{\mathbf{x}\in\mathbb{R}^d}(\|\mathbf{S}(\mathbf{A}\mathbf{x} - \mathbf{b})\|_2^2 + \lambda\mathbf{x}_2^2)$ *gives a* $(1+\varepsilon)$*-approximation to the original problem.*

Remark 3.6. The success probability of the sketch can be boosted from $9/10$ to $1-\delta$ by sampling a factor of $O(\log(1/\delta))$ more rows. See the discussion in Chen and Price [10, Section 2] about matrix concentration bounds for more details.

4 Kronecker Regression

Now we describe the key ingredients that allow us to design an approximate Kronecker regression algorithm whose running time is *subquadratic* in the number of columns in the design matrix.

1. The leverage score distribution of a Kronecker product matrix $\mathbf{K} = \mathbf{A}^{(1)} \otimes \cdots \otimes \mathbf{A}^{(N)}$ is a *product distribution* of the leverage score distributions of its factor matrices. Therefore, we can sample rows of $\mathbf{K}$ from $\boldsymbol{\ell}(\mathbf{K})$ with replacement in $\tilde{O}(1)$ time after a preprocessing step.
2. The normal matrix $\mathbf{K}^\intercal\mathbf{K} + \lambda\mathbf{I}$ in the ridge regression problem $\min_{\mathbf{x}}\|\mathbf{K}\mathbf{x} - \mathbf{b}\|_2^2 + \lambda\|\mathbf{x}\|_2^2$ is a $O(1)$-spectral approximation to the sketched version $(\mathbf{S}\mathbf{K})^\intercal\mathbf{S}\mathbf{K} + \lambda\mathbf{I}$ by Lemma 3.3. This means we can use Richardson iteration with $(\mathbf{K}^\intercal\mathbf{K} + \lambda\mathbf{I})^+$ as the preconditioner to *solve the sketched instance*, which guarantees us a $(1+\varepsilon)$-approximation. Using $(\mathbf{K}^\intercal\mathbf{K} + \lambda\mathbf{I})^+$ as the preconditioner allows us to *heavily exploit the Kronecker structure* with fast matrix-vector multiplications.
3. At this point, *Kronecker matrix-vector multiplications* are still the bottleneck, so we partition the factor matrices into two groups by their number of columns and use fast rectangular matrix multiplication to get a subquadratic running time.

This first result shows how λ-ridge leverage scores of a Kronecker product matrix decompose according to the SVDs of its factor matrices. All missing proofs in this section are deferred to Appendix B.

Lemma 4.1. *Let* $\mathbf{K} = \mathbf{A}^{(1)} \otimes \mathbf{A}^{(2)} \otimes \cdots \otimes \mathbf{A}^{(N)}$*, where each factor matrix* $\mathbf{A}^{(n)} \in \mathbb{R}^{I_n\times R_n}$. *Let* $(i_1, i_2, \ldots, i_N)$ *be the natural row indexing of* $\mathbf{K}$ *by its factors. Let the factor SVDs be* $\mathbf{A}^{(n)} = \mathbf{U}^{(n)}\boldsymbol{\Sigma}^{(n)}\mathbf{V}^{(n)\intercal}$. *For any* $\lambda \geq 0$*, the* λ*-ridge leverage scores of* $\mathbf{K}$ *are*

$$\ell^{\lambda}_{(i_1,\ldots,i_N)}(\mathbf{K}) = \sum_{\mathbf{t}\in T}\frac{\prod_{n=1}^N \sigma_{t_n}^2(\mathbf{A}^{(n)})}{\prod_{n=1}^N \sigma_{t_n}^2(\mathbf{A}^{(n)}) + \lambda}\left(\prod_{n=1}^N u^{(n)}_{i_n t_n}\right)^2, \tag{5}$$

where the sum is over $T = [R_1] \times [R_2] \times \cdots \times [R_N]$. *For statistical leverage scores, this simplifies to* $\ell_{(i_1,\ldots,i_N)}(\mathbf{K}) = \prod_{n=1}^N \ell_{i_n}(\mathbf{A}^{(n)})$.

This proof repeatedly uses the mixed-product property for Kronecker products and the definition of λ-ridge leverage scores in Equation (3).

4.1 Iterative Methods

Now we state a result for the convergence rate of preconditioned Richardson iteration [52], which uses the notation $\|\mathbf{x}\|_{\mathbf{M}}^2 = \mathbf{x}^\intercal\mathbf{M}\mathbf{x}$.

Lemma 4.2 (Preconditioned Richardson iteration)**.** *Let* $\mathbf{M}$ *be a matrix where* $\mathbf{A}^\intercal\mathbf{A} \preccurlyeq \mathbf{M} \preccurlyeq \kappa \cdot \mathbf{A}^\intercal\mathbf{A}$ *for some* $\kappa \geq 1$*. Let* $\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} - \mathbf{M}^+(\mathbf{A}^\intercal\mathbf{A}\mathbf{x}^{(k)} - \mathbf{A}^\intercal\mathbf{b})$*. Then,*

$$\|\mathbf{x}^{(k)} - \mathbf{x}^*\|_{\mathbf{M}} \leq (1 - 1/\kappa)^k \|\mathbf{x}^{(0)} - \mathbf{x}^*\|_{\mathbf{M}},$$

where $\mathbf{x}^* = \arg\min_{\mathbf{x}\in\mathbb{R}^d} \|\mathbf{A}\mathbf{x} - \mathbf{b}\|_2^2$*.*

Remark 4.3. The ridge regression algorithm in Chowdhury et al. [12] is also based on sketching and preconditioned Richardson iteration. They consider short and wide matrices where $d \gg n$ and use the *sketched normal matrix as the preconditioner* to solve the original problem. One of our main technical contributions is to use the *original normal matrix as the preconditioner* to solve the sketched problem. Reversing this is advantageous because computing the pseduoinverse and matrix-vector products with the original Kronecker matrix is substantially less inexpensive due to its Kronecker structure. However, this still motivates the need for faster Kronecker matrix-vector multiplications.

4.2 Fast Kronecker-Matrix Multiplication

The next result is a simple but useful observation about extracting the rightmost factor matrix from the Kronecker product and recursively computing a new less expensive Kronecker-matrix multiplication.

Lemma 4.4. *Let* $\mathbf{A}^{(n)} \in \mathbb{R}^{I_n \times J_n}$*, for* $n \in [N]$*, and* $\mathbf{B} \in \mathbb{R}^{J_1 \cdots J_N \times K}$*. There is an algorithm* $\texttt{KronMatMul}([\mathbf{A}^{(1)}, \ldots, \mathbf{A}^{(N)}], \mathbf{B})$ *that computes*

$$(\mathbf{A}^{(1)} \otimes \mathbf{A}^{(2)} \otimes \cdots \otimes \mathbf{A}^{(N)})\mathbf{B} \in \mathbb{R}^{(I_1 \cdots I_N) \times K}$$

in $O(K \sum_{n=1}^N J_1 \cdots J_n I_n \cdots I_N)$ *time.*

The following theorem is more sophisticated. We write the statement in terms of rectangular matrix multiplication time $\mathrm{MM}(a, b, c)$, which is the time to multiply an $a \times b$ matrix by a $b \times c$ matrix.

Theorem 4.5. *Let* $\mathbf{A}^{(n)} \in \mathbb{R}^{I_n \times R_n}$*, for* $n \in [N]$*,* $I = I_1 \cdots I_N$*,* $R = R_1 \cdots R_N$*,* $\mathbf{b} \in \mathbb{R}^I$*,* $\mathbf{c} \in \mathbb{R}^R$*, and* $\mathbf{S} \in \mathbb{R}^{I \times I}$ *be a diagonal matrix with* $\tilde{O}(R\varepsilon^{-1})$ *nonzeros. The vectors* $(\mathbf{A}_1 \otimes \cdots \otimes \mathbf{A}_N)^\intercal\mathbf{S}\mathbf{b}$ *and* $\mathbf{S}(\mathbf{A}_1 \otimes \cdots \otimes \mathbf{A}_N)\mathbf{c}$ *can be computed in time* $\tilde{O}(\min_{T\subseteq[N]} \mathit{MM}(\prod_{n\in T} R_n, R\varepsilon^{-1}, \prod_{n\notin T} R_n))$*.*

The core idea behind Theorem 4.5 is that the factor matrices can be partitioned into two groups to achieve a good "column-product" balance, i.e., $\min_{T\subseteq[N]} \max\{\prod_{n\in T} R_n, \prod_{n\notin T} R_n\}$ is close to $\sqrt{R}$. Then we use the fact that $\mathrm{nnz}(\mathbf{S}) = \tilde{O}(R\varepsilon^{-1})$ with a sparsity-aware `KronMatMul` to solve each part of this partition separately, and combine them with fast rectangular matrix multiplication. If we achieve perfect balance, the running time is $\tilde{O}(R^{1.626}\varepsilon^{-1})$ using results of Gall and Urrutia [21], which are explained in detail in van den Brand and Nanongkai [57, Appendix C]. If one of these two factor matrix groups has at most 0.9 of the "column-product mass," the running time is $\tilde{O}(R^{1.9}\varepsilon^{-1})$.

4.3 Main Algorithm

We are now ready to present our main algorithm for solving approximate Kronecker regression.

Theorem 4.6. *For any Kronecker product matrix* $\mathbf{K} = \mathbf{A}^{(1)} \otimes \cdots \otimes \mathbf{A}^{(N)} \in \mathbb{R}^{I_1 \cdots I_N \times R_1 \cdots R_N}$*,* $\mathbf{b} \in \mathbb{R}^{I_1 \cdots I_N}$*,* $\lambda \geq 0$*,* $\varepsilon \in (0, 1/4]$*, and* $\delta > 0$*,* `FastKroneckerRegression` *returns* $\mathbf{x}^* \in \mathbb{R}^{R_1 \cdots R_N}$ *in* $\tilde{O}((R_1 \cdots R_N)^2\varepsilon^{-1}\log(1/\delta) + \sum_{n=1}^N I_n R_n^2)$ *time such that, with probability at least* $1 - \delta$*,*

$$\|\mathbf{K}\mathbf{x}^* - \mathbf{b}\|_2^2 + \lambda\|\mathbf{x}\|_2^2 \leq (1 + \varepsilon) \min_{\mathbf{x}} \|\mathbf{K}\mathbf{x} - \mathbf{b}\|_2^2 + \lambda\|\mathbf{x}\|_2^2.$$

We defer the proof to Appendix B.2. The core idea is that the matrix-vector products with $\mathbf{M}^+$ in each Richardson iteration (Line 13 of Algorithm 2) uses two fast Kronecker matrix-vector products. The approximation guarantee follows from the fact that the original normal matrix $\mathbf{K}^\intercal\mathbf{K} + \lambda\mathbf{I}$ is a spectral approximation (hence a valid preconditioner) for the *sketched* Kronecker regression problem.

Next, by using the theoretically faster MM-based Kronecker matrix-vector products in Theorem 4.5 instead of `KronMatMul` (Lemma 4.4), we decrease the time complexity of Algorithm 2 from quadratic to subquadratic.

Corollary 4.7. *Let* $R = R_1 \cdots R_N$*. The algorithm* `FastKroneckerRegression` *can run in time* $\tilde{O}(R^{2-\theta^*}\varepsilon^{-1}\log(1/\delta) + \sum_{n=1}^N I_n R_n^2)$*, where* $\theta^* > 0$ *is a constant derived from the optimally balanced* MM *expression in Theorem 4.5, i.e.,* $R^{2-\theta^*} = \min_{T\subseteq[N]} \mathrm{MM}(\prod_{n\in T} R_n, R, \prod_{n\notin T} R_n)$*.*

Algorithm 2 `FastKroneckerRegression`

Input: Factor matrices $\mathbf{A}^{(n)} \in \mathbb{R}^{I_n \times R_n}$, response vector $\mathbf{b} \in \mathbb{R}^{I_1 \cdots I_N}$, L2 regularization strength λ, error ε, failure probability δ

1: Set $R \leftarrow R_1 R_2 \cdots R_N$
2: **for** $n = 1$ to N **do**
3: Compute the SVD of $(\mathbf{A}^{(n)})^\intercal \mathbf{A}^{(n)} = \mathbf{V}^{(n)}((\mathbf{\Sigma}^{(n)})^\intercal \mathbf{\Sigma}^{(n)})(\mathbf{V}^{(n)})^\intercal$
4: Compute the leverage scores $\boldsymbol{\ell}(\mathbf{A}^{(n)})$ using Equation (3)
5: Initialize product distribution data structure $\mathcal{P}$ to sample indices from $(\boldsymbol{\ell}(\mathbf{A}^{(1)}), \cdots, \boldsymbol{\ell}(\mathbf{A}^{(N)}))$
6: Set $\mathbf{D} \leftarrow (\mathbf{\Sigma}^{(1)\intercal}\mathbf{\Sigma}^{(1)} \otimes \cdots \otimes \mathbf{\Sigma}^{(N)\intercal}\mathbf{\Sigma}^{(N)} + \lambda \mathbf{I}_R)^+$
7: Let $\mathbf{M}^+ = (\mathbf{V}^{(1)} \otimes \cdots \otimes \mathbf{V}^{(N)})\mathbf{D}(\mathbf{V}^{(1)} \otimes \cdots \otimes \mathbf{V}^{(N)})^\intercal$
8: Set $s \leftarrow \lceil 1680 R \ln(40R) \ln(1/\delta)/\varepsilon \rceil$
9: Set $\mathbf{S} \leftarrow \texttt{SampleRows}(\mathbf{K}, s, \mathcal{P})$
10: Let $\tilde{\mathbf{K}} = \mathbf{S}\mathbf{K}$ and $\tilde{\mathbf{b}} = \mathbf{S}\mathbf{b}$
11: Initialize $\mathbf{x} \leftarrow \mathbf{0}_R$
12: **repeat**
13: $\mathbf{x} \leftarrow \mathbf{x} - (1 - \sqrt{\varepsilon})\mathbf{M}^+(\tilde{\mathbf{K}}^\intercal \tilde{\mathbf{K}}\mathbf{x} + \lambda \mathbf{x} - \tilde{\mathbf{K}}^\intercal \tilde{\mathbf{b}})$ using fast Kronecker-matrix multiplication
14: **until** convergence
15: **return** $\mathbf{x}$

5 Applications to Low-Rank Tucker Decomposition

Now we apply our fast Kronecker regression algorithm to `TuckerALS` and prove Theorem 1.2. We list the running times of different factor matrix and core update algorithms in Table 2 (Appendix C) and analyze these subroutines in Appendix C.3.

Core Tensor Update. The core update running time in Theorem 1.2 is a direct consequence of our algorithm for fast Kronecker regression in Theorem 4.6. The only difference is that we avoid recomputing the SVD and Gram matrix of each factor since these are computed at the end of each factor matrix update and stored for future use.

Factor Matrix Update. The factor matrix updates require more work because of the $\mathbf{G}_{(n)}^\intercal \mathbf{y}$ term in Line 8 of `TuckerALS`. To overcome this, we substitute variables and recast each factor update as an equality-constrained Kronecker regression problem with an appended low-rank block to account for the L2 regularization of the original variables. To support this new low-rank block, we use the *Woodbury matrix identity* to extend the technique of using Richardson iterations with fast Kronecker matrix-vector multiplication for solving sketched regression instances.

The next result formalizes this substitution and reduces the problem to block Kronecker regression with a new subspace constraint. This relies on the fact that the least squares solution to $\|\mathbf{M}\mathbf{x} - \mathbf{z}\|_2^2$ with minimum norm is $\mathbf{M}^+\mathbf{z}$ [50]. All proofs in this section are deferred to Appendix C.

Lemma 5.1. *Let $\mathbf{A} \in \mathbb{R}^{n \times m}$, $\mathbf{M} \in \mathbb{R}^{m \times d}$, $\mathbf{b} \in \mathbb{R}^n$, and $\lambda \geq 0$. For any ridge regression problem of the form $\arg\min_{\mathbf{x} \in \mathbb{R}^d}(\|\mathbf{A}\mathbf{M}\mathbf{x} - \mathbf{b}\|_2^2 + \lambda\|\mathbf{x}\|_2^2)$, we can solve $\mathbf{z}_{\text{opt}} = \arg\min_{\mathbf{N}\mathbf{z}=\mathbf{0}}\|\mathbf{A}\mathbf{z} - \mathbf{b}\|_2^2 + \lambda\|\mathbf{M}^+\mathbf{z}\|_2^2$, where $\mathbf{N} = \mathbf{I}_m - \mathbf{M}\mathbf{M}^+$, and return vector $\mathbf{M}^+\mathbf{z}_{\text{opt}}$ instead.*

Letting $\mathbf{z} = \mathbf{G}_{(n)}^\intercal \mathbf{y}$ in Line 8 of `TuckerALS` and modifying `FastKroneckerRegression` to support additional low-rank updates to the preconditioner, we get the `FastFactorMatrixUpdate` algorithm, presented as Algorithm 3 in Appendix C.2. The analysis is similar to the proofs of Theorem 4.6 and Corollary 4.7. The factor matrix updates benefit in the same way as before from fast Kronecker matrix-vector products, and new low-rank block updates are supported via the Woodbury identity.

Theorem 5.2. *For any $\lambda \geq 0$, $\varepsilon \in (0, 1/4]$, and $\delta > 0$, the* `FastFactorMatrixUpdate` *algorithm updates $\mathbf{A}_{(n)} \in \mathbb{R}^{I_n \times R_n}$ in* `TuckerALS` *with a $(1+\varepsilon)$-approximation, with probability at least $1 - \delta$, in time $\tilde{O}(I_n R_{\neq n}^2 \varepsilon^{-1} \log(1/\delta) + I_n R \sum_{k=1}^N R_k + R_n^\omega)$.*

Corollary 5.3. `FastFactorMatrixUpdate` *updates $\mathbf{A}^{(n)} \in \mathbb{R}^{I_n \times R_n}$ in $\tilde{O}(I_n R_{\neq n}^{2-\theta^*} \varepsilon^{-1} \log(1/\delta) + I_n R \sum_{k=1}^N R_k + R_n^\omega)$ time, where $\theta^* > 0$ is the optimally balanced* MM *exponent in Theorem 4.5.*

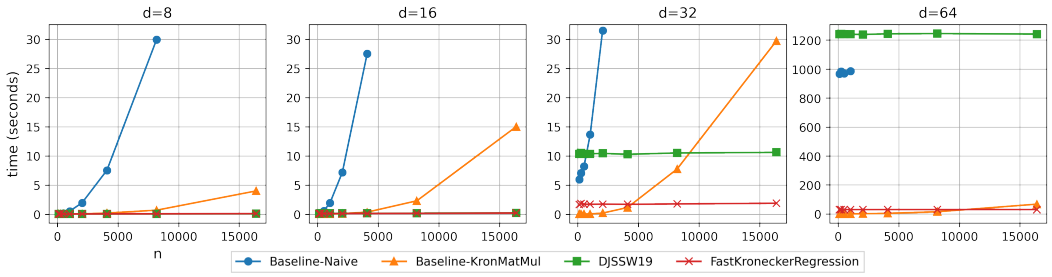

Figure 1: Running times of Kronecker regression algorithms with a design matrix of size $n^2 \times d^2$.

6 Experiments

All experiments use NumPy [23] with an Intel Xeon W-2135 processor (8.25MB cache, 3.70 GHz) and 128GB of RAM. The `FastKroneckerRegression`-based ALS experiments for low-rank Tucker decomposition on image tensors are deferred to Appendix D.2.

Kronecker regression. We build on the numerical experiments in [15, 16] for Kronecker regression that use two random factor matrices. We generate matrices $\mathbf{A}^{(1)}, \mathbf{A}^{(2)} \in \mathbb{R}^{n\times d}$ where each entry is drawn i.i.d. from the normal distribution $\mathcal{N}(1, 0.001)$ and compare several algorithms for solving $\min_{\mathbf{x}} \|(\mathbf{A}^{(1)} \otimes \mathbf{A}^{(2)})\mathbf{x} - \mathbf{1}_{n^2}\|_2^2 + \lambda\|\mathbf{x}\|_2^2$ as we increase n, d. The running times are plotted in Figure 1.

The algorithms we compare are: (1) a baseline that solves the normal equation $(\mathbf{K}^\intercal\mathbf{K}+\lambda\mathbf{I})^+\mathbf{K}^\intercal\mathbf{b}$ and fully exploits the Kronecker structure of $\mathbf{K}^\intercal\mathbf{K}$ before calling `np.linalg.pinv()`; (2) an enhanced baseline that combines the SVDs of $\mathbf{A}^{(n)}$ with Lemma 4.4, e.g., $\texttt{KronMatMul}([(\mathbf{U}^{(1)})^\intercal, (\mathbf{U}^{(2)})^\intercal], \mathbf{b})$, using only Kronecker-vector products; (3) the sketching algorithm of Diao et al. [16, Algorithm 1]; and (4) our `FastKroneckerRegression` algorithm in Algorithm 2. For both sketching algorithms, we use $\varepsilon = 0.1$ and $\delta = 0.01$. We reduce the number of row samples in both algorithms by $\alpha = 10^{-5}$ so that the algorithms are more practical and comparable to the earlier experiments in [15, 16]. Lastly, we set $\lambda = 10^{-3}$. We discuss additional parameter choice details and the full results in Appendix D.1.

The running times in Figure 1 demonstrate several different behaviors. The naive baseline quickly becomes impractical for moderately large values of n or d. `KronMatMul` is competitive for $n \leq 10^4$, especially since it is an exact method. The runtimes of the sketching algorithms are nearly-independent of n. Diao et al. [16] works well for small d, but deteriorates tremendously as d grows because it computes $((\mathbf{SK})^\intercal\mathbf{SK} + \lambda\mathbf{I})^+ \in \mathbb{R}^{d^2\times d^2}$ and cannot exploit the Kronecker structure of $\mathbf{K}$, which takes $O(d^6)$ time. `FastKroneckerRegression`, on the other hand, runs in $O(d^4)$ time because it uses quadratic-time Kronecker-vector products in each Richardson iteration step (Line 13).

Table 1: Kronecker regression losses for $d = 64$. OPT denotes the loss of the `KronMatMul` algorithm, `DJSSW19` is Diao et al. [16, Algorithm 1], and Algorithm 2 is `FastKroneckerRegression`. We also record the relative error of each algorithm and the number of rows sampled from $\mathbf{A}^{(1)} \otimes \mathbf{A}^{(2)}$.

n	OPT	Algorithm 2	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

These experiments also show that combining sketching with iterative methods can give better *sketch efficiency*. Table 1 compares the loss of [16, Algorithm 1] and `FastKroneckerRegression` to an exact baseline OPT for $d = 64$. Both algorithms use the exact same sketch $\mathbf{SK}$ for each value of n. Our algorithm uses the original $(\mathbf{K}^\intercal\mathbf{K} + \lambda\mathbf{I})^+$ as a preconditioner to solve the sketched problem, whereas Diao et al. [16, Algorithm 1] computes $((\mathbf{SK})^\intercal\mathbf{SK} + \lambda\mathbf{I})^+(\mathbf{SK})^\intercal\mathbf{Sb}$ exactly and becomes numerically unstable for $n \geq 2048$ when $d \in \{16, 32, 64\}$. This raises the question about how to combine sketched information with the original data to achieve more efficient algorithms, even when solving sketched instances. We leave this question of sketch efficiency as an interesting future work.

References

[1] Evrim Acar, Daniel M Dunlavy, Tamara G Kolda, and Morten Mørup. Scalable tensor factorizations for incomplete data. *Chemometrics and Intelligent Laboratory Systems*, 106(1):41–56, 2011.

[2] Salman Ahmadi-Asl, Stanislav Abukhovich, Maame G Asante-Mensah, Andrzej Cichocki, Anh Huy Phan, Tohishisa Tanaka, and Ivan Oseledets. Randomized algorithms for computation of tucker decomposition and higher order svd (hosvd). *IEEE Access*, 9:28684–28706, 2021.

[3] Ahmed Alaoui and Michael W. Mahoney. Fast randomized kernel ridge regression with statistical guarantees. In *Advances in Neural Information Processing Systems*, volume 28, 2015.

[4] Josh Alman and Virginia Vassilevska Williams. A refined laser method and faster matrix multiplication. In *Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 522–539. SIAM, 2021.

[5] Brett W. Bader and Tamara G. Kolda. Tensor toolbox for MATLAB, version 3.2.1. `https://www.tensortoolbox.org/`, 2021.

[6] Raghavendran Balu and Teddy Furon. Differentially private matrix factorization using sketching techniques. In *Proceedings of the 4th ACM Workshop on Information Hiding and Multimedia Security*, pages 57–62, 2016.

[7] Casey Battaglino, Grey Ballard, and Tamara G Kolda. A practical randomized cp tensor decomposition. *SIAM Journal on Matrix Analysis and Applications*, 39(2):876–901, 2018.

[8] Kamalika Chaudhuri, Claire Monteleoni, and Anand D Sarwate. Differentially private empirical risk minimization. *Journal of Machine Learning Research*, 12(3), 2011.

[9] Maolin Che and Yimin Wei. Randomized algorithms for the approximations of tucker and the tensor train decompositions. *Advances in Computational Mathematics*, 45(1):395–428, 2019.

[10] Xue Chen and Eric Price. Active regression via linear-sample sparsification. In *Conference on Learning Theory*, pages 663–695. PMLR, 2019.

[11] Dehua Cheng, Richard Peng, Yan Liu, and Ioakeim Perros. SPALS: Fast alternating least squares via implicit leverage scores sampling. *Advances in Neural Information Processing Systems*, 29:721–729, 2016.

[12] Agniva Chowdhury, Jiasen Yang, and Petros Drineas. An iterative, sketching-based framework for ridge regression. In *International Conference on Machine Learning*, pages 989–998. PMLR, 2018.

[13] Michael B Cohen, Yin Tat Lee, Cameron Musco, Christopher Musco, Richard Peng, and Aaron Sidford. Uniform sampling for matrix approximation. In *Proceedings of the 2015 Conference on Innovations in Theoretical Computer Science*, pages 181–190, 2015.

[14] Michael B Cohen, Cameron Musco, and Christopher Musco. Input sparsity time low-rank approximation via ridge leverage score sampling. In *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1758–1777. SIAM, 2017.

[15] Huaian Diao, Zhao Song, Wen Sun, and David Woodruff. Sketching for kronecker product regression and p-splines. In *International Conference on Artificial Intelligence and Statistics*, pages 1299–1308. PMLR, 2018.

[16] Huaian Diao, Rajesh Jayaram, Zhao Song, Wen Sun, and David Woodruff. Optimal sketching for kronecker product regression and low rank approximation. *Advances in neural information processing systems*, 32, 2019.

[17] Petros Drineas, Ravi Kannan, and Michael W Mahoney. Fast Monte Carlo algorithms for matrices I: Approximating matrix multiplication. *SIAM Journal on Computing*, 36(1):132–157, 2006.

[18] Petros Drineas, Michael W. Mahoney, Shan Muthukrishnan, and Tamás Sarlós. Faster least squares approximation. *Numerische Mathematik*, 117(2):219–249, 2011.

[19] Marko Filipović and Ante Jukić. Tucker factorization with missing data with application to low-n-rank tensor completion. *Multidimensional systems and signal processing*, 26(3):677–692, 2015.

[20] Abraham Frandsen and Rong Ge. Optimization landscape of tucker decomposition. *Mathematical Programming*, pages 1–26, 2020.

[21] François Le Gall and Florent Urrutia. Improved rectangular matrix multiplication using powers of the coppersmith-winograd tensor. In *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1029–1046. SIAM, 2018.

[22] Lars Grasedyck, Melanie Kluge, and Sebastian Kramer. Variants of alternating least squares tensor completion in the tensor train format. *SIAM Journal on Scientific Computing*, 37(5): A2424–A2450, 2015.

[23] Charles R. Harris, K. Jarrod Millman, Stéfan J. van der Walt, Ralf Gommers, Pauli Virtanen, David Cournapeau, Eric Wieser, Julian Taylor, Sebastian Berg, Nathaniel J. Smith, Robert Kern, Matti Picus, Stephan Hoyer, Marten H. van Kerkwijk, Matthew Brett, Allan Haldane, Jaime Fernández del Río, Mark Wiebe, Pearu Peterson, Pierre Gérard-Marchant, Kevin Sheppard, Tyler Reddy, Warren Weckesser, Hameer Abbasi, Christoph Gohlke, and Travis E. Oliphant. Array programming with NumPy. *Nature*, 585(7825):357–362, September 2020. doi: 10.1038/s41586-020-2649-2. URL `https://doi.org/10.1038/s41586-020-2649-2`.

[24] Christopher J Hillar and Lek-Heng Lim. Most tensor problems are NP-hard. *Journal of the ACM (JACM)*, 60(6):1–39, 2013.

[25] Prateek Jain and Sewoong Oh. Provable tensor factorization with missing data. *arXiv preprint arXiv:1406.2784*, 2014.

[26] Prateek Jain, Praneeth Netrapalli, and Sujay Sanghavi. Low-rank matrix completion using alternating minimization. In *Proceedings of the forty-fifth annual ACM symposium on Theory of computing*, pages 665–674, 2013.

[27] M James. The generalised inverse. *The Mathematical Gazette*, 62(420):109–114, 1978.

[28] Jun-Gi Jang and U Kang. Fast and memory-efficient tucker decomposition for answering diverse time range queries. In *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining*, pages 725–735, 2021.

[29] Praneeth Kacham and David P Woodruff. Sketching algorithms and lower bounds for ridge regression. *arXiv preprint arXiv:2204.06653*, 2022.

[30] Hiroyuki Kasai and Bamdev Mishra. Low-rank tensor completion: a Riemannian manifold preconditioning approach. In *International Conference on Machine Learning*, pages 1012–1021. PMLR, 2016.

[31] Tamara G. Kolda and Brett W. Bader. Tensor decompositions and applications. *SIAM Review*, 51(3):455–500, 2009.

[32] Jean Kossaifi, Yannis Panagakis, Anima Anandkumar, and Maja Pantic. Tensorly: Tensor learning in Python. *Journal of Machine Learning Research (JMLR)*, 20(26), 2019.

[33] Daniel Kressner, Michael Steinlechner, and Bart Vandereycken. Low-rank tensor completion by Riemannian optimization. *BIT Numerical Mathematics*, 54(2):447–468, 2014.

[34] Brett W. Larsen and Tamara G. Kolda. Practical leverage-based sampling for low-rank tensor decomposition. *arXiv preprint arXiv:2006.16438v2*, 2020.

[35] Brett W. Larsen and Tamara G. Kolda. Sketching matrix least squares via leverage scores estimates. *arXiv preprint arXiv:2201.10638*, 2022.

[36] Mu Li, Gary L Miller, and Richard Peng. Iterative row sampling. In *2013 IEEE 54th Annual Symposium on Foundations of Computer Science*, pages 127–136. IEEE, 2013.

[37] Zhu Li, Jean-Francois Ton, Dino Oglic, and Dino Sejdinovic. Towards a unified analysis of random Fourier features. In *International Conference on Machine Learning*, pages 3905–3914. PMLR, 2019.

[38] Allen Liu and Ankur Moitra. Tensor completion made practical. In *Advances in Neural Information Processing Systems*, volume 33, pages 18905–18916, 2020.

[39] Ji Liu, Przemyslaw Musialski, Peter Wonka, and Jieping Ye. Tensor completion for estimating missing values in visual data. *IEEE transactions on pattern analysis and machine intelligence*, 35(1):208–220, 2012.

[40] Linjian Ma and Edgar Solomonik. Fast and accurate randomized algorithms for low-rank tensor decompositions. *Advances in Neural Information Processing Systems*, 34, 2021.

[41] Osman Asif Malik and Stephen Becker. Low-rank Tucker decomposition of large tensors using TensorSketch. *Advances in Neural Information Processing Systems*, 31:10096–10106, 2018.

[42] Osman Asif Malik and Stephen Becker. A sampling-based method for tensor ring decomposition. In *International Conference on Machine Learning*, pages 7400–7411. PMLR, 2021.

[43] Ana Marco, José-Javier Martínez, and Raquel Viaña. Least squares problems involving generalized kronecker products and application to bivariate polynomial regression. *Numerical Algorithms*, 82(1):21–39, 2019.

[44] Shannon McCurdy. Ridge regression and provable deterministic ridge leverage score sampling. In *Advances in Neural Information Processing Systems*, volume 31, 2018.

[45] Cameron Musco and Christopher Musco. Recursive sampling for the Nyström method. In *Advances in Neural Information Processing Systems*, volume 30, 2017.

[46] Sérgio MC Nascimento, Kinjiro Amano, and David H Foster. Spatial distributions of local illumination color in natural scenes. *Vision research*, 120:39–44, 2016.

[47] Sameer A Nene, Shree K Nayar, Hiroshi Murase, et al. Columbia object image library (coil-20). 1996.

[48] Madhav Nimishakavi, Pratik Kumar Jawanpuria, and Bamdev Mishra. A dual framework for low-rank tensor completion. In *Advances in Neural Information Processing Systems*, volume 31, 2018.

[49] Rasmus Pagh. Compressed matrix multiplication. *ACM Transactions on Computation Theory (TOCT)*, 5(3):1–17, 2013.

[50] M. Planitz. Inconsistent systems of linear equations. *The Mathematical Gazette*, 63(425): 181–185, 1979.

[51] Stephan Rabanser, Oleksandr Shchur, and Stephan Günnemann. Introduction to tensor decompositions and their applications in machine learning. *arXiv preprint arXiv:1711.10781*, 2017.

[52] Yousef Saad. *Iterative methods for sparse linear systems*. SIAM, 2003.

[53] Nicholas D Sidiropoulos, Lieven De Lathauwer, Xiao Fu, Kejun Huang, Evangelos E Papalexakis, and Christos Faloutsos. Tensor decomposition for signal processing and machine learning. *IEEE Transactions on Signal Processing*, 65(13):3551–3582, 2017.

[54] Zhao Song, David P Woodruff, and Peilin Zhong. Relative error tensor low rank approximation. In *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 2772–2789. SIAM, 2019.

[55] Yiming Sun, Yang Guo, Charlene Luo, Joel Tropp, and Madeleine Udell. Low-rank tucker approximation of a tensor from streaming data. *SIAM Journal on Mathematics of Data Science*, 2(4):1123–1150, 2020.

[56] Abraham Traore, Maxime Berar, and Alain Rakotomamonjy. Singleshot : a scalable tucker tensor decomposition. In *Advances in Neural Information Processing Systems*, volume 32, 2019.

[57] Jan van den Brand and Danupon Nanongkai. Dynamic approximate shortest paths and beyond: Subquadratic and worst-case update time. In *2019 IEEE 60th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 436–455. IEEE, 2019.

[58] Shusen Wang, Alex Gittens, and Michael W Mahoney. Sketched ridge regression: Optimization perspective, statistical perspective, and model averaging. In *International Conference on Machine Learning*, pages 3608–3616. PMLR, 2017.

[59] David P. Woodruff. *Sketching as a Tool for Numerical Linear Algebra*. 2014.

[60] Rose Yu and Yan Liu. Learning from multiway data: Simple and efficient tensor regression. In *International Conference on Machine Learning*, pages 373–381. PMLR, 2016.

[61] Huamin Zhang and Feng Ding. On the kronecker products and their applications. *Journal of Applied Mathematics*, 2013, 2013.

[62] Guoxu Zhou, Andrzej Cichocki, and Shengli Xie. Decomposition of big tensors with low multilinear rank. *arXiv preprint arXiv:1412.1885*, 2014.

[63] Hua Zhou, Lexin Li, and Hongtu Zhu. Tensor regression with applications in neuroimaging data analysis. *Journal of the American Statistical Association*, 108(502):540–552, 2013.

Checklist

1. For all authors...
   (a) Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope? [Yes]
   (b) Did you describe the limitations of your work? [Yes]
   (c) Did you discuss any potential negative societal impacts of your work? [N/A]
   (d) Have you read the ethics review guidelines and ensured that your paper conforms to them? [Yes]
2. If you are including theoretical results...
   (a) Did you state the full set of assumptions of all theoretical results? [Yes]
   (b) Did you include complete proofs of all theoretical results? [Yes]
3. If you ran experiments...
   (a) Did you include the code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL)? [Yes]
   (b) Did you specify all the training details (e.g., data splits, hyperparameters, how they were chosen)? [Yes]
   (c) Did you report error bars (e.g., with respect to the random seed after running experiments multiple times)? [No]
   (d) Did you include the total amount of compute and the type of resources used (e.g., type of GPUs, internal cluster, or cloud provider)? [Yes]
4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets...
   (a) If your work uses existing assets, did you cite the creators? [Yes]
   (b) Did you mention the license of the assets? [Yes]
   (c) Did you include any new assets either in the supplemental material or as a URL? [Yes]
   (d) Did you discuss whether and how consent was obtained from people whose data you're using/curating? [N/A]
   (e) Did you discuss whether the data you are using/curating contains personally identifiable information or offensive content? [N/A]
5. If you used crowdsourcing or conducted research with human subjects...
   (a) Did you include the full text of instructions given to participants and screenshots, if applicable? [N/A]
   (b) Did you describe any potential participant risks, with links to Institutional Review Board (IRB) approvals, if applicable? [N/A]
   (c) Did you include the estimated hourly wage paid to participants and the total amount spent on participant compensation? [N/A]