Topics in Matrix and Tensor Computations

by

Osman Asif Malik

B.S., Chalmers University of Technology, 2009

B.S., University of Gothenburg, 2009

M.S., Imperial College London, 2012

M.S., University of Washington, 2016

A thesis submitted to the

Faculty of the Graduate School of the

University of Colorado in partial fulfillment

of the requirements for the degree of

Doctor of Philosophy

Department of Applied Mathematics

2021

Committee Members:

Stephen Becker, Chair

Alireza Doostan

Gregory Beylkin

François Meyer

Per-Gunnar Martinsson

Malik, Osman Asif (Ph.D., Applied Mathematics)

Topics in Matrix and Tensor Computations

Thesis directed by Prof. Stephen Becker

This dissertation looks at matrices and tensors from a computational perspective. An important problem in both matrix and tensor computations is decomposition. Several chapters in this dissertation deal with this problem. Chapter 2 develops randomized algorithms for Tucker tensor decomposition which are faster and can handle larger datasets than competing methods. These algorithms are the first ever one-pass methods for Tucker decomposition to appear in the literature. Chapter 4 develops randomized algorithms with guarantees for both matrix and tensor interpolative decomposition (ID). A key contribution is the first ever theoretical guarantees for any randomized tensor ID algorithm. Chapter 7 develops a randomized algorithm for the tensor ring decomposition. By using leverage score sampling, the resulting iterative algorithm has a per iteration cost which is sublinear in the number of input tensor entries. Chapter 8 considers the binary matrix factorization problem and develops a QUBO formulation for solving it on special purpose hardware like the Fujitsu Digital Annealer and the D-Wave Quantum Annealer.

In the tensor decomposition works mention above, randomization—more specifically, matrix sketching—is a key tool used to make algorithms faster and more efficient. In Chapter 3 we dive deeper into this topic and consider one such sketch, the Kronecker fast Johnson–Lindenstrauss transform (KFJLT). We provide a novel proof that the KFJLT indeed is a Johnson–Lindenstrauss transform. Randomization is also used in Chapter 5 to improve the performance of fast matrix multiplication algorithms plagued by either numerical rounding error or error in the computation formula itself. An upshot of this work is a simple method that can help make fast algorithms such as Strassen's more robust, especially on low-precision hardware like GPUs.

Machine learning is a field where tensor methods have received considerable attention. In Chapter 6, we develop a tensor based graph neural network that can be used for a variety of

prediction tasks on time varying graphs. We achieve competitive performance in prediction tasks, including in a COVID-19 contact tracing application.

Acknowledgements

First and foremost, I thank my advisor Stephen Becker for all his support and encouragement over the years, and for teaching me how to do research. I thank Alireza Doostan for serving on my committee and for the opportunity to work with him during the last year of my PhD. I also thank Gregory Beylkin, François Meyer and Per-Gunnar Martinsson for serving on my committee.

I have greatly enjoyed my time in Boulder. In large part, this is due to the many wonderful people I have had the privilege of getting to know during my time here. In particular, I thank Richie Clancy, Erik Johnson, David Kozak and Kevin Doherty for "making time for what's important."

I am thankful to all my collaborators. I want to thank Lior Horesh, Shashanka Ubaru, Misha Kilmer and Haim Avron for their mentorship during my internship at IBM Research. I want to thank Indradeep Ghosh, Hayato Ushijima-Mwesigwa, Avradip Mandal and Arnab Roy for their mentorship during my internship at Fujitsu Laboratories of America. I also want to thank Todd Murray and Venkatalakshmi Narumanchi for teaching me about photoacoustics and medical imaging.

Last, but not least, I thank my family, my friends and my partner Cassie for all their love and support.

Parts of the work in this dissertation was supported by the National Science Foundation under Grant No. ECCS-1810314 and by AFOSR Grant No. FA9550-20-1-0138. This work utilized resources from the University of Colorado Boulder Research Computing Group, which is supported by the National Science Foundation (awards ACI-1532235 and ACI-1532236), the University of Colorado Boulder, and Colorado State University.

Contents

Chapter

Tables

Table

Figures

Figure

Chapter 1

Introduction

Matrix computations are ubiquitous in mathematics, engineering and other quantitative sciences. The reasons for this is that they can be used to represent basic building blocks that come up again and again in computations. One way to view a matrix is as a **table of data** that we may want to compress or find some hidden structure within. A matrix can also be seen as a concrete representation of a **linear map** from one finite dimensional space to another. Since many phenomena can be locally approximated by linear functions, such matrix representations are extremely useful. Tensors can be seen as higher dimensional variants of matrices. From the same two viewpoints mentioned above, tensors can be seen as **multidimensional arrays** of numbers, or as **multilinear maps**. Just like for matrices, it is interesting to consider problems such as compression, finding hidden (e.g. low-rank) structure, and the design of more efficient computational algorithms for tensors. These are the sort of problems addressed in this dissertation.

The contents of Chapters 2–8 are based on material from seven papers [142, 143, 144, 145, 146, 147, 148], all of which I am the first author of. Six of these papers have been published or accepted for publication in journals or peer reviewed journal equivalent conferences. The chapters are ordered chronologically after publication date with the most recent (unpublished) paper last. As the title of this dissertation suggests, these works all deal with matrices and tensors from a computational perspective. There are a few reoccurring themes in these chapters that I want to highlight:

- **Decomposition.** Chapters 2, 4, 7 and 8 all deal with decomposition of tensors and matrices.

More specifically, they consider how to improve decomposition algorithms in some way, e.g., making them faster, more pass efficient, or more accurate.

- **Randomized algorithms.** Work on randomized algorithms in numerical linear algebra and scientific computing has been a hugely successful line of work [40]. The work in Chapters 2, 3, 4, 5 and 7 incorporates randomization in various ways.

- **Special purpose hardware.** One exciting aspect of computing is the many new hardware platforms that are being developed. While providing benefits, these platforms also pose new challenges. Parts of the work in Chapter 5 is relevant to computation on low-precision hardware like GPUs. The work in Chapter 8 is concerned with algorithm development on the Fujitsu Digital Annealer, and these ideas can also be implemented on other hardware that solves QUBO problems like the D-Wave Quantum Annealer.

- **Machine learning/data science.** The work in Chapter 6 considers an application of tensor methods in machine learning. Chapters 2 and 7 also have a data science flavor and are written for a machine learning audience.

1.1 Details on each chapter

The work in Chapter 2 was done in collaboration with Stephen Becker at University of Colorado Boulder. It has been previously published:

> **Osman Asif Malik** and Stephen Becker. *Low-rank Tucker decomposition of large tensors using TensorSketch.* Advances in Neural Information Processing Systems (NeurIPS), pages 10096–10106, 2018.

The abstract of the paper follows below.

> We propose two randomized algorithms for low-rank Tucker decomposition of tensors. The algorithms, which incorporate sketching, only require a single pass of the input tensor and can handle tensors whose elements are streamed in any order. To the

best of our knowledge, ours are the only algorithms which can do this. We test our algorithms on sparse synthetic data and compare them to multiple other methods. We also apply one of our algorithms to a real dense 38 GB tensor representing a video and use the resulting decomposition to correctly classify frames containing disturbances.

The work in Chapter 3 was done in collaboration with Stephen Becker at University of Colorado Boulder. It has been previously published:

> **Osman Asif Malik** and Stephen Becker. *Guarantees for the Kronecker fast Johnson–Lindenstrauss transform using a coherence and sampling argument.* Linear Algebra and its Applications 602, pages 120–137, 2020. DOI: 10.1016/j.laa.2020.05.004

The abstract of the paper follows below.

In the recent paper [Jin, Kolda & Ward, Inf. Inference, 2020], it is proved that the Kronecker fast Johnson–Lindenstrauss transform (KFJLT) is, in fact, a Johnson–Lindenstrauss transform, which had previously only been conjectured. In this paper, we provide an alternative proof of this, for when the KFJLT is applied to Kronecker vectors, using a coherence and sampling argument. Our proof yields a different bound on the embedding dimension, which can be combined with the bound in the paper by Jin et al. to get a better bound overall. As a stepping stone to proving our result, we also show that the KFJLT is a subspace embedding for matrices with columns that have Kronecker product structure. Lastly, we compare the KFJLT to four other sketch techniques in numerical experiments on both synthetic and real-world data.

The work in Chapter 4 was done in collaboration with Stephen Becker at University of Colorado Boulder. It has been previously published:

Osman Asif Malik and Stephen Becker. *Fast randomized matrix and tensor interpolative decomposition using CountSketch.* Advances in Computational Mathematics 46, article number 76, pages 1–28, 2020. DOI: 10.1007/s10444-020-09816-9

The abstract of the paper follows below.

We propose a new fast randomized algorithm for interpolative decomposition of matrices which utilizes CountSketch. We then extend this approach to the tensor interpolative decomposition problem introduced by Biagioni et al. [J. Comput. Phys. 281(C), 116–134 (2015)]. Theoretical performance guarantees are provided for both the matrix and tensor settings. Numerical experiments on both synthetic and real data demonstrate that our algorithms maintain the accuracy of competing methods, while running in less time, achieving at least an order of magnitude speedup on large matrices and tensors.

The work in Chapter 5 was done in collaboration with Stephen Becker at University of Colorado Boulder. It has been previously published:

Osman Asif Malik and Stephen Becker. *Randomization of approximate bilinear computation for matrix multiplication.* International Journal of Computer Mathematics: Computer Systems Theory 6(1), pages 54–93, 2021.
DOI: 10.1080/23799927.2020.1861104

The abstract of the paper follows below.

We present a method for randomizing formulas for bilinear computation of matrix products which does not increase the leading order complexity of the computation. We consider the implications of such randomization when there are two sources of error. The first source is due to the computation formula itself only being approximately correct. Such formulas come up when numerically searching for faster matrix multiplication algorithms. The second source is due to using floating

point arithmetic. This kind of error is especially important when computing on low precision hardware like GPUs. Our theoretical results and numerical experiments indicate that our method can improve performance when the two kinds of error are present individually, as well as when they are present at the same time.

The work in Chapter 6 was done in collaboration with Shashanka Ubaru and Lior Horesh at IBM Research, Misha Kilmer at Tufts University, and Haim Avron at Tel Aviv University. I did most of the work on this paper during an internship at IBM Research in Yorktown Heights, NY, during the summer of 2019. It has been previously published:

Osman Asif Malik, Shashanka Ubaru, Lior Horesh, Misha E. Kilmer and Haim Avron. *Dynamic graph convolutional networks using the tensor M-product.* SIAM International Conference on Data Mining (SDM), pages 729–737, 2021. DOI: 10.1137/1.9781611976700.82

The abstract of the paper follows below.

Many irregular domains such as social networks, financial transactions, neuron connections, and natural language constructs are represented using graph structures. In recent years, a variety of graph neural networks (GNNs) have been successfully applied for representation learning and prediction on such graphs. In many of the real-world applications, the underlying graph changes over time, however, most of the existing GNNs are inadequate for handling such dynamic graphs. In this paper we propose a novel technique for learning embeddings of dynamic graphs using a tensor algebra framework. Our method extends the popular graph convolutional network (GCN) for learning representations of dynamic graphs using the recently proposed tensor M-product technique. Theoretical results presented establish a connection between the proposed tensor approach and spectral convolution of tensors. The proposed method TM-GCN is consistent with the Message Passing Neural Network (MPNN) framework, accounting for both spatial and temporal

message passing. Numerical experiments on real-world datasets demonstrate the performance of the proposed method for edge classification and link prediction tasks on dynamic graphs. We also consider an application related to the COVID-19 pandemic, and show how our method can be used for early detection of infected individuals from contact tracing data.

The work in Chapter 7 was done in collaboration with Stephen Becker at University of Colorado Boulder. It has been accepted for publication and will appear in July 2021:

> **Osman Asif Malik** and Stephen Becker. *A sampling-based method for tensor ring decomposition.* To appear in the proceedings of International Conference on Machine Learning (ICML), 2021.

The abstract of the paper follows below.

We propose a sampling-based method for computing the tensor ring (TR) decomposition of a data tensor. The method uses leverage score sampled alternating least squares to fit the TR cores in an iterative fashion. By taking advantage of the special structure of TR tensors, we can efficiently estimate the leverage scores and attain a method which has complexity sublinear in the number of input tensor entries. We provide high-probability relative-error guarantees for the sampled least squares problems. We compare our proposal to existing methods in experiments on both synthetic and real data. Our method achieves substantial speedup—sometimes two or three orders of magnitude—over competing methods, while maintaining good accuracy. We also provide an example of how our method can be used for rapid feature extraction.

The work in Chapter 8 was done in collaboration with Hayato Ushijima-Mwesigwa, Arnab Roy, Avradip Mandal and Indradeep Ghosh who are all at Fujitsu Laboratories of America. I did most of the work on this paper during an internship at Fujitsu Laboratories of America in Sunnyvale,

CA, (although I worked remotely from Boulder due to COVID-19) during the summer of 2020. We have a paper which is under review at the time of writing, with a preprint available on arXiv:

> **Osman Asif Malik**, Hayato Ushijima-Mwesigwa, Arnab Roy, Avradip Mandal and Indradeep Ghosh. *Binary matrix factorization on special purpose hardware*. arXiv:2010.08693, 2020.

The abstract of the paper follows below.

Many fundamental problems in data mining can be reduced to one or more NP-hard combinatorial optimization problems. Recent advances in novel technologies such as quantum and quantum inspired hardware promise a substantial speedup for solving these problems compared to when using general purpose computers but often require the problem to be modeled in a special form, such as an Ising or QUBO model, in order to take advantage of these devices. In this work, we focus on the important binary matrix factorization (BMF) problem which has many applications in data mining. We propose two QUBO formulations for BMF. We show how clustering constraints can easily be incorporated into these formulations. The special purpose hardware we consider is limited in the number of variables it can handle which presents a challenge when factorizing large matrices. We propose a sampling based approach to overcome this challenge, allowing us to factorize large rectangular matrices. We run experiments on the Fujitsu Digital Annealer, a quantum inspired CMOS annealer, on both synthetic and real data, including gene expression data. These experiments show that our approach is able to produce more accurate BMFs than competing methods.

The papers that Chapters 5–8 are based on have supplements containing, e.g., additional experiments, proofs and detailed complexity analyses. These supplements appear in Chapters A–D in the appendix of this dissertation.

Chapter 2

Low-rank Tucker decomposition of large tensors using TensorSketch

2.1 Introduction

Many real datasets have more than two dimensions and are therefore better represented using tensors, or multi-way arrays, rather than matrices. In the same way that methods such as the singular value decomposition (SVD) can help in the analysis of data in matrix form, tensor decompositions are important tools when working with tensor data. As multidimensional datasets grow larger and larger, there is an increasing need for methods that can handle them, even on modest hardware. One approach to the challenge of handling big data, which has proven to be very fruitful in the past, is the use of randomization. In this paper, we present two algorithms for computing the Tucker decomposition of a tensor which incorporate random sketching. A key challenge to incorporating sketching in the Tucker decomposition is that the relevant design matrices are Kronecker products of the factor matrices. This makes them too large to form and store in RAM, which prohibits the application of standard sketching techniques. Recent work [9, 64, 170, 174] has led to a new technique called TensorSketch which is ideally suited for sketching Kronecker products. It is based on this technique that we develop our algorithms. Our algorithms, which are single pass and can handle streamed data, are suitable when the decomposition we seek is of low-rank. When we say that our algorithms can handle streamed data, we mean that they can decompose a tensor whose elements are revealed one at a time and then discarded, no matter which order this is done in. These streaming properties of our methods follow directly from the streaming properties of TensorSketch.

In some applications, such as the compression of scientific data produced by high-fidelity simulations, the data tensors can be very large (see e.g. the recent work [7]). Since such data frequently is produced incrementally, e.g. by stepping forward in time, a compression algorithm which is one-pass and can handle the tensor elements being streamed would make it possible to compress the data without ever having to store it in full. Our algorithms have these properties.

In summary, our paper makes the following algorithmic contributions:

- We propose two algorithms for Tucker decomposition which incorporate TensorSketch. They are intended to be used for low-rank decompositions.
- We propose an idea for defining the sketch operators upfront. In addition to increasing accuracy and reducing run time, it allows us to make several other improvements. These include only requiring a single pass of the data, and being able to handle tensors whose elements are streamed. To the best of our knowledge, ours are the only algorithms which can do this.

2.1.1 A brief introduction to tensors and the Tucker decomposition

We use the same notations and definitions as in the review paper by Kolda and Bader [116]. A **tensor** $\boldsymbol{\mathcal{X}} \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_N}$ is an array of dimension N, also called an N-way tensor. Boldface Euler script letters, e.g. $\boldsymbol{\mathcal{X}}$, denote tensors of dimension three or greater; bold capital letters, e.g. $\boldsymbol{X}$, denote matrices; bold lowercase letters, e.g. $\boldsymbol{x}$, denote vectors; and lowercase letters, e.g. x, denote scalars. We use a colon to denote all elements along a certain dimension; for example, $\boldsymbol{x}_{n:}$ is the nth row of $\boldsymbol{X}$, $\boldsymbol{x}_{:n}$ is the nth column of $\boldsymbol{X}$, and $\boldsymbol{X}_{::n}$ is the nth so called frontal slice of the 3-way tensor $\boldsymbol{\mathcal{X}}$. For scalars indicating dimension size, uppercase letters, e.g. I, will be used. The **Kronecker product** of two matrices $\boldsymbol{A} \in \mathbb{R}^{I_1 \times R_1}$ and $\boldsymbol{B} \in \mathbb{R}^{I_2 \times R_2}$ is denoted by $\boldsymbol{A} \otimes \boldsymbol{B} \in \mathbb{R}^{I_1 I_2 \times R_1 R_2}$ and is

defined by

$$\boldsymbol{A} \otimes \boldsymbol{B} \stackrel{\text{def}}{=} \begin{bmatrix} a_{11}\boldsymbol{B} & a_{12}\boldsymbol{B} & \cdots & a_{1R_1}\boldsymbol{B} \\ a_{21}\boldsymbol{B} & a_{22}\boldsymbol{B} & \cdots & a_{2R_1}\boldsymbol{B} \\ \vdots & \vdots & & \vdots \\ a_{I_11}\boldsymbol{B} & a_{I_12}\boldsymbol{B} & \cdots & a_{I_1R_1}\boldsymbol{B} \end{bmatrix}.$$

The **Khatri-Rao product** of two matrices $\boldsymbol{A} \in \mathbb{R}^{I_1 \times R}$ and $\boldsymbol{B} \in \mathbb{R}^{I_2 \times R}$ is denoted by $\boldsymbol{A} \odot \boldsymbol{B} \in \mathbb{R}^{I_1 I_2 \times R}$ and is defined by

$$\boldsymbol{A} \odot \boldsymbol{B} \stackrel{\text{def}}{=} \begin{bmatrix} \boldsymbol{a}_{:1} \otimes \boldsymbol{b}_{:1} & \boldsymbol{a}_{:2} \otimes \boldsymbol{b}_{:2} & \cdots & \boldsymbol{a}_{:R} \otimes \boldsymbol{b}_{:R} \end{bmatrix}.$$

The **mode-n matricization** of a tensor $\boldsymbol{\mathcal{X}} \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_N}$ is denoted by $\boldsymbol{X}_{(n)} \in \mathbb{R}^{I_n \times \prod_{i \neq n} I_i}$ and maps the element on position $(i_1, i_2, \ldots, i_N)$ in $\boldsymbol{\mathcal{X}}$ to position (i_n, j) in $\boldsymbol{X}_{(n)}$, where

$$j = 1 + \sum_{\substack{k=1 \\ k \neq n}}^{N} (i_k - 1) J_k \quad \text{with} \quad J_k = \prod_{\substack{m=1 \\ m \neq n}}^{k-1} I_m.$$

Similarly, $\boldsymbol{x}_{(:)} \in \mathbb{R}^{\prod_n I_n}$ will denote the vectorization of $\boldsymbol{\mathcal{X}}$ we get by stacking the columns of $\boldsymbol{X}_{(1)}$ into a long column vector. The **n-mode product** of a tensor $\boldsymbol{\mathcal{X}} \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_N}$ and matrix $\boldsymbol{A} \in \mathbb{R}^{J \times I_n}$ is denoted by $\boldsymbol{\mathcal{X}} \times_n \boldsymbol{A} \in \mathbb{R}^{I_1 \times \cdots \times I_{n-1} \times J \times I_{n+1} \times \cdots \times I_N}$ and defined by

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

We can express this definition more compactly using matrix notation as follows:

$$\boldsymbol{\mathcal{Y}} = \boldsymbol{\mathcal{X}} \times_n \boldsymbol{A} \quad \Leftrightarrow \quad \boldsymbol{Y}_{(n)} = \boldsymbol{A}\boldsymbol{X}_{(n)}.$$

The **norm** of $\boldsymbol{\mathcal{X}}$ is defined as

$$\|\boldsymbol{\mathcal{X}}\| \stackrel{\text{def}}{=} \sqrt{\sum_{i_1=1}^{I_1} \sum_{i_2=1}^{I_2} \cdots \sum_{i_N=1}^{I_N} x_{i_1 i_2 \cdots i_N}^2}.$$

For a positive integer n, we use the notation $[n] := \{1, 2, \ldots, n\}$.

There are multiple tensor decompositions. In this paper, we consider the **Tucker decomposition**. A Tucker decomposition of a tensor $\boldsymbol{\mathcal{X}} \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_N}$ is

$$\boldsymbol{\mathcal{X}} = \boldsymbol{\mathcal{G}} \times_1 \boldsymbol{A}^{(1)} \times_2 \boldsymbol{A}^{(2)} \cdots \times_N \boldsymbol{A}^{(N)} = [\![\boldsymbol{\mathcal{G}}; \boldsymbol{A}^{(1)}, \boldsymbol{A}^{(2)}, \ldots, \boldsymbol{A}^{(N)}]\!], \tag{2.1}$$

where $\mathcal{G} \in \mathbb{R}^{R_1 \times R_2 \times \cdots \times R_N}$ is called the **core tensor** and each $\boldsymbol{A}^{(n)} \in \boldsymbol{R}^{I_n \times R_n}$ is called a **factor matrix**. Without loss of generality, the factor matrices can be assumed to have orthonormal columns, which we will assume as well. We say that $\mathcal{X}$ in (2.1) is a rank-$(R_1, R_2, \ldots, R_N)$ tensor.

The Tucker decomposition problem of decomposing a data tensor $\mathcal{Y} \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_N}$ can be formulated as

$$\underset{\mathcal{G}, \boldsymbol{A}^{(1)}, \ldots, \boldsymbol{A}^{(N)}}{\arg\min} \{ \| \mathcal{Y} - [\![\mathcal{G}; \boldsymbol{A}^{(1)}, \ldots, \boldsymbol{A}^{(N)}]\!] \| : \mathcal{G} \in \mathbb{R}^{R_1 \times \cdots \times R_N}, \boldsymbol{A}^{(n)} \in \mathbb{R}^{I_n \times R_n} \text{ for } n \in [N] \}.$$

The standard approach to this problem is to use alternating least-squares (ALS). By rewriting the objective function appropriately (use e.g. Proposition 3.7 in [115]), we get the following steps, which are repeated until convergence:

1. For $n = 1, \ldots, N$, update

$$\boldsymbol{A}^{(n)} = \underset{\boldsymbol{A} \in \mathbb{R}^{I_n \times R_n}}{\arg\min} \left\| \Big(\bigotimes_{\substack{i=N \\ i \neq n}}^{1} \boldsymbol{A}^{(i)} \Big) \boldsymbol{G}_{(n)}^{\top} \boldsymbol{A}^{\top} - \boldsymbol{Y}_{(n)}^{\top} \right\|_F^2. \tag{2.2}$$

2. Update

$$\mathcal{G} = \underset{\mathcal{Z} \in \mathbb{R}^{R_1 \times \cdots \times R_N}}{\arg\min} \left\| \Big(\bigotimes_{i=N}^{1} \boldsymbol{A}^{(i)} \Big) \boldsymbol{z}_{(:)} - \boldsymbol{y}_{(:)} \right\|_2^2. \tag{2.3}$$

One can show that the solution for the nth factor matrix $\boldsymbol{A}^{(n)}$ in (2.2) is given by the R_n leading left singular vectors of the mode-n matricization of

$$\mathcal{Y} \times_1 \boldsymbol{A}^{(1)\top} \cdots \times_{n-1} \boldsymbol{A}^{(n-1)\top} \times_{n+1} \boldsymbol{A}^{(n+1)\top} \cdots \times_N \boldsymbol{A}^{(N)\top}.$$

Since each $\boldsymbol{A}^{(i)}$ has orthogonal columns, it turns out that the solution to (2.3) is given by

$$\mathcal{G} = \mathcal{Y} \times_1 \boldsymbol{A}^{(1)\top} \times_2 \boldsymbol{A}^{(2)\top} \cdots \times_N \boldsymbol{A}^{(N)\top}.$$

These insights lead to Algorithm 1, which we will refer to as TUCKER-ALS. It is also frequently called **higher-order orthogonal iteration** (HOOI), and is more accurate than **higher-order SVD** (HOSVD) which is another standard algorithm for Tucker decomposition. More details can be found in [116].

Algorithm 1: TUCKER-ALS (aka HOOI)

input : $\mathcal{Y}$, target rank $(R_1, R_2, \ldots, R_N)$
output: Rank-$(R_1, R_2, \ldots, R_N)$ Tucker decomposition $[\![\mathcal{G}; \boldsymbol{A}^{(1)}, \ldots, \boldsymbol{A}^{(N)}]\!]$ of $\mathcal{Y}$

Initialize $\boldsymbol{A}^{(2)}, \boldsymbol{A}^{(3)}, \ldots, \boldsymbol{A}^{(N)}$
repeat
  for $n = 1, \ldots, N$ **do**
    $\mathcal{Z} = \mathcal{Y} \times_1 \boldsymbol{A}^{(1)\top} \cdots \times_{n-1} \boldsymbol{A}^{(n-1)\top} \times_{n+1} \boldsymbol{A}^{(n+1)\top} \cdots \times_N \boldsymbol{A}^{(N)\top}$
    $\boldsymbol{A}^{(n)} = R_n$ leading left singular vectors of $\boldsymbol{Z}_{(n)}$ /* Solves Eq. (2.2) */
  end
until termination criteria met
$\mathcal{G} = \mathcal{Y} \times_1 \boldsymbol{A}^{(1)\top} \times_2 \boldsymbol{A}^{(2)\top} \cdots \times_N \boldsymbol{A}^{(N)\top}$ /* Solves Eq. (2.3) */
return $\mathcal{G}$, $\boldsymbol{A}^{(1)}, \ldots, \boldsymbol{A}^{(N)}$

2.1.2 A brief introduction to CountSketch and TensorSketch

We apply TensorSketch to approximate the solution to large overdetermined least-squares problems, and to approximate chains of TTM products similar to those in (2.1). TensorSketch is a randomized method which allows us to reduce the cost and memory usage of these computations in exchange for somewhat reduced accuracy. It can be seen as a specialized version of another sketching method called CountSketch, which was introduced in [44] and further analyzed in [50]. We first explain how CountSketch can be applied to least-squares problems and for matrix multiplication, and then explain how these ideas extend to TensorSketch. Nothing in this subsection is our own work; it is simply a brief introduction to the sketching techniques we use.

The basic idea of applying CountSketch to least-squares regression is the following. Suppose $\boldsymbol{A} \in \mathbb{R}^{I \times R}$, where $I \gg R$, and $\boldsymbol{y} \in \mathbb{R}^I$, and consider solving the overdetermined least-squares problem

$$\boldsymbol{x}^* \stackrel{\text{def}}{=} \arg\min_{\boldsymbol{x} \in \mathbb{R}^R} \|\boldsymbol{A}\boldsymbol{x} - \boldsymbol{y}\|_2.$$

CountSketch allows us to reduce the size of this problem by first applying a subspace embedding matrix $\boldsymbol{S} : \mathbb{R}^I \to \mathbb{R}^J$ to $\boldsymbol{A}$ and $\boldsymbol{y}$, where J is much smaller than I, and so that we instead solve

$$\boldsymbol{x}' \stackrel{\text{def}}{=} \arg\min_{\boldsymbol{x} \in \mathbb{R}^R} \|\boldsymbol{S}\boldsymbol{A}\boldsymbol{x} - \boldsymbol{S}\boldsymbol{y}\|_2. \tag{2.4}$$

One way to define the matrix $\boldsymbol{S}$ is as $\boldsymbol{S} = \boldsymbol{P}\boldsymbol{D}$, where

- $\boldsymbol{P} \in \mathbb{R}^{J \times I}$ is a matrix with $p_{h(i),i} = 1$, and all other entries equal to 0;
- $h : [I] \to [J]$ is a random map such that $(\forall i \in [I])(\forall j \in [J])\ \mathbb{P}(h(i) = j) = 1/J$; and
- $\boldsymbol{D} \in \mathbb{R}^{I \times I}$ is a diagonal matrix, with each diagonal entry equal to $+1$ or -1 with equal probability.

For a fixed $\varepsilon > 0$, one can ensure that the solution to the sketched problem (2.4) satisfies

$$\|\boldsymbol{A}\boldsymbol{x}' - \boldsymbol{y}\|_2 \leq (1 + \varepsilon)\|\boldsymbol{A}\boldsymbol{x}^* - \boldsymbol{y}\|_2 \tag{2.5}$$

with high probability by choosing the sketch dimension J sufficiently large. Moreover, the matrix $\boldsymbol{S}$ can be applied in $O(\text{nnz}(\boldsymbol{A}))$ time, where $\text{nnz}(\boldsymbol{A})$ denotes the number of nonzero elements of $\boldsymbol{A}$, making it an especially efficient sketch if $\boldsymbol{A}$ is sparse; see [50] for further details.

CountSketch can also be used for approximate matrix multiplication. Let $\boldsymbol{S} \in \mathbb{R}^{J \times I}$ denote the CountSketch operator. Informally, for matrices $\boldsymbol{A}$ and $\boldsymbol{B}$ with I rows and a fixed $\varepsilon > 0$, one can ensure that

$$\|\boldsymbol{A}^\top \boldsymbol{S}^\top \boldsymbol{S}\boldsymbol{B} - \boldsymbol{A}^\top \boldsymbol{B}\|_F \leq \varepsilon \|\boldsymbol{A}\|_F \|\boldsymbol{B}\|_F \tag{2.6}$$

with high probability by choosing the sketch dimension J sufficiently large. A formal statement and proof of this is given in Lemma 32 of [50].

Now suppose the matrix $\boldsymbol{A}$ is of the form $\boldsymbol{A} = \boldsymbol{A}^{(1)} \otimes \boldsymbol{A}^{(2)} \otimes \cdots \otimes \boldsymbol{A}^{(N)}$, where each $\boldsymbol{A}^{(n)} \in \mathbb{R}^{I_n \times R_n}$, $I_n > R_n$. Then $\boldsymbol{A}$ is of size $I \times R$, where $I \stackrel{\text{def}}{=} \prod_n I_n$ and $R \stackrel{\text{def}}{=} \prod_n R_n$. TensorSketch is a restricted variant of CountSketch which allows us to sketch $\boldsymbol{A}$ without having to first form the matrix. This is done by instead sketching each factor matrix $\boldsymbol{A}^{(n)}$ individually, and then computing the corresponding sketch of $\boldsymbol{A}$, which can be done efficiently using the fast Fourier transform (FFT).

TensorSketch was first introduced in [170] where it is applied to compressed matrix multiplication. In [174], TensorSketch is used for approximating support vector machine polynomial kernels efficiently. Avron et al. [9] show that TensorSketch provides an oblivious subspace embedding. Diao et al. [64] show that an approximate solution to the least-squares problem in the sense of (2.5) when $\boldsymbol{A}$ is a Kronecker product can be obtained with high probability by solving the TensorSketched

problem instead of the full problem. The remainder of this subsection will briefly describe how TensorSketch works.

In the following, in order to adhere to Matlab notation with indexing starting at 1, some definitions will differ slightly from those in [64].

Definition 1 (k-wise independent)**.** A family $\mathcal{H} \stackrel{\text{def}}{=} \{h : [I] \to [J]\}$ of hash functions is **k-wise independent** if for any distinct $x_1, \ldots, x_k \in [I]$, and uniformly random $h \in \mathcal{H}$, the hash codes $h(x_1), \ldots, h(x_k)$ are independent random variables, and the hash code of any fixed $x \in [I]$ is uniformly distributed in $[J]$. We also call a function h drawn randomly and uniformly from $\mathcal{H}$ k-wise independent.

For $n \in [N]$, let

- $h_n : [I_n] \to [J]$ be 3-wise independent hash functions, and let
- $s_n : [I_n] \to \{-1, +1\}$ be 4-wise independent sign functions.

Furthermore, define the hash function

$$H : [I_1] \times [I_2] \times \cdots \times [I_N] \to [J] : (i_1, \ldots, i_N) \mapsto \Big(\sum_{n=1}^{N} (h_n(i_n) - 1) \mod J \Big) + 1$$

and the sign function

$$S : [I_1] \times [I_2] \times \cdots \times [I_N] \to \{-1, 1\} : (i_1, \ldots, i_N) \mapsto \prod_{n=1}^{N} s_n(i_n).$$

TensorSketch is the same as applying CountSketch to $\boldsymbol{A}$ with $\boldsymbol{P}$ defined using H instead of h, and with the diagonal of $\boldsymbol{D}$ given by S. To understand how the maps H and S map a certain row index, note that there is a bijection between the set of rows $i \in [\prod_n I_n]$ of $\boldsymbol{A}$ and the N-tuples $(i_1, i_2, \ldots, i_N) \in [I_1] \times [I_2] \times \cdots \times [I_N]$. Specifically, for the ith row, there is a unique N-tuple $(i_1, i_2, \ldots, i_N)$ such that $\boldsymbol{a}_{i:} = \boldsymbol{a}^{(1)}_{i_1:} \otimes \boldsymbol{a}^{(2)}_{i_2:} \otimes \cdots \otimes \boldsymbol{a}^{(N)}_{i_N:}$. Pagh [170] observed that the application of CountSketch based on H and S can be done efficiently—without ever forming $\boldsymbol{A}$—by CountSketching each factor matrix $\boldsymbol{A}^{(n)}$ using h_n and s_n, and then computing the TensorSketch of $\boldsymbol{A}$ using FFT. Let $\boldsymbol{S}^{(n)}$ be the CountSketch matrix corresponding to using the hash function h_n and the diagonal

matrix $\boldsymbol{D}^{(n)}$ with diagonal $d_{ii}^{(n)} = s_n(i)$. The application of $\boldsymbol{S}^{(n)}$ to $\boldsymbol{a}_{:r_n}^{(n)}$, the r_nth column of $\boldsymbol{A}^{(n)}$, can be represented by the $J-1$ degree polynomial

$$\mathcal{P}_{n,r_n}(\omega) = \sum_{i=1}^{I_n} s_n(i) a_{ir_n}^{(n)} \omega^{h_n(i)-1} = \sum_{j=1}^{J} c_{jr_n}^{(n)} \omega^{j-1},$$

where $\boldsymbol{c}_{r_n}^{(n)} \stackrel{\text{def}}{=} (c_{1r_n}^{(n)}, \ldots, c_{Jr_n}^{(n)})$ are the polynomial coefficients. With this representation, the elements of $\boldsymbol{a}_{:r_n}^{(n)}$ are multiplied with the correct sign given by s_n, and then grouped together in different polynomial coefficients depending on which bucket h_n assigns them to. Letting $\boldsymbol{T}$ denote the TensorSketch operator, the rth column of $\boldsymbol{TA}$ can similarly be represented as the polynomial

$$\begin{aligned}
\mathcal{P}_r(\omega) &= \sum_{i=1}^{I} S(i_1, \ldots, i_N) a_{ir} \omega^{H(i_1,\ldots,i_N)} \qquad (2.7)\\
&= \sum_{i=1}^{I} s_1(i_1) \cdots s_N(i_N) a_{i_1 r_1}^{(1)} \cdots a_{i_N r_N}^{(N)} \omega^{h_1(i_1)+\cdots+h_N(i_N)-N \mod J}\\
&= \mathrm{FFT}^{-1}\Big(\mathrm{FFT}\Big(\boldsymbol{c}_{r_1}^{(1)}\Big) * \cdots * \mathrm{FFT}\Big(\boldsymbol{c}_{r_N}^{(N)}\Big)\Big),
\end{aligned}$$

where FFT denotes the unpadded FFT, "$*$" denotes Hadamard (element-wise) product and where the i in (2.7) corresponds to the N-tuple $(i_1, i_2, \ldots, i_N) \in [I_1] \times [I_2] \times \cdots \times [I_N]$. Similarly, r corresponds to the N-tuple $(r_1, r_2, \ldots, r_N) \in [R_1] \times [R_2] \times \cdots \times [R_N]$. So each column of $\boldsymbol{TA}$ can be computed efficiently from the sketches of the corresponding columns of the factor matrices. More concisely, we can write

$$\boldsymbol{TA} = \mathrm{FFT}^{-1}\Big(\Big(\bigodot_{n=1}^{N} \Big(\mathrm{FFT}\Big(\boldsymbol{S}^{(n)}\boldsymbol{A}^{(n)}\Big)\Big)^{\top}\Big)^{\top}\Big), \qquad (2.8)$$

where FFT is applied columnwise to a matrix argument. Applying $\boldsymbol{T}$ naively to $\boldsymbol{A}$ would, in the general dense case, cost $O(J\prod_{n=1}^{N} I_n R_n)$. With the trick above, however, it is straightforward to compute the reduced cost to be $O(J\sum_{n=1}^{N} I_n R_n + J\log J \sum_{n=1}^{N} R_n + J \log J \prod_{n=1}^{N} R_n)$.

Theorem 3.1 in [64] gives theoretical guarantees for the optimality of the solution to (2.4) when $\boldsymbol{S}$ is TensorSketch. We have found the sketch dimension of that theorem to be pessimistic in practice, with much smaller sketch dimensions yielding satisfying results. Indeed, Example 6.1 of [64] also achieves good accuracy with a significantly smaller sketch dimension than what the theorem dictates.

Approximate matrix multiplication as in (2.6) also holds for TensorSketch; see Lemma B.1 in [64] for a formal statement and proof.

2.2 Related work

Randomized algorithms have been applied to tensor decompositions before. Wang et al. [210] and Battaglino et al. [15] apply sketching techniques to the CANDECOMP/PARAFAC (CP) decomposition. Drineas and Mahoney [68], Zhou and Cichocki [227], Da Costa et al. [53] and Tsourakakis [199] propose different randomized methods for computing HOSVD. The method in [199], which is called MACH, is also extended to computing HOOI. Mahoney et al. [140] and Caiafa and Cichocki [42] present results that extend the CUR factorization for matrices to tensors. Other decomposition methods that only consider a small number of the tensor entries include those by Oseledets et al. [168] and Friedland et al. [78].

Another approach to decomposing large tensors is to use memory efficient and distributed methods. Kolda and Sun [117] introduce the Memory Efficient Tucker (MET) decomposition for sparse tensors as a solution to the so called intermediate blow-up problem which occurs when computing the chain of TTM products in HOOI. Other papers that use memory efficient and distributed methods include [7, 14, 104, 109, 127, 128, 133, 163, 187].

Other research focuses on handling streamed tensor data. Sun et al. [192] introduce a framework for incremental tensor analysis. The basic idea of their method is to find one set of factor matrices which works well for decomposing a sequence of tensors that arrive over time. Fanaee-T and Gama [77] introduce multi-aspect-streaming tensor analysis which is based on the histogram approximation concept rather than linear algebra techniques. Neither of these methods correspond to Tucker decomposition of a tensor whose elements are streamed. Gujral et al. [88] present a method for incremental CP decomposition.

We compare our algorithms to TUCKER-ALS and MET in Tensor Toolbox version 2.6 [11, 117], FSTD1 with adaptive index selection from [42], as well as the HOOI version of the

MACH algorithm in [199].[1] TUCKER-ALS and MET, which are mathematically equivalent, provide good accuracy, but run out of memory as the tensor size increases. MACH scales somewhat better, but also runs out of memory for larger tensors. Its accuracy is also lower than that of TUCKER-ALS/MET. None of these algorithms are one-pass. FSTD1 scales well, but has accuracy issues on very sparse tensors. FSTD1 does not need to access all elements of the tensor and is one-pass, but since the entire tensor needs to be accessible the method cannot handle streamed data.

2.3 Tucker decomposition using TensorSketch

We now present our proposed algorithms. More detailed versions of them can be found in Section S3 of the supplement of our paper. A Matlab implementation of our algorithms can be found at `https://github.com/OsmanMalik/tucker-tensorsketch`.

2.3.1 First proposed algorithm: TUCKER-TS

For our first algorithm, we TensorSketch both the least-squares problems in (2.2) and (2.3), and then solve the smaller resulting problems. We give an algorithm for this approach in Algorithm 2. We call it TUCKER-TS, where "TS" stands for TensorSketch. The core tensor and factor matrices in line 1 are initialized randomly with each element i.i.d. Uniform$(-1, 1)$. The factor matrices are subsequently orthogonalized. On line 2 we define TensorSketch operators of appropriate size. This is done by first defining CountSketch operators $\boldsymbol{S}_1^{(n)} \in \mathbb{R}^{J_1 \times I_n}$ and $\boldsymbol{S}_2^{(n)} \in \mathbb{R}^{J_2 \times I_n}$ for $n \in [N]$, as explained in Section 2.1.2. Then each operator $\boldsymbol{T}^{(n)}$, for $n \in [N]$, is defined as in (2.8) but based on $\{\boldsymbol{S}_1^{(n)}\}_{n\in[N]}$ and with the nth term excluded in the Kronecker and Khatri-Rao products. $\boldsymbol{T}^{(N+1)}$ is defined similarly, but based on $\{\boldsymbol{S}_2^{(n)}\}_{n\in[N]}$ and without excluding any terms in the Kronecker and Khatri-Rao products. The reason we use two different sets $\{\boldsymbol{S}_1^{(n)}\}_{n\in[N]}$ and $\{\boldsymbol{S}_2^{(n)}\}_{n\in[N]}$ of

[1] For FSTD1, we use the Matlab code from the website of one of the authors (`http://ccaiafa.wixsite.com/cesar`). For MACH, we adapted the Python code provided on the author's website (`https://tsourakakis.com/mining-tensors/`) to Matlab. MACH requires an algorithm for computing the HOOI decomposition of the sparsified tensor. For this, we use TUCKER-ALS and then switch to higher orders of MET as necessary when we run out of memory. As recommended in [199], we keep each nonzero entry in the original tensor with probability 0.1 when using MACH.

CountSketch operators with different target sketch dimensions J_1 and J_2, respectively, is that the design matrix in (2.3) has more rows than that in (2.2). In practice, this means that we choose $J_2 > J_1$. In Section 2.4 we provide some guidance on how to choose J_1 and J_2. We also want to point out that none of the sketch operators are stored explicitly as matrices in our implementation. Instead, we only generate and store the function h and the diagonal of $\boldsymbol{D}$, which were defined in Section 2.1.2, for each CountSketch operator. We then use the formula in (2.8) when applying one of the TensorSketch operators to a Kronecker product matrix. The computations $\boldsymbol{T}^{(n)}\boldsymbol{Y}_{(n)}^{\top}$ and $\boldsymbol{T}^{(N+1)}\boldsymbol{y}_{(:)}$ on lines 5 and 7 cannot be done using the formula in (2.8), but are still computed implicitly without forming any full sketching matrices.

Since all sketch operators used on line 5 are defined in terms of the same set $\{\boldsymbol{S}_1^{(n)}\}_{n\in[N]}$, the least-squares problem on all iterations of that line except the first will depend in some way on the sketch operator $\boldsymbol{T}^{(n)}$ being applied. A similar dependence will exist between the least-squares problem on line 7 and $\boldsymbol{T}^{(N+1)}$ beyond the first iteration. It is important to note that the guarantees for TensorSketched least-squares in [64] hold when the random sketch is independent of the least-squares problem it is applied to. For these guarantees to hold, we would need to define a new TensorSketch operator each time a least-squares problem is solved in Algorithm 2. In all of our experiments, we observe that our approach of instead defining the sketch operators upfront leads to a substantial reduction in the error for the algorithm as a whole (see Figures 2.1 and 2.2). We have not yet been able to provide theoretical justification for why this is.

The following proposition shows that the normal equations formulation of the least-squares problem on line 7 in Algorithm 2 is well-conditioned with high probability if J_2 is sufficiently large, and therefore can be efficiently solved using conjugate gradient (CG). This is true because the factor matrices are orthogonal, and does not hold for the smaller system on line 5, so this system we solve via direct methods. In our experiments, for an accuracy of 1e-6, CG takes about 15 iterations regardless of I.

Proposition 2. *Assume $\boldsymbol{T}^{(N+1)}$ is defined as in line 2 in Algorithm 2. Let*

$$\boldsymbol{M} \stackrel{\text{def}}{=} (\boldsymbol{T}^{(N+1)} \bigotimes_{i=N}^{1} \boldsymbol{A}^{(i)})^{\top} (\boldsymbol{T}^{(N+1)} \bigotimes_{i=N}^{1} \boldsymbol{A}^{(i)}),$$

where all $\boldsymbol{A}^{(n)}$ have orthonormal columns, and suppose $\varepsilon, \delta \in (0,1)$. If $J_2 \geq (\prod_n R_n)^2(2+3^N)/(\varepsilon^2\delta)$, then the 2-norm condition number of $\boldsymbol{M}$ satisfies $\kappa(\boldsymbol{M}) \leq (1+\varepsilon)^2/(1-\varepsilon)^2$ with at least probability $1-\delta$.

Proof. Let V be the subspace spanned by the columns of $\boldsymbol{U} \stackrel{\text{def}}{=} \bigotimes_{i=N}^{1} \boldsymbol{A}^{(i)}$. Note that since each $\boldsymbol{A}^{(i)}$ has orthonormal columns, so does $\boldsymbol{U}$ [203]. Suppose $J_2 \geq (\prod_n R_n)^2(2+3^N)/(\varepsilon^2\delta)$. Using Theorem 4.2.2 in [97], we have that the maximum and minimum eigenvalues of $\boldsymbol{M}$ satisfy

$$\lambda_{\max}(\boldsymbol{M}) = \max_{\boldsymbol{x}:\|\boldsymbol{x}\|=1} \|\boldsymbol{T}^{(N+1)}\boldsymbol{U}\boldsymbol{x}\|_2^2,$$

$$\lambda_{\min}(\boldsymbol{M}) = \min_{\boldsymbol{x}:\|\boldsymbol{x}\|=1} \|\boldsymbol{T}^{(N+1)}\boldsymbol{U}\boldsymbol{x}\|_2^2.$$

Due to Lemma B.1 in [64], we therefore have

$$\lambda_{\max}(\boldsymbol{M}) \leq (1+\varepsilon)^2 \max_{\boldsymbol{x}:\|\boldsymbol{x}\|=1} \|\boldsymbol{U}\boldsymbol{x}\|_2^2 = (1+\varepsilon)^2 \max_{\boldsymbol{x}:\|\boldsymbol{x}\|=1} \|\boldsymbol{x}\|_2^2 = (1+\varepsilon)^2,$$

$$\lambda_{\min}(\boldsymbol{M}) \geq (1-\varepsilon)^2 \min_{\boldsymbol{x}:\|\boldsymbol{x}\|=1} \|\boldsymbol{U}\boldsymbol{x}\|_2^2 = (1-\varepsilon)^2 \min_{\boldsymbol{x}:\|\boldsymbol{x}\|=1} \|\boldsymbol{x}\|_2^2 = (1-\varepsilon)^2,$$

with probability at least $1-\delta$, since $\boldsymbol{U}\boldsymbol{x} \in V$ for all $\boldsymbol{x} \in \mathbb{R}^{\prod_n R_n}$. Since $\boldsymbol{M}$ is symmetric and positive semi-definite, we also have singular values $\sigma_{\max}(\boldsymbol{M}) = \lambda_{\max}(\boldsymbol{M}) \leq (1+\varepsilon)^2$, $\sigma_{\min}(\boldsymbol{M}) = \lambda_{\min}(\boldsymbol{M}) \geq (1-\varepsilon)^2$. The result now follows:

$$\kappa(\boldsymbol{M}) = \frac{\sigma_{\max}}{\sigma_{\min}} \leq \frac{(1+\varepsilon)^2}{(1-\varepsilon)^2}.$$

□

Remark 3. Defining the sketching operators upfront allows us to make the following improvements:

(a) Since $\mathcal{Y}$ remains unchanged throughout the algorithm, the $N+1$ sketches of $\mathcal{Y}$ only need to be computed once, which we do upfront in a single pass over the data (using a careful implementation). This can also be done if elements of $\mathcal{Y}$ are streamed.

(b) Since the same CountSketch is applied to each $\boldsymbol{A}^{(n)}$ when sketching the Kronecker product in the inner loop, we can compute the quantity $\hat{\boldsymbol{A}}_{s_1}^{(n)} \stackrel{\text{def}}{=} (\text{FFT}(\boldsymbol{S}_1^{(n)}\boldsymbol{A}^{(n)}))^\top$ after updating $\boldsymbol{A}^{(n)}$ and reuse it when computing other factor matrices until $\boldsymbol{A}^{(n)}$ is updated again.

(c) When $I_n \geq J_1 + J_2$ for some $n \in [N]$, we can reduce the size of the least-squares problem on line 5. Note that the full matrix $\boldsymbol{A}^{(n)}$ is not needed until the return statement—only the sketches $\boldsymbol{S}_1^{(n)}\boldsymbol{A}^{(n)}$ and $\boldsymbol{S}_2^{(n)}\boldsymbol{A}^{(n)}$ are necessary to compute the different TensorSketches. Replacing $\boldsymbol{T}^{(n)}\boldsymbol{Y}_{(n)}^\top$ on line 5 with $\boldsymbol{T}^{(n)}[\boldsymbol{Y}_{(n)}^\top\boldsymbol{S}_1^{(n)\top},\ \boldsymbol{Y}_{(n)}^\top\boldsymbol{S}_2^{(n)\top}]$, which also can be computed upfront, we get a smaller least-squares problem which has the solution $[\boldsymbol{S}_1^{(n)}\boldsymbol{A}^{(n)},\ \boldsymbol{S}_2^{(n)}\boldsymbol{A}^{(n)}]$. Before the return statement, we then compute the full factor matrix $\boldsymbol{A}^{(n)}$. With this adjustment, we cannot orthogonalize the factor matrices on each iteration, and therefore Proposition 2 does not apply. In this case, we therefore use a dense method instead of CG when computing $\mathcal{G}$ in Algorithm 2.

Algorithm 2: TUCKER-TS (proposal)

Data: $\mathcal{Y}$, target rank $(R_1, R_2, \ldots, R_N)$, sketch dimensions (J_1, J_2)
Result: Rank-$(R_1, R_2, \ldots, R_N)$ Tucker decomposition $[\![\mathcal{G}; \boldsymbol{A}^{(1)}, \ldots, \boldsymbol{A}^{(N)}]\!]$ of $\mathcal{Y}$

1 Initialize $\mathcal{G}, \boldsymbol{A}^{(2)}, \boldsymbol{A}^{(3)}, \ldots, \boldsymbol{A}^{(N)}$
2 Define TensorSketch operators $\boldsymbol{T}^{(n)} \in \mathbb{R}^{J_1 \times \prod_{i \neq n} I_i}$, for $n \in [N]$, and $\boldsymbol{T}^{(N+1)} \in \mathbb{R}^{J_2 \times \prod_i I_i}$
3 **repeat**
4 **for** $n = 1, \ldots, N$ **do**
5 $\boldsymbol{A}^{(n)} = \arg\min_{\boldsymbol{A}} \| \left(\boldsymbol{T}^{(n)} \bigotimes_{i=N, i\neq n}^{1} \boldsymbol{A}^{(i)}\right) \boldsymbol{G}_{(n)}^\top \boldsymbol{A}^\top - \boldsymbol{T}^{(n)}\boldsymbol{Y}_{(n)}^\top \|_F^2$
6 **end**
7 $\mathcal{G} = \arg\min_{\mathcal{Z}} \| \left(\boldsymbol{T}^{(N+1)} \bigotimes_{i=N}^{1} \boldsymbol{A}^{(i)}\right) \boldsymbol{z}_{(:)} - \boldsymbol{T}^{(N+1)}\boldsymbol{y}_{(:)} \|_2^2$
8 Orthogonalize each $\boldsymbol{A}^{(i)}$ and update $\mathcal{G}$
9 **until termination criteria met**
10 **return** $\mathcal{G}$, $\boldsymbol{A}^{(1)}, \ldots, \boldsymbol{A}^{(N)}$

2.3.2 Second proposed algorithm: TUCKER-TTMTS

We can rewrite the TTM product on line 4 of Algorithm 1 to $\boldsymbol{Z}_{(n)} = \boldsymbol{Y}_{(n)} \bigotimes_{\substack{i=N \\ i\neq n}}^{1} \boldsymbol{A}^{(i)}$. We TensorSketch this formulation as follows: $\tilde{\boldsymbol{Z}}_{(n)} = (\boldsymbol{T}^{(n)}\boldsymbol{Y}_{(n)}^{\top})^{\top}\boldsymbol{T}^{(n)} \bigotimes_{\substack{i=N \\ i\neq n}}^{1} \boldsymbol{A}^{(i)}$, $n \in [N]$, where each $\boldsymbol{T}^{(n)} \in \mathbb{R}^{J_1 \times \prod_{i\neq n} I_i}$ is a TensorSketch operator with target dimension J_1. We can similarly sketch the computation on line 8 in Algorithm 1 using a TensorSketch operator $\boldsymbol{T}^{(N+1)} \in \mathbb{R}^{J_2 \times \prod_i I_i}$ with target dimension J_2. Replacing the computations on lines 4 and 8 in Algorithm 1 with these sketched computations, we get our second algorithm which we call TUCKER-TTMTS, where "TTMTS" stands for "TTM TensorSketch." The algorithm is given in Algorithm 3. The initialization of the factor matrices on line 1a and the definition of the sketching operators on line 1b are done in the same way as in Algorithm 2. Since the sketch operators are defined upfront here as well, the same caveat applies here as for Algorithm 2. The main benefit of TUCKER-TTMTS over TUCKER-TS is that it scales better with the target rank (see Section 2.3.4).

The following proposition shows that the error for each sketched computation in TUCKER-TTMTS is additive rather than multiplicative as for TUCKER-TS. Let

$$\mathcal{G} = \mathcal{Y} \times_1 \boldsymbol{A}^{(1)\top} \times_2 \boldsymbol{A}^{(2)\top} \cdots \times_N \boldsymbol{A}^{(N)\top}. \tag{2.9}$$

Moreover, let $\boldsymbol{A}^*$ and $\mathcal{G}^*$ be the updated values on lines 5 and 8 in Algorithm 1, and let $\tilde{\boldsymbol{A}}$ and $\tilde{\mathcal{G}}$ be the corresponding updates in Algorithm 3.

Proposition 4 (TUCKER-TTMTS)**.** *Assume that $\boldsymbol{G}_{(n)}\boldsymbol{G}_{(n)}^{\top}$ is invertible, with $\mathcal{G}$ given in* (2.9)*, and that each TensorSketch operator is redefined prior to being used. If the sketch dimension $J_1 \geq (2 + 3^{N-1})/(\varepsilon^2\delta)$, then with probability at least $1 - \delta$ it holds that*

$$\Big\| \Big(\bigotimes_{\substack{i=N \\ i\neq n}}^{1} \boldsymbol{A}^{(i)} \Big) \boldsymbol{G}_{(n)}^{\top} \tilde{\boldsymbol{A}}^{\top} - \boldsymbol{Y}_{(n)}^{\top} \Big\|_F \leq \Big\| \Big(\bigotimes_{\substack{i=N \\ i\neq n}}^{1} \boldsymbol{A}^{(i)} \Big) \boldsymbol{G}_{(n)}^{\top} \boldsymbol{A}^{*\top} - \boldsymbol{Y}_{(n)}^{\top} \Big\|_F + \varepsilon \Big\| \Big(\boldsymbol{G}_{(n)}\boldsymbol{G}_{(n)}^{\top} \Big)^{-1} \Big\|_F \|\mathcal{G}\|^2 \|\mathcal{Y}\| \prod_{\substack{i=1 \\ i\neq n}}^{N} R_i.$$

Moreover, if the sketch dimension $J_2 \geq (2 + 3^N)/(\varepsilon^2\delta)$, then with probability at least $1 - \delta$ it holds

that

$$\Big\|\Big(\bigotimes_{i=N}^{1}\boldsymbol{A}^{(i)}\Big)\tilde{\boldsymbol{g}}_{(:)}-\boldsymbol{y}_{(:)}\Big\|_2\leq\Big\|\Big(\bigotimes_{i=N}^{1}\boldsymbol{A}^{(i)}\Big)\boldsymbol{g}^*_{(:)}-\boldsymbol{y}_{(:)}\Big\|_2+\varepsilon\|\mathcal{Y}\|\sqrt{\prod_{i=1}^{N}R_i}.$$

Proof. It is straightforward to show that $\boldsymbol{A}^*$ solves

$$\boldsymbol{A}^*=\underset{\boldsymbol{A}\in\mathbb{R}^{I_n\times R_n}}{\arg\min}\Big\|\Big(\bigotimes_{\substack{i=N\\i\neq n}}^{1}\boldsymbol{A}^{(i)}\Big)\boldsymbol{G}_{(n)}^\top\boldsymbol{A}^\top-\boldsymbol{X}_{(n)}^\top\Big\|_F^2;$$

see e.g. the discussion in Subsection 4.2 in [116]. Setting the gradient of the objective function equal to zero and rearranging, we get that

$$\boldsymbol{G}_{(n)}\boldsymbol{G}_{(n)}^\top\boldsymbol{A}^{*\top}=\boldsymbol{G}_{(n)}\Big(\bigotimes_{\substack{i=N\\i\neq n}}^{1}\boldsymbol{A}^{(i)}\Big)^\top\boldsymbol{Y}_{(n)}^\top. \tag{2.10}$$

By a similar argument, we find

$$\boldsymbol{G}_{(n)}\boldsymbol{G}_{(n)}^\top\tilde{\boldsymbol{A}}^\top=\boldsymbol{G}_{(n)}\Big(\bigotimes_{\substack{i=N\\i\neq n}}^{1}\boldsymbol{A}^{(i)}\Big)^\top\boldsymbol{T}^{(n)\top}\boldsymbol{T}^{(n)}\boldsymbol{Y}_{(n)}^\top.$$

The matrix $\tilde{\boldsymbol{A}}^\top$ is therefore the solution to a perturbed version of the system in (2.10). Using Lemma B.1 in [64], we get that the perturbation satisfies

$$\begin{aligned}&\Big\|\boldsymbol{G}_{(n)}\Big(\bigotimes_{\substack{i=N\\i\neq n}}^{1}\boldsymbol{A}^{(i)}\Big)^\top\boldsymbol{Y}_{(n)}^\top-\boldsymbol{G}_{(n)}\Big(\bigotimes_{\substack{i=N\\i\neq n}}^{1}\boldsymbol{A}^{(i)}\Big)^\top\boldsymbol{T}^{(n)\top}\boldsymbol{T}^{(n)}\boldsymbol{Y}_{(n)}^\top\Big\|_F\\&\leq\varepsilon\|\mathcal{G}\|\Big\|\bigotimes_{\substack{i=N\\i\neq n}}^{1}\boldsymbol{A}^{(i)}\Big\|_F\|\mathcal{Y}\|\end{aligned}$$

with probability at least $1-\delta$ if $J_1\geq(2+3^{N-1})/(\varepsilon^2\delta)$. Standard sensitivity analysis in linear algebra (see e.g. Subsection 2.6 in [83]) now gives that

$$\|\boldsymbol{A}^{*\top}-\tilde{\boldsymbol{A}}^\top\|_F\leq\varepsilon\Big\|\Big(\boldsymbol{G}_{(n)}\boldsymbol{G}_{(n)}^\top\Big)^{-1}\Big\|_F\|\mathcal{G}\|\Big\|\bigotimes_{\substack{i=N\\i\neq n}}^{1}\boldsymbol{A}^{(i)}\Big\|_F\|\mathcal{Y}\|$$

with probability at least $1-\delta$ if $J_1 \geq (2+3^{N-1})/(\varepsilon^2\delta)$. It follows that

$$\begin{aligned}
&\Big\|\Big(\bigotimes_{\substack{i=N\\ i\neq n}}^{1} \boldsymbol{A}^{(i)}\Big)\boldsymbol{G}_{(n)}^{\top}\tilde{\boldsymbol{A}}^{\top} - \boldsymbol{Y}_{(n)}^{\top}\Big\|_F \\
&\leq \Big\|\Big(\bigotimes_{\substack{i=N\\ i\neq n}}^{1} \boldsymbol{A}^{(i)}\Big)\boldsymbol{G}_{(n)}^{\top}\boldsymbol{A}^{*\top} - \boldsymbol{Y}_{(n)}^{\top}\Big\|_F + \Big\|\bigotimes_{\substack{i=N\\ i\neq n}}^{1} \boldsymbol{A}^{(i)}\Big\|_F \|\mathcal{G}\| \|\boldsymbol{A}^{*\top} - \tilde{\boldsymbol{A}}^{\top}\|_F \\
&\leq \Big\|\Big(\bigotimes_{\substack{i=N\\ i\neq n}}^{1} \boldsymbol{A}^{(i)}\Big)\boldsymbol{G}_{(n)}^{\top}\boldsymbol{A}^{*\top} - \boldsymbol{Y}_{(n)}^{\top}\Big\|_F + \varepsilon\Big\|\Big(\boldsymbol{G}_{(n)}\boldsymbol{G}_{(n)}^{\top}\Big)^{-1}\Big\|_F \|\mathcal{G}\|^2 \|\mathcal{Y}\| \prod_{\substack{i=1\\ i\neq n}}^{N} R_i
\end{aligned}$$

with probability at least $1-\delta$ if $J_1 \geq (2+3^{N-1})/(\varepsilon^2\delta)$, and where we used the fact that $\bigotimes_{\substack{i=N\\ i\neq n}}^{1} \boldsymbol{A}^{(i)}$ is an orthogonal matrix of size $(\prod_{i\neq n} I_i) \times (\prod_{i\neq n} R_i)$ and therefore

$$\|\bigotimes_{\substack{i=N\\ i\neq n}}^{1} \boldsymbol{A}^{(i)}\|_F^2 = \prod_{\substack{i=1\\ i\neq n}}^{N} R_i.$$

For the other claim, Lemma B.1 in [64] gives that

$$\begin{aligned}
\|\boldsymbol{g}_{(:)}^{*} - \tilde{\boldsymbol{g}}_{(:)}\|_2 &= \Big\|\Big(\bigotimes_{i=N}^{1} \boldsymbol{A}^{(i)}\Big)^{\top}\boldsymbol{y}_{(:)} - \Big(\boldsymbol{T}^{(N+1)}\bigotimes_{i=N}^{1} \boldsymbol{A}^{(i)}\Big)^{\top}\boldsymbol{T}^{(N+1)}\boldsymbol{y}_{(:)}\Big\|_F \\
&\leq \varepsilon\Big\|\bigotimes_{i=N}^{1} \boldsymbol{A}^{(i)}\Big\|_F \|\mathcal{Y}\|
\end{aligned}$$

with probability at least $1-\delta$ if $J_2 \geq (2+3^{N})/(\varepsilon^2\delta)$. It follows that

$$\begin{aligned}
&\Big\|\Big(\bigotimes_{i=N}^{1} \boldsymbol{A}^{(i)}\Big)\tilde{\boldsymbol{g}}_{(:)} - \boldsymbol{y}_{(:)}\Big\|_2 \\
&\leq \Big\|\Big(\bigotimes_{i=N}^{1} \boldsymbol{A}^{(i)}\Big)\boldsymbol{g}_{(:)}^{*} - \boldsymbol{y}_{(:)}\Big\|_2 + \Big\|\bigotimes_{i=N}^{1} \boldsymbol{A}^{(i)}\Big\|_2 \|\boldsymbol{g}_{(:)}^{*} - \tilde{\boldsymbol{g}}_{(:)}\|_2 \\
&\leq \Big\|\Big(\bigotimes_{i=N}^{1} \boldsymbol{A}^{(i)}\Big)\boldsymbol{g}_{(:)}^{*} - \boldsymbol{y}_{(:)}\Big\|_2 + \varepsilon\|\mathcal{Y}\|\sqrt{\prod_{i=1}^{N} R_i},
\end{aligned}$$

where we again use the fact that $\bigotimes_{i=N}^{1} \boldsymbol{A}^{(i)}$ is a $(\prod_i I_i) \times (\prod_i R_i)$ orthogonal matrix, and its 2-norm therefore is 1, and its Frobenius norm is equal to $\sqrt{\prod_i R_i}$.

□

Algorithm 3: TUCKER-TTMTS (proposal)

/* Identical to Algorithm 1, except for the lines below */

1a Initialize $\boldsymbol{A}^{(2)}, \boldsymbol{A}^{(3)}, \ldots, \boldsymbol{A}^{(N)}$

1b Define TensorSketch operators $\boldsymbol{T}^{(n)} \in \mathbb{R}^{J_1 \times \prod_{i \neq n} I_i}$, for $n \in [N]$, and $\boldsymbol{T}^{(N+1)} \in \mathbb{R}^{J_2 \times \prod_i I_i}$

4 $\boldsymbol{Z}_{(n)} = \left(\boldsymbol{T}^{(n)} \boldsymbol{Y}_{(n)}^{\top}\right)^{\top} \left(\boldsymbol{T}^{(n)} \bigotimes_{i=N, i \neq n}^{1} \boldsymbol{A}^{(i)}\right)$

8 $\boldsymbol{g}_{(:)} = \left(\boldsymbol{T}^{(N+1)} \bigotimes_{i=N}^{1} \boldsymbol{A}^{(i)}\right)^{\top} \boldsymbol{T}^{(N+1)} \boldsymbol{y}_{(:)}$

2.3.3 Stopping conditions and orthogonalization

Unless stated otherwise, we stop after 50 iterations or when the change in $\|\mathcal{G}\|$ is less than 1e-3. The same type of convergence criteria are used in [117]. In Algorithm 2, we orthogonalize the factor matrices and update $\mathcal{G}$ using the reduced QR factorization. The computation

$$\begin{aligned} &\boldsymbol{Q}^{(n)} \boldsymbol{R}^{(n)} = \boldsymbol{A}^{(n)} \quad \text{reduced QR factorization} \\ &\boldsymbol{A}^{(n)} = \boldsymbol{Q}^{(n)} \\ &\mathcal{G} = \mathcal{G} \times_n \boldsymbol{R}^{(n)} \end{aligned}$$

ensures that each $\boldsymbol{A}^{(n)}$ is orthogonal, and that the Tucker decomposition remains unchanged. If we use the improvement in Remark 3 (c), we need to approximate $\mathcal{G}$. Please see Section S3.1 of the supplementary material of our paper [142] for details about this. In Algorithm 3, we compute an estimate of $\mathcal{G}$ using the same formula as in line 8, but using the smaller sketch dimension J_1 instead. Unlike in TUCKER-ALS, the objective is not guaranteed to decrease on each iteration of our algorithms. Despite this, the only practical different between our algorithms and TUCKER-ALS is that the tolerance may need to be set differently.

We would like to point out that we cannot provide global convergence guarantees for our algorithms. Although a global analysis would be desirable, it is important to note that such an analysis is difficult even for TUCKER-ALS. Indeed, TUCKER-ALS is not guaranteed to converge to the global optimum or even a stationary point (see Section 4.2 in [116]).

2.3.4 Complexity analysis

We compare the complexity of Algorithms 1–3, FSTD1 and MACH for the case $N = 3$. We assume that $I_n = I$ and $R_n = R < I$ for all $n \in [N]$. Furthermore, we assume that $J_1 = KR^{N-1}$ and $J_2 = KR^N$ for some constant $K > 1$, which is a choice that works well in practice. Table 2.1 shows the complexity when each of the variables I and R are assumed to be large. Please see Section S3.2 of the supplementary material of our paper [142] for a more detailed complexity analysis.

	Variable assumed to be large	
Algorithm	I = size of fiber	R = rank
T.-ALS (Alg. 1)	$(\#iter + 1) \cdot RI^3$	$(\#iter + 1) \cdot RI^3$
FSTD1 [42]	IR^4	R^5
MACH [199]	$(\#iter + 1) \cdot RI^3$	$(\#iter + 1) \cdot RI^3$
T.-TS (proposal, Alg. 2)	$\text{nnz}(\mathcal{Y}) + IR^4$	$R^3 + \#iter \cdot R^6$
T.-TTMTS (proposal, Alg. 3)	$\text{nnz}(\mathcal{Y}) + IR^4 + \#iter \cdot IR^4$	$R^6 + \#iter \cdot R^4$

Table 2.1: Leading order computational complexity, ignoring log factors and assuming $K = O(1)$, where $\#iter$ is the number of main loop iterations. $\mathcal{Y}$ is the 3-way data tensor we decompose. The main benefits of our proposed algorithms is reducing the $O(I^N)$ complexity of Algorithm 1 to $R^{O(N)}$ complexity due to the sketching, since typically $R \ll I$. The complexity of MACH is the same as that of TUCKER-ALS, but with a smaller constant factor.

2.4 Experiments

In this section we present results from experiments. Our Matlab implementation that we provided a link to at the beginning of Section 2.3 comes with demo script files for running experiments similar to those presented here. All synthetic results are averages over ten runs in an environment using four cores of an Intel Xeon E5-2680 v3 @2.50GHz CPU and 21 GB of RAM.

For Algorithms 2 and 3, the sketch dimensions J_1 and J_2 must be chosen. We have found that the choice $J_1 = KR^{N-1}$ and $J_2 = KR^N$, for a constant $K > 4$, works well in practice. Figures 2.1 and 2.2 respectively show examples of how the error of TUCKER-TS and TUCKER-TTMTS, in relation to that of TUCKER-ALS, changes with K. It also shows results for variants of each

algorithm for which the TensorSketch operator is redefined each time it is used (called "multi-pass" in the figures). For both algorithms, defining TensorSketch operators upfront leads to higher accuracy than redefining them before each application. In subsequent experiments, we always define the sketch operators upfront (i.e., as written in Algorithms 2 and 3) and, unless stated otherwise, always use $K = 10$.

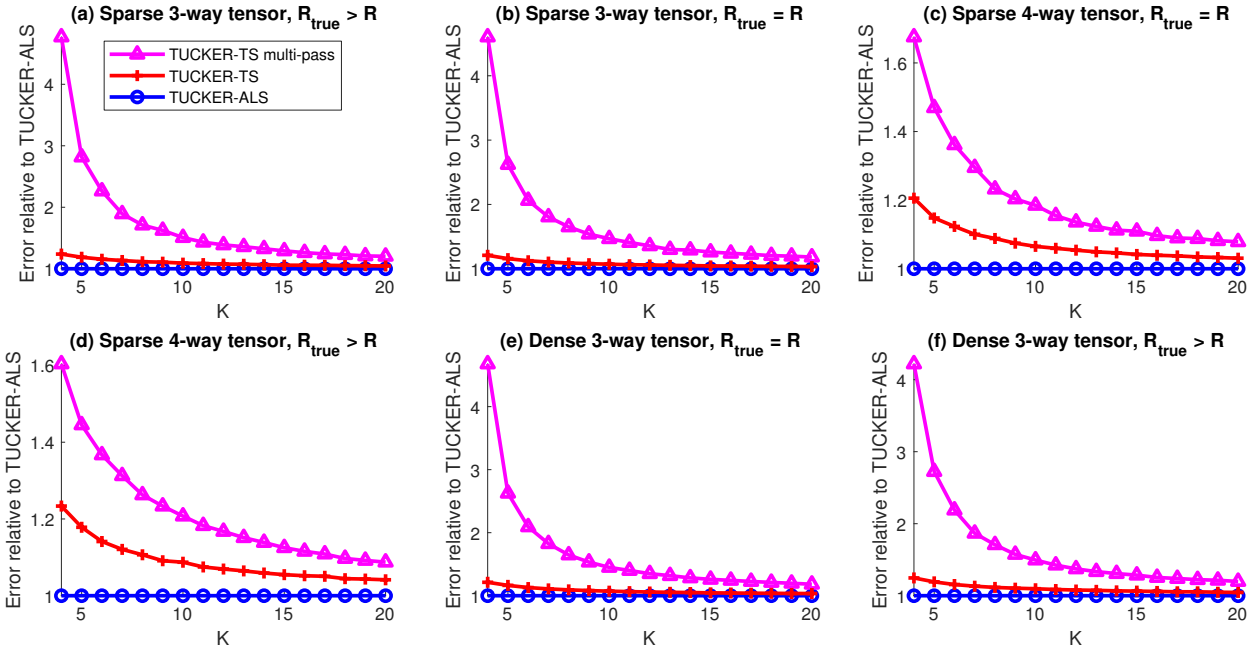

Figure 2.1: Error of TUCKER-TS relative to that of TUCKER-ALS for different values of the sketch dimension parameter K. For plot (a), the tensor size is $500 \times 500 \times 500$ with $\text{nnz}(\mathcal{Y}) \approx$ 1e+6 and true rank $(15, 15, 15)$. The algorithms use a target rank of $(10, 10, 10)$. For plot (b), the tensor size is $500 \times 500 \times 500$ with $\text{nnz}(\mathcal{Y}) \approx$ 1e+6, and with both the true and algorithm target rank equal to $(10, 10, 10)$. For plot (c), the tensor size is $100 \times 100 \times 100 \times 100$ with $\text{nnz}(\mathcal{Y}) \approx$ 1e+7, and with both the true and algorithm target rank equal to $(5, 5, 5, 5)$. For plot (d), the tensor size is $100 \times 100 \times 100 \times 100$ with $\text{nnz}(\mathcal{Y}) \approx$ 1e+7, with true rank $(10, 10, 10, 10)$ and algorithm target rank $(5, 5, 5, 5)$. For plot (e), the tensor is dense and of size $500 \times 500 \times 500$, and with both the true and algorithm target rank equal to $(10, 10, 10)$. For plot (f), the tensor is dense and of size $500 \times 500 \times 500$, and with true rank $(15, 15, 15)$ and algorithm target rank $(10, 10, 10)$.

2.4.1 Sparse synthetic data

In this subsection, we apply our algorithms to synthetic sparse tensors. For all synthetic data we use $I_n = I$ and $R_n = R$ for all $n \in [N]$. The sparse tensors are each created from a random

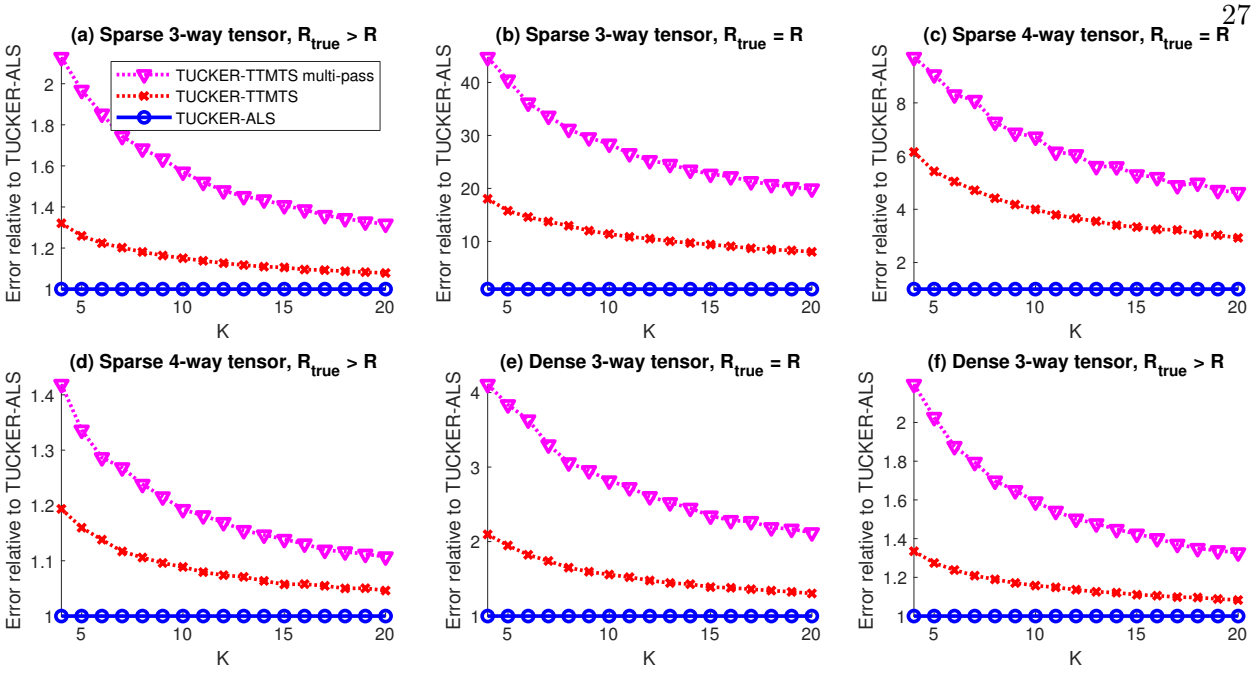

Figure 2.2: Error of TUCKER-TTMTS relative to that of TUCKER-ALS for different values of the sketch dimension parameter K. The experiment setup for the different subplots are the same as in Figure 2.1.

dense core tensor and random sparse factor matrices, where the sparsity of the factor matrices is chosen to achieve the desired sparsity of the tensor. We add i.i.d. normally distributed noise with standard deviation 1e-3 to all nonzero tensor elements.

Figures 2.3 and 2.4 show how the algorithms scale with **increased dimension** size I when three-way tensors are decomposed. Figures 2.5 and 2.6 show results from similar experiments with four-way tensors. Figure 2.7 and 2.8 show how the algorithms scale with tensor **density** and algorithm **target rank** R, respectively. TUCKER-ALS/MET and MACH run out of memory when $I =$ 1e+5 when decomposing both three-way and four-way tensors. FSTD1 is fast and scalable but inaccurate for very sparse tensors. The algorithm repeatedly finds indices of $\mathcal{Y}$ by identifying the element of maximum magnitude in fibers of the residual tensor. However, when $\mathcal{Y}$ is very sparse, it frequently happens that whole fibers in the residual tensor are zero. In those cases, the algorithm fails to find a good set of indices. This explains its poor accuracy in our experiments. We see that TUCKER-TS performs very well when $\mathcal{Y}$ truly is low-rank and we use that same rank for

reconstruction. TUCKER-TTMTS in general has a larger error than TUCKER-TS, but scales better with higher target rank. Moreover, when the true rank of the input tensor is greater than the target rank (Figures 2.4 and 2.6), which is closer to what real data might look like, the error of TUCKER-TTMTS is much closer to that of TUCKER-TS.

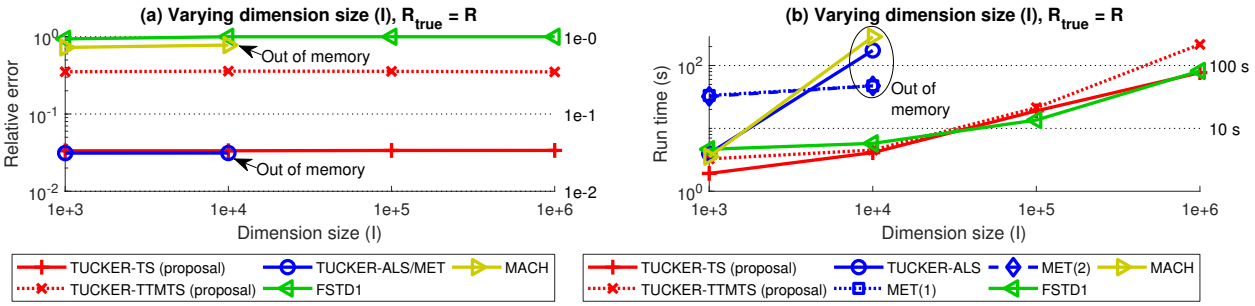

Figure 2.3: Relative error and run time for random sparse 3-way tensors with varying dimension size I and $\text{nnz}(\mathcal{Y}) \approx$ 1e+6. Both the true and target ranks are $(10, 10, 10)$.

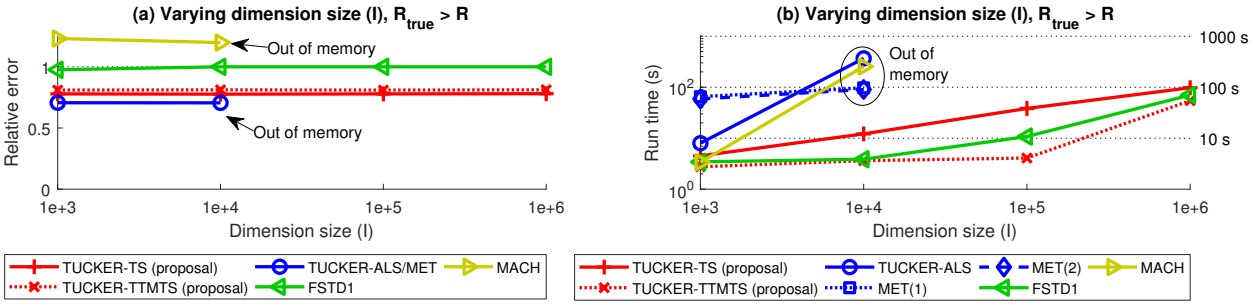

Figure 2.4: Relative error and run time for random sparse 3-way tensors with varying dimension size I and $\text{nnz}(\mathcal{Y}) \approx$ 1e+6. The true rank is $(15, 15, 15)$ and target rank is $(10, 10, 10)$. A convergence tolerance of 1e-1 is used for these experiments.

2.4.2 Dense real-world data

In this section we apply TUCKER-TTMTS to a real dense tensor representing a grayscale video. The video consists of 2,200 frames, each of size 1,080 by 1,980 pixels. The whole tensor, which requires 38 GB of RAM, is too large to load at the same time. Instead, it is loaded in pieces which are sketched and then added together. The video shows a natural scene, which is disturbed by a person passing by the camera twice. Since the camera is in a fixed position, we can expect this

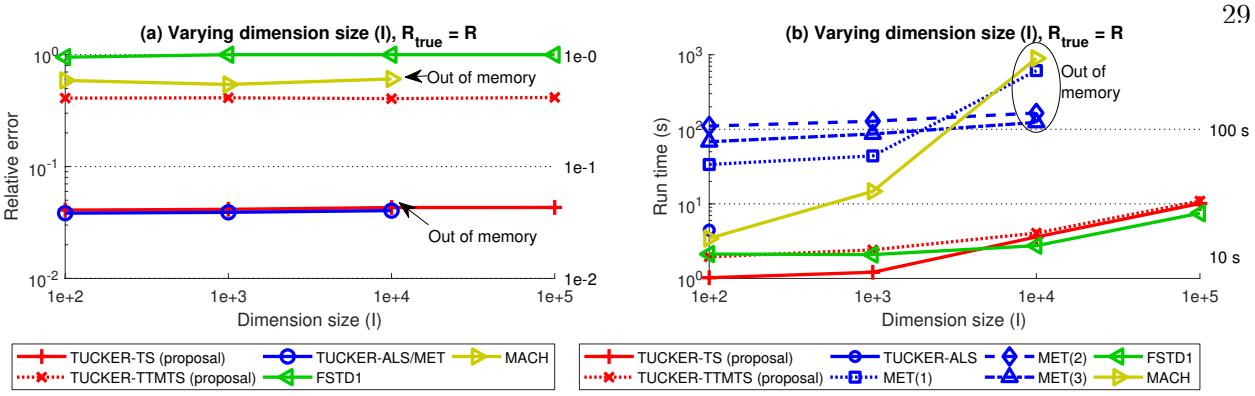

Figure 2.5: Comparison of relative error and run time for random sparse 4-way tensors. The number of nonzero elements is $\text{nnz}(\mathcal{Y}) \approx$ 1e+6. Both the true and target ranks are $(5, 5, 5, 5)$. We used $K = 10$ in these experiments.

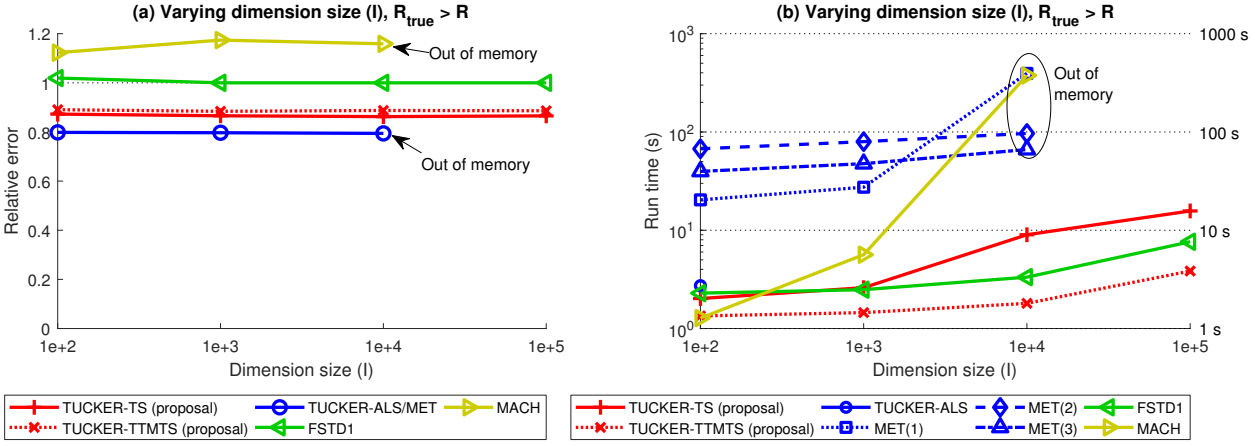

Figure 2.6: Comparison of relative error and run time for random sparse 4-way tensors. The number of nonzero elements is $\text{nnz}(\mathcal{Y}) \approx$ 1e+6. The true rank is $(7, 7, 7, 7)$ and the algorithm target rank is $(5, 5, 5, 5)$. In these experiments, we used $K = 10$ and a convergence tolerance of 1e-1.

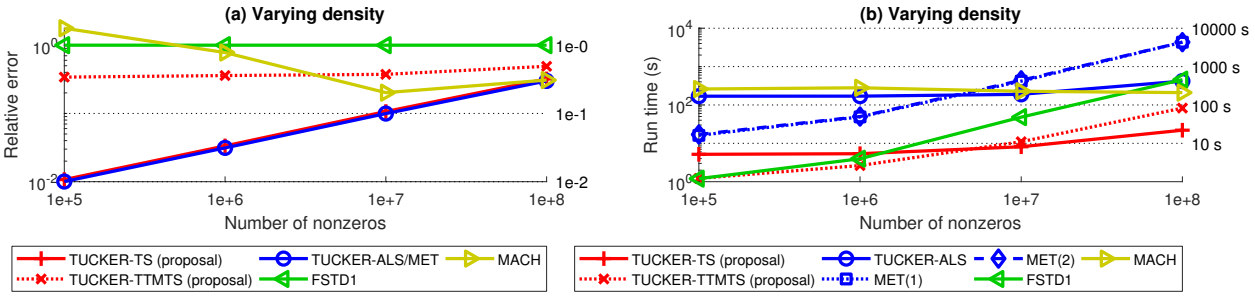

Figure 2.7: Relative error and run time for random sparse 3-way tensors with dimension size $I =$ 1e+4 and varying number of nonzeros. Both the true and target ranks are $(10, 10, 10)$.

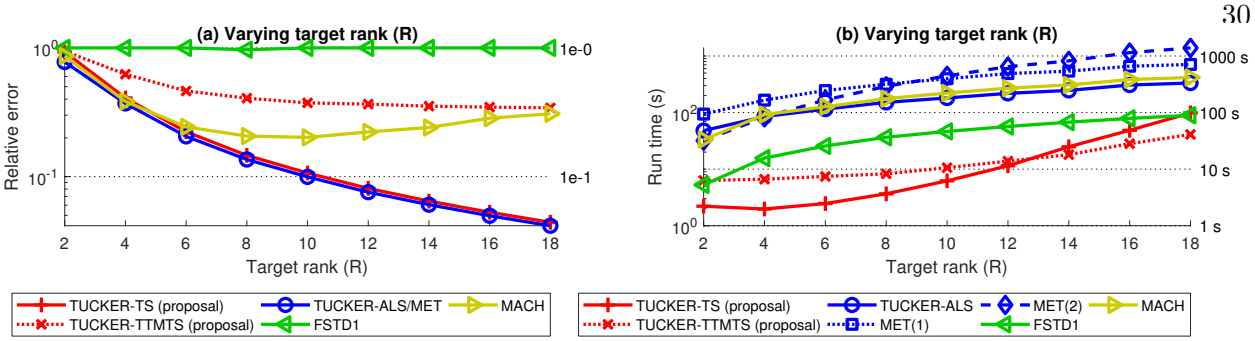

Figure 2.8: Relative error and run time for random sparse 3-way tensors with dimension size $I = 1\text{e}{+4}$ and $\text{nnz}(\mathcal{Y}) \approx 1\text{e}{+7}$. The true and target ranks are (R, R, R), with R varying.

tensor to be compressible. We compute a rank $(10, 10, 10)$ Tucker decomposition of the tensor using TUCKER-TTMTS with the sketch dimension parameter set to $K = 100$ and a maximum of 30 iterations. We then apply k-means clustering to the factor matrix $\boldsymbol{A}^{(3)} \in \mathbb{R}^{2200\times 10}$ corresponding to the time dimension, classifying each frame using the corresponding row in $\boldsymbol{A}^{(3)}$ as a feature vector. We find that using three clusters works better than using two. We believe this is due to the fact that the light intensity changes through the video due to clouds, which introduces a third time varying factor. Figure 2.9 shows five sample frames with the corresponding assigned clusters. With few exceptions, the frames which contain a disturbance are correctly grouped together into class 3 with the remaining frames grouped into classes 1 and 2. The video experiment is online and a link to it is provided at `https://github.com/OsmanMalik/tucker-tensorsketch`.

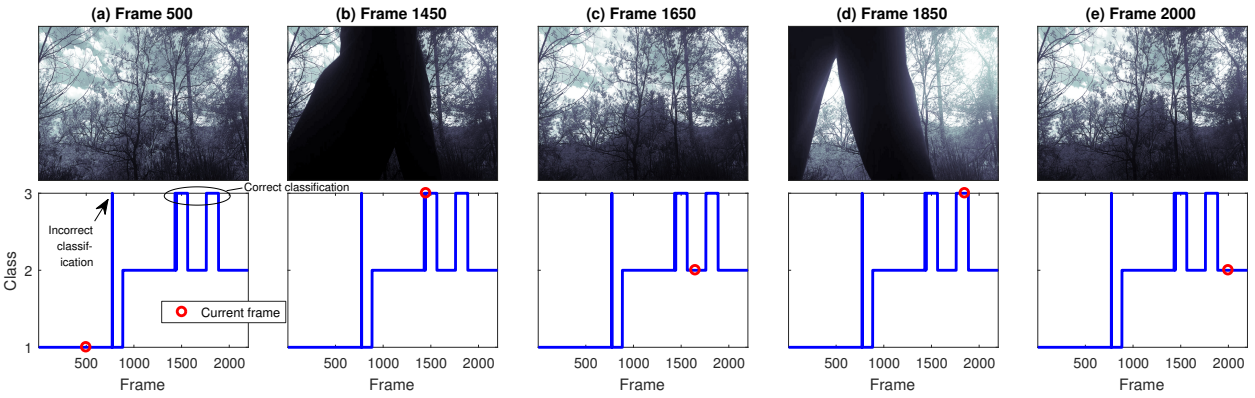

Figure 2.9: Five sample frames with their assigned classes. The frames (b) and (d) contain a disturbance.

2.5 Conclusion

We have proposed two algorithms for low-rank Tucker decomposition which incorporate TensorSketch and can handle streamed data. Experiments corroborate our complexity analysis which shows that the algorithms scale well both with dimension size and density. TUCKER-TS, and to a lesser extent TUCKER-TTMTS, scale poorly with target rank, so they are most useful when $R \ll I$.

Chapter 3

Guarantees for the Kronecker fast Johnson–Lindenstrauss transform using a coherence and sampling argument

3.1 Introduction

The Johnson–Lindenstrauss lemma, which was introduced by Johnson and Lindenstrauss [107], is the following fact.

Theorem 5 (Johnson–Lindenstrauss lemma [54])**.** *Let $\varepsilon \in (0,1)$ be a real number, let $\mathcal{X} \subseteq \mathbb{R}^I$ be a set of N points, and suppose $J \geq C\varepsilon^{-2} \log N$, where C is an absolute constant. Then there exists a map $f : \mathbb{R}^I \to \mathbb{R}^J$ such that for all $\boldsymbol{x}, \boldsymbol{y} \in \mathcal{X}$,*

$$(1-\varepsilon)\|\boldsymbol{x}-\boldsymbol{y}\|_2^2 \leq \|f(\boldsymbol{x})-f(\boldsymbol{y})\|_2^2 \leq (1+\varepsilon)\|\boldsymbol{x}-\boldsymbol{y}\|_2^2.$$

Any mapping f which has this property is called a Johnson–Lindenstrauss transform. Typically, such transforms are random maps, which motivates the following, more precise, definition.

Definition 6 (Johnson–Lindenstrauss transform [212])**.** A probability distribution on a family of maps $\mathcal{F}$, where each $f \in \mathcal{F}$ maps $\mathcal{Y} \subseteq \mathbb{R}^I$ to $\mathbb{R}^J$, is a **Johnson–Lindenstrauss transform** with parameters ε, δ, and N, or JLT(ε, δ, N), on $\mathcal{Y}$ if, for any subset $\mathcal{X} \subseteq \mathcal{Y}$ containing N elements, the probability of drawing a map $f \in \mathcal{F}$ which satisfies

$$(\forall \boldsymbol{x}, \boldsymbol{y} \in \mathcal{X}) \quad (1-\varepsilon)\|\boldsymbol{x}-\boldsymbol{y}\|_2^2 \leq \|f(\boldsymbol{x})-f(\boldsymbol{y})\|_2^2 \leq (1+\varepsilon)\|\boldsymbol{x}-\boldsymbol{y}\|_2^2$$

is at least $1-\delta$. Following common usage, we will refer to a random map as a JLT when the corresponding distribution satisfies this definition.

JLTs are usually constructed using simple random matrices, such as Gaussians with i.i.d. entries. They have many uses in applications, such as nearest neighbor searching [3], least squares regression [8, 72], sketching of data streams [212], and clustering [141].

When the vectors in the set $\mathcal{Y}$ in Definition 6 have special structure, it is possible to construct a map f that leverages this fact to speed up the computation of $f(\boldsymbol{x})$ when $\boldsymbol{x} \in \mathcal{Y}$. One class of vectors with such special structure are the Kronecker vectors $\boldsymbol{x} = \boldsymbol{x}^{(1)} \otimes \boldsymbol{x}^{(2)} \otimes \cdots \otimes \boldsymbol{x}^{(P)}$, where each $\boldsymbol{x}^{(p)} \in \mathbb{R}^{I_p}$ and $\otimes$ denotes the Kronecker product. Vectors with Kronecker structure appear in various applications. When matricizing tensors in CP or Tucker format, the resulting matrices have columns which are Kronecker products. Computation with Kronecker vectors therefore feature in algorithms for computing these decompositions [116] and in related problems like tensor interpolative decomposition [22]. They also arise in areas such as higher dimensional numerical analysis [20, 21], tensor regression [64], and polynomial kernel approximation in machine learning [174]. The Kronecker fast Johnson–Lindenstrauss transform (KFJLT) is a map that can be applied very efficiently to Kronecker structured vectors. It was first proposed by Battaglino et al. [15] for solving the least squares problems that arise when computing the CP decomposition of tensors. Battaglino et al. [15] conjectured that the KFJLT is a JLT, but did not provide a proof. Recently, Jin et al. [105] provided a proof that the KFJLT indeed is a JLT.

In this paper, we provide an alternative proof of this fact for when the KFJLT is applied to Kronecker vectors, which is based on a coherence and sampling argument. As a stepping stone to proving our result, we also show that the KFJLT is an oblivious subspace embedding for matrices whose columns have Kronecker structure. Some ideas that we use in our proof were mentioned in [15]. Our guarantees are slightly different than those given in [105]: Ours have a worse dependence on the ambient dimensions $I_1, I_2, \ldots, I_P$ of the input vectors, but have a better dependence on the accuracy parameter ε. The two bounds can be combined into one which yields a better bound overall. Another distinction between [105] and our paper is that the result in [105] shows that the KFJLT is a JLT on vectors with arbitrary structure, whereas our result is restricted to vectors with Kronecker structure. This means that the guarantees in [105] will be applicable in situations when ours are

not. For example, KFJLT could be used instead of a standard fast JLT for sketching arbitrary vectors in order to reduce the number of random bits required to construct the sketch. However, in certain applications involving arbitrary vectors our guarantees on Kronecker vectors are sufficient. For example, when applying a KFJLT sketch to the least squares problem $\min_{\boldsymbol{x}} \|\boldsymbol{A}\boldsymbol{x} - \boldsymbol{y}\|_2$, where $\boldsymbol{A}$ is a Khatri–Rao product and $\boldsymbol{y}$ is arbitrary, it turns out that our subspace embedding result combined with sampled approximate matrix multiplication ideas from [70] is sufficient for deriving guarantees; see Remark 17 for further details.

3.2 Other related work

As mentioned in the introduction, a JLT can be constructed in many different ways. A popular choice is $f(\boldsymbol{x}) \stackrel{\text{def}}{=} \boldsymbol{\Omega}\boldsymbol{x}/\sqrt{J}$, where $\boldsymbol{\Omega} \in \mathbb{R}^{J\times I}$ has i.i.d. standard normal entries. More generally, the rows of $\boldsymbol{\Omega}$ can be chosen to be independent, mean zero, isotropic and sub-Gaussian random vectors in $\mathbb{R}^I$ [205]. Ailon and Chazelle [3] proposed a fast JLT which leverages the Hadamard transform to achieve a transform that can be applied faster than a general dense matrix $\boldsymbol{\Omega}$.

A concept related to the JLT is subspace embedding.

Definition 7 (Subspace embedding [212])**.** A $(1 \pm \varepsilon)$ ℓ_2**-subspace embedding** for the column space of a matrix $\boldsymbol{X} \in \mathbb{R}^{I\times R}$ is a matrix $\boldsymbol{M} \in \mathbb{R}^{J\times I}$ such that

$$(\forall \boldsymbol{z} \in \mathbb{R}^R) \quad (1-\varepsilon)\|\boldsymbol{X}\boldsymbol{z}\|_2^2 \le \|\boldsymbol{M}\boldsymbol{X}\boldsymbol{z}\|_2^2 \le (1+\varepsilon)\|\boldsymbol{X}\boldsymbol{z}\|_2^2. \tag{3.1}$$

We call a probability distribution on a family $\mathcal{F}$ of $J \times I$ matrices an (ε, δ) **oblivious** ℓ_2**-subspace embedding** for $I \times R$ matrices with columns in $\mathcal{Y} \subset \mathbb{R}^I$ if, for any matrix

$$\boldsymbol{X} = [\boldsymbol{x}_1,\ \boldsymbol{x}_2, \ldots,\ \boldsymbol{x}_R] \quad \text{with} \quad \boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_R \in \mathcal{Y},$$

the probability of drawing a matrix $\boldsymbol{M} \in \mathcal{F}$ satisfying (3.1) is at least $1 - \delta$.[1] Following common usage, we will refer to a random matrix as an oblivious subspace embedding when the corresponding distribution satisfies this definition. Unless specified otherwise, it is assumed that $\mathcal{Y} = \mathbb{R}^I$.

[1] In the definition of oblivious subspace embedding in Definition 2.2 of [212], the matrix $\boldsymbol{X}$ can have any structure, which corresponds to $\mathcal{Y} = \mathbb{R}^I$. We find it convenient for our purposes to consider random mappings that are oblivious subspace embeddings for matrices with certain structure.

Thus, a subspace embedding distorts the squared length of a vector in the range of $\boldsymbol{X}$ by only a small amount. Methods for subspace embedding include leverage score sampling [137] and CountSketch [50]. Leverage score sampling is not an oblivious subspace embedding, since the sampling probabilities depend on $\boldsymbol{X}$. CountSketch, on the other hand, is an oblivious subspace embedding. Note that a subspace embedding is not necessarily a JLT. CountSketch, for example, is not a JLT [212]. For a more complete survey of work related to the JLT and subspace embedding, we refer the reader to the surveys in [138, 212].

For vectors with Kronecker structure, Sun et al. [193] propose the so called tensor random projection (TRP), whose transpose is a Khatri–Rao product of arbitrary random projection maps. They prove that TRP is a JLT in the special case when the TRP is constructed from two smaller random projections which have entries that are i.i.d. sub-Gaussians with zero mean and unit variance. The TRP idea is used in the earlier work [22] for tensor interpolative decomposition, but no guarantees are provided there. Rakhshan and Rabusseau [175] extend the TRP to allow for a wider range of structured sketches which incorporate CP tensor and tensor-train structure. They assume that the factor matrices and factor tensors for the CP tensor and tensor-train structures, respectively, follow a Gaussian distribution, and prove that their proposed sketches are JLTs. Notably, their results hold for arbitrary orders of the underlying CP tensors and tensor-trains.

Cheng et al. [46] propose an estimated leverage score sampling algorithm for ℓ_2-regression when the design matrix is a Khatri–Rao product. They use this to speed up the alternating least squares algorithm for computing the tensor CP decomposition. A similar idea is proposed by Diao et al. [65] for ℓ_2-regression when the design matrix is a Kronecker product.

The papers [9, 64, 170, 174] develop a method called TensorSketch, which is a variant of CountSketch that can be applied particularly efficiently to matrices whose columns have Kronecker structure. Avron et al. [9] show that TensorSketch is an oblivious subspace embedding, and Diao et al. [64] provide guarantees for ℓ_2-regression based on TensorSketch. However, just like CountSketch, TensorSketch is not a JLT.

A paper by Iwen et al. [102] considers structured linear embedding operators for tensors.

These operators first apply a sketch matrix to each mode of the tensor, then vectorize the result and apply another random sketch. Under certain assumptions on the tensor coherence and sketch matrix properties, they show that their proposed embedding operator is a form of tensor subspace embedding. Combining their approach with results from Jin et al. [105], they also consider a variant of the KFJLT with improved embedding properties. We make some comparisons between our results and those in [105] and [102] in Section 3.4.

3.3 Preliminaries

We use bold uppercase letters, e.g. $\boldsymbol{A}$, to denote matrices; bold lowercase letters, e.g. $\boldsymbol{a}$, to denote vectors; and regular lowercase letters, e.g. a, to denote scalars. Regular uppercase letters, e.g. I, J, K, are usually used to denote the size of vectors and matrices. This means that I is a number and not the identity matrix. Subscripts are used to denote elements of matrices, and a colon denotes all elements in a row or column. For example, for a matrix $\boldsymbol{A}$, $\boldsymbol{A}_{ij}$ is the element on position (i, j), $\boldsymbol{A}_{i:}$ is the ith row, and $\boldsymbol{A}_{:j}$ is the jth column. Subscripts may be used to label different vectors. Superscripts in parentheses will be used for labeling both matrices and vectors. For example, $\boldsymbol{A}^{(1)}$ and $\boldsymbol{A}^{(2)}$ are two matrices. The norm $\|\cdot\|_2$ denotes the standard Euclidean norm for vectors, and the spectral norm for matrices. For matrices $\boldsymbol{A} \in \mathbb{R}^{I\times J}$ and $\boldsymbol{B} \in \mathbb{R}^{K\times L}$, their Kronecker product is denoted by $\boldsymbol{A} \otimes \boldsymbol{B} \in \mathbb{R}^{IK\times JL}$ and is defined as

$$\boldsymbol{A} \otimes \boldsymbol{B} \stackrel{\text{def}}{=} \begin{bmatrix} \boldsymbol{A}_{11}\boldsymbol{B} & \boldsymbol{A}_{12}\boldsymbol{B} & \cdots & \boldsymbol{A}_{1J}\boldsymbol{B} \\ \boldsymbol{A}_{21}\boldsymbol{B} & \boldsymbol{A}_{22}\boldsymbol{B} & \cdots & \boldsymbol{A}_{2J}\boldsymbol{B} \\ \vdots & \vdots & & \vdots \\ \boldsymbol{A}_{I1}\boldsymbol{B} & \boldsymbol{A}_{I2}\boldsymbol{B} & \cdots & \boldsymbol{A}_{IJ}\boldsymbol{B} \end{bmatrix}.$$

For matrices $\boldsymbol{A} \in \mathbb{R}^{I\times K}$ and $\boldsymbol{B} \in \mathbb{R}^{J\times K}$, their Khatri–Rao product is denoted by $\boldsymbol{A} \odot \boldsymbol{B} \in \mathbb{R}^{IJ\times K}$ and is defined as

$$\boldsymbol{A} \odot \boldsymbol{B} \stackrel{\text{def}}{=} \begin{bmatrix} \boldsymbol{A}_{:1} \otimes \boldsymbol{B}_{:1} & \boldsymbol{A}_{:2} \otimes \boldsymbol{B}_{:2} & \cdots & \boldsymbol{A}_{:K} \otimes \boldsymbol{B}_{:K} \end{bmatrix}.$$

For a positive integer n, we use the notation $[n] \stackrel{\text{def}}{=} \{1, 2, \ldots, n\}$. We let $\sigma_i(\boldsymbol{A})$ denote the ith singular value of the matrix $\boldsymbol{A}$.

We now introduce the different tools we use to prove our results.

Definition 8 (Randomized Hadamard transform [3])**.** Let $\boldsymbol{H} \in \mathbb{R}^{I\times I}$ be the normalized Hadamard transform, and let $\boldsymbol{D} \in \mathbb{R}^{I\times I}$ be a diagonal matrix with i.i.d. Rademacher random variables (i.e., equal to $+1$ or -1 with equal probability) on the diagonal. The $I \times I$ **randomized Hadamard transform** is defined as the random map $\boldsymbol{x} \mapsto \boldsymbol{HDx}$.

Definition 9 (Leverage score, coherence [212])**.** Let $\boldsymbol{A} \in \mathbb{R}^{I\times R}$ be a matrix, and let $\operatorname{col}(\boldsymbol{A})$ be a matrix of size $I \times \operatorname{rank}(\boldsymbol{A})$ whose columns form an orthonormal basis for $\operatorname{range}(\boldsymbol{A})$. Then

$$\ell_i(\boldsymbol{A}) \stackrel{\text{def}}{=} \|\operatorname{col}(\boldsymbol{A})_{i:}\|_2^2, \quad i \in [I],$$

is the ith **leverage score** of $\boldsymbol{A}$. The **coherence** of $\boldsymbol{A}$ is defined as

$$\mu(\boldsymbol{A}) \stackrel{\text{def}}{=} \max_{i\in[I]} \ell_i(\boldsymbol{A}).$$

The leverage scores, and consequently the coherence, do not depend on the particular basis chosen for the range of $\boldsymbol{A}$ [212], so these quantities are well-defined. The coherence satisfies $\operatorname{rank}(\boldsymbol{A})/I \leq \mu(\boldsymbol{A}) \leq 1$.

Definition 10 (Leverage score sampling [212])**.** Let $\boldsymbol{A} \in \mathbb{R}^{I\times R}$ and $p_i \stackrel{\text{def}}{=} \ell_i(\boldsymbol{A})/\operatorname{rank}(\boldsymbol{A})$ for all $i \in [I]$. Then $\boldsymbol{p} \stackrel{\text{def}}{=} [p_1, p_2, \ldots, p_I]$ is a probability distribution on $[I]$. Let $\boldsymbol{q} \stackrel{\text{def}}{=} [q_1, q_2, \ldots, q_I]$ be another probability distribution on $[I]$, and suppose that for some $\beta \in (0, 1]$ it satisfies $q_i \geq \beta p_i$ for all $i \in [I]$. Let $\boldsymbol{v} \in [I]^J$ be a random vector with independent elements satisfying $\mathbb{P}(\boldsymbol{v}_j = i) = q_i$ for all $(i, j) \in [I] \times [J]$. Let $\boldsymbol{\Omega} \in \mathbb{R}^{J\times I}$ and $\boldsymbol{R} \in \mathbb{R}^{J\times J}$ be a random sampling matrix and a diagonal rescaling matrix, respectively, defined as

$$\boldsymbol{\Omega}_{j:} \stackrel{\text{def}}{=} \boldsymbol{e}_{\boldsymbol{v}_j}^\top \quad \text{and} \quad \boldsymbol{R}_{jj} \stackrel{\text{def}}{=} \frac{1}{\sqrt{Jq_{\boldsymbol{v}_j}}}$$

for each $j \in [J]$, where $\boldsymbol{e}_i$ is the ith column of the $I \times I$ identity matrix. The **leverage score sampling** matrix $\boldsymbol{S}_{\boldsymbol{q}} \in \mathbb{R}^{J\times I}$ is then defined as $\boldsymbol{S}_{\boldsymbol{q}} \stackrel{\text{def}}{=} \boldsymbol{R\Omega}$, where the subscript indicates that the sampling is done according to the distribution $\boldsymbol{q}$.

Definition 11 (Kronecker fast Johnson–Lindenstrauss transform [105])**.** For each $p \in [P]$, let $\boldsymbol{H}^{(p)}\boldsymbol{D}^{(p)}$ be independent randomized Hadamard transforms[2] of size $I_p \times I_p$. The **Kronecker fast Johnson–Lindenstrauss transform** (KFJLT) of a vector $\boldsymbol{x} = \boldsymbol{x}^{(1)} \otimes \boldsymbol{x}^{(2)} \otimes \cdots \otimes \boldsymbol{x}^{(P)}$, with $\boldsymbol{x}^{(p)} \in \mathbb{R}^{I_p}$, is defined as

$$\boldsymbol{S}_{\boldsymbol{q}}\Big(\bigotimes_{p=1}^{P} \boldsymbol{H}^{(p)}\boldsymbol{D}^{(p)}\Big)\boldsymbol{x} = \boldsymbol{S}_{\boldsymbol{q}}\Big(\bigotimes_{p=1}^{P} \boldsymbol{H}^{(p)}\boldsymbol{D}^{(p)}\boldsymbol{x}^{(p)}\Big), \tag{3.2}$$

where $\boldsymbol{S}_{\boldsymbol{q}} \in \mathbb{R}^{J \times \tilde{I}}$ is a sampling matrix as in Definition 10 with $\boldsymbol{q}$ equal to the uniform distribution on $[\tilde{I}]$, where $\tilde{I} \stackrel{\text{def}}{=} I_1 I_2 \cdots I_P$. The equality in (3.2) follows from a basic property of the Kronecker product; see e.g. Lemma 4.2.10 in [96].

A benefit of the KFJLT is that the Kronecker structured vector does not have to be explicitly computed—it is sufficient to store the smaller vectors $\boldsymbol{x}^{(1)}, \boldsymbol{x}^{(2)}, \ldots, \boldsymbol{x}^{(P)}$. Another benefit is that each randomized Hadamard transform $\boldsymbol{H}^{(p)}\boldsymbol{D}^{(p)} \in \mathbb{R}^{I_p \times I_p}$ only costs $O(I_p \log I_p)$ to apply to $\boldsymbol{x}^{(p)}$.

Lemma 12 below is a variant of Lemma 3 in [72] but with an arbitrary probability of success. The proof is identical to that for Lemma 3 in [72]—which in turn follows similar reasoning as in the proof of Lemma 1 in [3]—but using an arbitrary failure probability η instead of $1/20$, combined with the definition of leverage score and coherence.

Lemma 12. *Let $\boldsymbol{A} \in \mathbb{R}^{I \times R}$ be a matrix and let $\boldsymbol{H}\boldsymbol{D}$ be the $I \times I$ randomized Hadamard transform. Then, with probability at least $1 - \eta$, the following holds:*

$$\mu(\boldsymbol{H}\boldsymbol{D}\boldsymbol{A}) \leq \frac{2R \ln(2IR/\eta)}{I}.$$

Lemma 13 below is a restated version of Theorem 3.3 in [46]. A similar statement is also made in Lemma 4 in [15].

Lemma 13. *For each $p \in [P]$, let $\boldsymbol{A}^{(p)} \in \mathbb{R}^{I_p \times R}$. Then*

$$\mu\Big(\bigodot_{p=1}^{P} \boldsymbol{A}^{(p)}\Big) \leq \prod_{p=1}^{P} \mu(\boldsymbol{A}^{(p)}).$$

[2] Jin et al. [105] use the discrete Fourier transform instead of the Hadamard transform in their definition.

Lemma 14 below is a slight restatement of Theorem 2.11 in [212], with a careful choice of the constant parameter[3] .

Lemma 14. *Let $\boldsymbol{A} \in \mathbb{R}^{I\times R}$ and assume $\varepsilon \in (0,1)$. Suppose*

$$J > \frac{8}{3}\frac{R\ln(2R/\eta)}{\beta\varepsilon^2} \tag{3.3}$$

and that $\boldsymbol{S_q} \in \mathbb{R}^{J\times I}$ is a leverage score sampling matrix as in Definition 10, where the β in that definition is the same as the β in (3.3)*. Then, with probability at least $1-\eta$, the following holds:*

$$(\forall i \in [\operatorname{rank}(\boldsymbol{A})]) \quad 1-\varepsilon \le \sigma_i^2(\boldsymbol{S_q}\operatorname{col}(\boldsymbol{A})) \le 1+\varepsilon.$$

3.4 Main results

We consider the set

$$\mathcal{Y} \stackrel{\text{def}}{=} \{\boldsymbol{x} \in \mathbb{R}^{\tilde{I}} \ : \ \boldsymbol{x} = \boldsymbol{x}^{(1)} \otimes \boldsymbol{x}^{(2)} \otimes \cdots \otimes \boldsymbol{x}^{(P)}, \text{ with } \boldsymbol{x}^{(p)} \in \mathbb{R}^{I_p} \text{ for each } p \in [P]\}$$

of Kronecker vectors. Theorem 15 shows that the KFJLT is an (ε,δ) oblivious ℓ_2-subspace embedding for matrices whose columns have Kronecker product structure when the embedding dimension J is sufficiently large.

Theorem 15. *Let $\boldsymbol{X} = [\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_R] \in \mathbb{R}^{\tilde{I}\times R}$ be a matrix with each column $\boldsymbol{x}_r = \bigotimes_{p=1}^{P} \boldsymbol{x}_r^{(p)} \in \mathcal{Y}$. For each $p \in [P]$, let $\boldsymbol{H}^{(p)}\boldsymbol{D}^{(p)}$ be independent randomized Hadamard transforms of size $I_p \times I_p$, and define*

$$\boldsymbol{\Phi} \stackrel{\text{def}}{=} \bigotimes_{p=1}^{P} \boldsymbol{H}^{(p)}\boldsymbol{D}^{(p)}.$$

Let $\boldsymbol{S_q} \in \mathbb{R}^{J\times\tilde{I}}$ be a sampling matrix as in Definition 10 with $\boldsymbol{q}$ equal to the uniform distribution, and assume $\varepsilon \in (0,1)$. If

$$J > \frac{8}{3}\cdot 2^P R^{P+1}\varepsilon^{-2}\ln\Big(\frac{2R(P+1)}{\delta}\Big)\prod_{p=1}^{P}\ln\Big(\frac{2I_pR(P+1)}{\delta}\Big), \tag{3.4}$$

then the following holds with probability at least $1-\delta$:

$$(\forall \boldsymbol{z} \in \mathbb{R}^R) \quad (1-\varepsilon)\|\boldsymbol{X}\boldsymbol{z}\|_2^2 \le \|\boldsymbol{S_q}\boldsymbol{\Phi}\boldsymbol{X}\boldsymbol{z}\|_2^2 \le (1+\varepsilon)\|\boldsymbol{X}\boldsymbol{z}\|_2^2.$$

[3] The statement in [212] has a constant 144 instead of 8/3. However, we found that 8/3 is sufficient under the assumption that $\varepsilon \in (0,1)$. The proof given in [212] otherwise remains the same.

Proof. If all columns of $\boldsymbol{X}$ are the zero vector, the claim is trivially true. So we now assume that at least one column of $\boldsymbol{X}$ is nonzero. Note that $\boldsymbol{X} = \bigodot_{p=1}^{P} \boldsymbol{X}^{(p)}$, where each $\boldsymbol{X}^{(p)} \stackrel{\text{def}}{=} [\boldsymbol{x}_1^{(p)},\ \boldsymbol{x}_2^{(p)}, \ldots,\ \boldsymbol{x}_R^{(p)}]$. By Lemma 12, for a fixed $p \in [P]$, the following holds with probability at least $1 - \eta$:

$$\mu(\boldsymbol{H}^{(p)}\boldsymbol{D}^{(p)}\boldsymbol{X}^{(p)}) \leq \frac{2R\ln(2I_pR/\eta)}{I_p}.$$

Hence, taking a union bound, the following holds with probability at least $1 - P\eta$:

$$(\forall p \in [P]) \quad \mu(\boldsymbol{H}^{(p)}\boldsymbol{D}^{(p)}\boldsymbol{X}^{(p)}) \leq \frac{2R\ln(2I_pR/\eta)}{I_p}.$$

Now applying Lemma 13, we have that the following holds with probability at least $1 - P\eta$:

$$\mu(\boldsymbol{\Phi}\boldsymbol{X}) = \mu\Big(\bigodot_{p=1}^{P} \boldsymbol{H}^{(p)}\boldsymbol{D}^{(p)}\boldsymbol{X}^{(p)}\Big) \leq \prod_{p=1}^{P} \mu(\boldsymbol{H}^{(p)}\boldsymbol{D}^{(p)}\boldsymbol{X}^{(p)}) \leq \frac{1}{\tilde{I}} \prod_{p=1}^{P} 2R\ln(2I_pR/\eta). \tag{3.5}$$

For $i \in [\tilde{I}]$, let $p_i = \ell_i(\boldsymbol{\Phi}\boldsymbol{X})/\operatorname{rank}(\boldsymbol{\Phi}\boldsymbol{X})$. Since $\boldsymbol{\Phi}$ is a Kronecker product of orthogonal matrices, $\boldsymbol{\Phi}$ is also orthogonal [203], and since $\boldsymbol{X}$ is nonzero, it follows that $\operatorname{rank}(\boldsymbol{\Phi}\boldsymbol{X}) \geq 1$, so p_i is well defined. Instead of sampling according to the unknown distribution $[p_1,\ p_2, \ldots,\ p_{\tilde{I}}]$, we sample according to the uniform distribution $\boldsymbol{q} = [q_1,\ q_2, \ldots,\ q_{\tilde{I}}]$. To get guarantees, we want to apply Lemma 14. To do this, we first need to find some $\beta \in (0, 1]$ such that

$$(\forall i \in [\tilde{I}]) \quad q_i = \frac{1}{\tilde{I}} \geq \beta p_i.$$

From (3.5), the following holds with probability at least $1 - P\eta$:

$$p_i = \frac{\ell_i(\boldsymbol{\Phi}\boldsymbol{X})}{\operatorname{rank}(\boldsymbol{\Phi}\boldsymbol{X})} \leq \mu(\boldsymbol{\Phi}\boldsymbol{X}) \leq \frac{1}{\tilde{I}} \prod_{p=1}^{P} 2R\ln(2I_pR/\eta).$$

Hence, choosing β such that

$$\beta^{-1} = \prod_{p=1}^{P} 2R\ln(2I_pR/\eta)$$

ensures that $q_i = 1/\tilde{I} \geq \beta p_i$ for all $i \in [\tilde{I}]$ with probability at least $1 - P\eta$. Let $\alpha \stackrel{\text{def}}{=} \operatorname{rank}(\boldsymbol{\Phi}\boldsymbol{X}) = \operatorname{rank}(\boldsymbol{X})$, and let $\boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}^\top = \boldsymbol{X}$ be the SVD of $\boldsymbol{X}$ with $\boldsymbol{U} \in \mathbb{R}^{\tilde{I}\times\alpha}$, $\boldsymbol{\Sigma} \in \mathbb{R}^{\alpha\times\alpha}$ and $\boldsymbol{V} \in \mathbb{R}^{R\times\alpha}$. Note that the columns of $\boldsymbol{\Phi}\boldsymbol{U}$ form an orthonormal basis for $\operatorname{range}(\boldsymbol{\Phi}\boldsymbol{X})$. Hence, we can choose $\operatorname{col}(\boldsymbol{\Phi}\boldsymbol{X}) = \boldsymbol{\Phi}\boldsymbol{U}$. Using Lemma 14, with $\boldsymbol{A} = \boldsymbol{\Phi}\boldsymbol{X} \in \mathbb{R}^{\tilde{I}\times R}$, it follows that if

$$J > \frac{8}{3} \cdot 2^P R^{P+1} \varepsilon^{-2} \ln(2R/\eta) \prod_{p=1}^{P} \ln(2I_pR/\eta), \tag{3.6}$$

then the following holds with probability at least $1-(P+1)\eta$:

$$(\forall i \in [\alpha]) \quad 1-\varepsilon \leq \sigma_i^2(\boldsymbol{S_q}\boldsymbol{\Phi}\boldsymbol{U}) \leq 1+\varepsilon.$$

By the minimax characterization of singular values (see e.g. Theorem 8.6.1 in [83]), it follows that

$$(\forall \boldsymbol{w} \in \mathbb{R}^{\alpha}) \quad (1-\varepsilon)\|\boldsymbol{w}\|_2^2 \leq \|\boldsymbol{S_q}\boldsymbol{\Phi}\boldsymbol{U}\boldsymbol{w}\|_2^2 \leq (1+\varepsilon)\|\boldsymbol{w}\|_2^2$$

holds with probability at least $1-(P+1)\eta$. In particular, for any $\boldsymbol{z} \in \mathbb{R}^R$, this is true for $\boldsymbol{w} = \boldsymbol{\Sigma}\boldsymbol{V}^\top\boldsymbol{z} \in \mathbb{R}^{\alpha}$. Consequently,

$$(\forall \boldsymbol{z} \in \mathbb{R}^R) \quad (1-\varepsilon)\|\boldsymbol{\Sigma}\boldsymbol{V}^\top\boldsymbol{z}\|_2^2 \leq \|\boldsymbol{S_q}\boldsymbol{\Phi}\boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}^\top\boldsymbol{z}\|_2^2 \leq (1+\varepsilon)\|\boldsymbol{\Sigma}\boldsymbol{V}^\top\boldsymbol{z}\|_2^2,$$

or equivalently,

$$(\forall \boldsymbol{z} \in \mathbb{R}^R) \quad (1-\varepsilon)\|\boldsymbol{X}\boldsymbol{z}\|_2^2 \leq \|\boldsymbol{S_q}\boldsymbol{\Phi}\boldsymbol{X}\boldsymbol{z}\|_2^2 \leq (1+\varepsilon)\|\boldsymbol{X}\boldsymbol{z}\|_2^2,$$

holds with probability at least $1-(P+1)\eta = 1-\delta$, where $\delta \stackrel{\text{def}}{=} (P+1)\eta$. Replacing $\eta = \delta/(P+1)$ in (3.6) gives (3.4). □

The following theorem is our main result. It shows that the KFJLT is a JLT(ε, δ, N) on $\mathcal{Y}$ when the embedding dimension J is sufficiently large.

Theorem 16. *Let $\mathcal{X} \subseteq \mathcal{Y}$ consist of N distinct vectors with Kronecker structure. Let $\boldsymbol{\Phi}$ be defined as in Theorem 15. Let $\boldsymbol{S_q} \in \mathbb{R}^{J\times\tilde{I}}$ be a sampling matrix as in Definition 10 with $\boldsymbol{q}$ equal to the uniform distribution, and assume $\varepsilon \in (0,1)$. If*

$$J > \frac{16}{3}\cdot 4^P\varepsilon^{-2}\ln\Big(\frac{4N^2(P+1)}{\delta}\Big)\prod_{p=1}^{P}\ln\Big(\frac{4I_pN^2(P+1)}{\delta}\Big), \tag{3.7}$$

then the following holds with probability at least $1-\delta$:

$$(\forall \boldsymbol{x}, \boldsymbol{y} \in \mathcal{X}) \quad (1-\varepsilon)\|\boldsymbol{x}-\boldsymbol{y}\|_2^2 \leq \|\boldsymbol{S_q}\boldsymbol{\Phi}\boldsymbol{x} - \boldsymbol{S_q}\boldsymbol{\Phi}\boldsymbol{y}\|_2^2 \leq (1+\varepsilon)\|\boldsymbol{x}-\boldsymbol{y}\|_2^2. \tag{3.8}$$

Proof. Fix $\boldsymbol{x}, \boldsymbol{y} \in \mathcal{X}$ and set $\boldsymbol{X} \stackrel{\text{def}}{=} [\boldsymbol{x},\ \boldsymbol{y}]$. From Theorem 15, we know that if

$$J > \frac{8}{3}\cdot 2^P 2^{P+1}\varepsilon^{-2}\ln\Big(\frac{4(P+1)}{\eta}\Big)\prod_{p=1}^{P}\ln\Big(\frac{4I_p(P+1)}{\eta}\Big), \tag{3.9}$$

then the following holds with probability at least $1-\eta$:

$$(\forall \boldsymbol{z} \in \mathbb{R}^2) \quad (1-\varepsilon)\|\boldsymbol{X}\boldsymbol{z}\|_2^2 \leq \|\boldsymbol{S_q}\boldsymbol{\Phi}\boldsymbol{X}\boldsymbol{z}\|_2^2 \leq (1+\varepsilon)\|\boldsymbol{X}\boldsymbol{z}\|_2^2.$$

In particular, setting $\boldsymbol{z} = [1,\ -1]^\top$, we have that, with probability at least $1-\eta$,

$$(1-\varepsilon)\|\boldsymbol{x}-\boldsymbol{y}\|_2^2 \leq \|\boldsymbol{S_q}\boldsymbol{\Phi}\boldsymbol{x} - \boldsymbol{S_q}\boldsymbol{\Phi}\boldsymbol{y}\|_2^2 \leq (1+\varepsilon)\|\boldsymbol{x}-\boldsymbol{y}\|_2^2.$$

Taking a union bound over all distinct N^2-N pairs $\boldsymbol{x}, \boldsymbol{y} \in \mathcal{X}$, we have that (3.8) holds with probability at least $1-N^2\eta = 1-\delta$, where $\delta \stackrel{\text{def}}{=} N^2\eta$. Replacing $\eta = \delta/N^2$ in (3.9) gives (3.7). $\square$

Assuming $N > \max\{P, 4\}$, the bound in (3.7) can be simplified to

$$J > C_1\varepsilon^{-2}C_2^P \log\Big(\frac{N}{\delta}\Big)\prod_{p=1}^{P}\log\Big(\frac{I_pN}{\delta}\Big), \tag{3.10}$$

where C_1 and C_2 are absolute constants. For comparison, and expressed in the same notation as in this paper, the bound on J in Theorem 2.1 in [105] needed to guarantee that the KFJLT is a JLT$(\varepsilon, \delta + 2^{-\Omega(\log \tilde{I})}, N)$ on $\mathbb{R}^{\tilde{I}}$ is of the form

$$J > C\varepsilon^{-2}\log^{2P-1}\Big(\frac{PN}{\delta}\Big)\log^4\Big(\varepsilon^{-1}\log^P\Big(\frac{PN}{\delta}\Big)\Big)\log\Big(\prod_{p=1}^{P} I_p\Big), \tag{3.11}$$

where C is an absolute constant. Whether (3.10) or (3.11) yield a better bound depends on the various parameters. For example, the bound in (3.11) has a nicer dependence on the dimension sizes $I_1, I_2, \ldots, I_P$ than the bound in (3.10). Indeed, the term

$$\log\Big(\prod_{p=1}^{P} I_p\Big) = \sum_{p=1}^{P}\log(I_p)$$

in (3.11) is a sum of logs of $I_1, I_2, \ldots, I_P$, whereas the term

$$\prod_{p=1}^{P}\log\Big(\frac{I_pN}{\delta}\Big)$$

in (3.10) is a product of logs of $I_1N/\delta, I_2N/\delta, \ldots, I_PN/\delta$. On the other hand, the bound in (3.10) has a nicer dependence on ε than (3.11) does. Indeed, (3.10) contains the term (ε^{-2}) whereas (3.11) contains the term $(\varepsilon^{-2}\log^4(\varepsilon^{-1}))$. These two bounds can therefore be combined to yield a better bound on the size of J required to ensure that the KFJLT is a JLT on $\mathcal{Y}$.

As noted by Iwen et al. [102], one of the intermediate embedding dimension results in Theorem 1 of their paper can be translated to a subspace embedding result of the same flavor as what we present in Theorem 15. Their result has a better dependence on R: It is proportional to R^2 while our bound in (3.4) is proportional to R^{P+1}. Moreover, their results hold for a large family of sketch matrices, whereas our result is limited to the KFJLT sketch. On the other hand, their bound has worse dependence on ε: It is proportional to ε^{-2P} while our bound is proportional to ε^{-2}. Their guarantees also require a coherence assumption. In Theorems 2 and 8 of their paper, they provide guarantees for a variant of KFJLT, with an intermediate embedding result that corresponds to a subspace embedding variant of Theorem 2.1 in [105].

Remark 17. Our coherence and sampling argument can also be used to provide guarantees for sketched least squares regression. Let $\boldsymbol{X}$, $\boldsymbol{\Phi}$ and $\boldsymbol{S_q}$ be defined as in Theorem 15 and suppose $\boldsymbol{X}$ is full rank, let $\boldsymbol{y} \in \mathbb{R}^{\tilde{I}}$ be an arbitrary vector with no assumptions on its structure, and let $\varepsilon \in (0, 1)$. One can show that if J is large enough, then $\|\boldsymbol{X}\hat{\boldsymbol{z}} - \boldsymbol{y}\|_2 \leq (1+\varepsilon)\mathrm{OPT}$, where $\hat{\boldsymbol{z}} \overset{\text{def}}{=} \arg\min_{\boldsymbol{z}} \|\boldsymbol{S_q}\boldsymbol{\Phi}(\boldsymbol{X}\boldsymbol{z} - \boldsymbol{y})\|_2$ and $\mathrm{OPT} \overset{\text{def}}{=} \min_{\boldsymbol{z}} \|\boldsymbol{X}\boldsymbol{z} - \boldsymbol{y}\|_2$. This can be done by following the same arguments as in the proof of Theorem 2 in Section 4 of [72]. Let $\boldsymbol{U}$ be the top R singular values of $\boldsymbol{X}$ and let $\boldsymbol{y}_\perp$ be the portion of $\boldsymbol{y}$ which is perpendicular to range($\boldsymbol{U}$). The proof boils down to showing that $\sigma_R^2(\boldsymbol{S_q}\boldsymbol{\Phi}\boldsymbol{U}) \geq 1/\sqrt{2}$ and $\|\boldsymbol{U}^\top(\boldsymbol{S_q}\boldsymbol{\Phi})^\top(\boldsymbol{S_q}\boldsymbol{\Phi})\boldsymbol{y}_\perp\|_2^2 \leq \varepsilon \cdot \mathrm{OPT}^2/2$ with high probability for sufficiently large J. The first statement follows directly from Theorem 15, and the second statement follows from Monte Carlo sampling results in [70] which require no information about $\boldsymbol{y}_\perp$. We refer to [72] for further details.

3.5 Numerical experiments

We present results from experiments on both synthetic and real-world data. These experiments were implemented in Matlab.[4]

[4] Our code is available online at `https://github.com/OsmanMalik/kronecker-sketching`.

3.5.1 Experiment 1: Synthetic data

In this section, we present the results from an experiment which compares five different sketches when applied to random Kronecker vectors with three different random distributions. The five methods we compare are the following.

- **Gaussian** sketch uses an unstructured $J \times \tilde{I}$ matrix with i.i.d. standard normal entries which are scaled by $1/\sqrt{J}$. This approach is not scalable, but interesting to use as a baseline in this experiment.
- **KFJLT** is the sketch discussed in this paper, and which is defined in Definition 11, with the only difference that the uniform sampling of rows is done without replacement.
- **TRP** is the method proposed in [193]. As sub-matrices, we use matrices with i.i.d. standard normal entries of size $I_p \times J$ and rescale appropriately.
- **TensorSketch** is the method developed in [9, 64, 170, 174].
- **Sampling** is the method proposed by [46]. It computes an estimate of the leverage scores for each row of the matrix to be sampled and uses these to compute a distribution which is used for sampling.

All of these methods, except the first, are specifically designed for sketching vectors with Kronecker structure. As input, we use random Kronecker vectors $\boldsymbol{x} = \bigotimes_{p=1}^{3} \boldsymbol{x}^{(p)}$, where each $\boldsymbol{x}^{(p)} \in \mathbb{R}^{16}$ has one of the following distributions.

- Each $\boldsymbol{x}^{(p)}$ has i.i.d. standard normal entries.
- Each $\boldsymbol{x}^{(p)}$ is sparse with three nonzero elements, which are independent and normally distributed with mean zero and standard deviation 100. The positions of the three nonzero elements are drawn uniformly at random without replacement.
- Each $\boldsymbol{x}^{(p)}$ contains a single nonzero entry, which is chosen uniformly at random. This nonzero entry is equal to 100.

Sparse Kronecker vectors are interesting in many data science applications, and arise in decomposition of sparse tensors, for example.

In the experiment, we draw two random vectors $\boldsymbol{x}, \boldsymbol{y} \in \mathbb{R}^{4096}$, each of which is a Kronecker product of three smaller vectors of length 16, drawn according to one of the three distributions above. For each of the five sketches f and for some embedding dimension J, we then compute how well they preserve the distance between $\boldsymbol{x}$ and $\boldsymbol{y}$ by computing the quantity

$$\left| \frac{\|f(\boldsymbol{x}) - f(\boldsymbol{y})\|_2}{\|\boldsymbol{x} - \boldsymbol{y}\|_2} - 1 \right|. \tag{3.12}$$

For each of the three distributions and each embedding dimension

$$J \in \{100, 200, \ldots, 1000\} \tag{3.13}$$

we repeat this 1000 times and compute the mean, standard deviation and maximum of (3.12) over those 1000 trials. For J as in (3.13), applying one of the sketches to $\boldsymbol{x}$ and $\boldsymbol{y}$ reduces the number of entries in those vectors by between 76% (for $J = 1000$) and 98% (for $J = 100$). Figures 3.1–3.3 present the results for each of the three distributions.

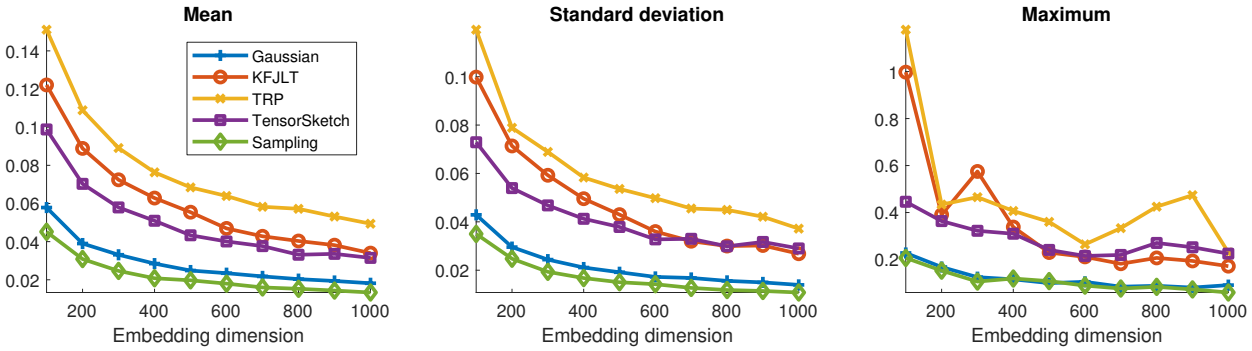

Figure 3.1: Mean, standard deviation, and maximum of the quantity in (3.12) over 1000 trials when the test vectors are Kronecker products of vectors with i.i.d. standard normal entries.

No one method produces the best results for all three distributions. The leverage score sampling approach does very well on dense vectors, even outperforming the Gaussian sketch, but does less well on sparser inputs. Although TensorSketch has an impressive mean performance on the two sparser inputs, it sometimes produces high distortion rates on those inputs. On the sparser

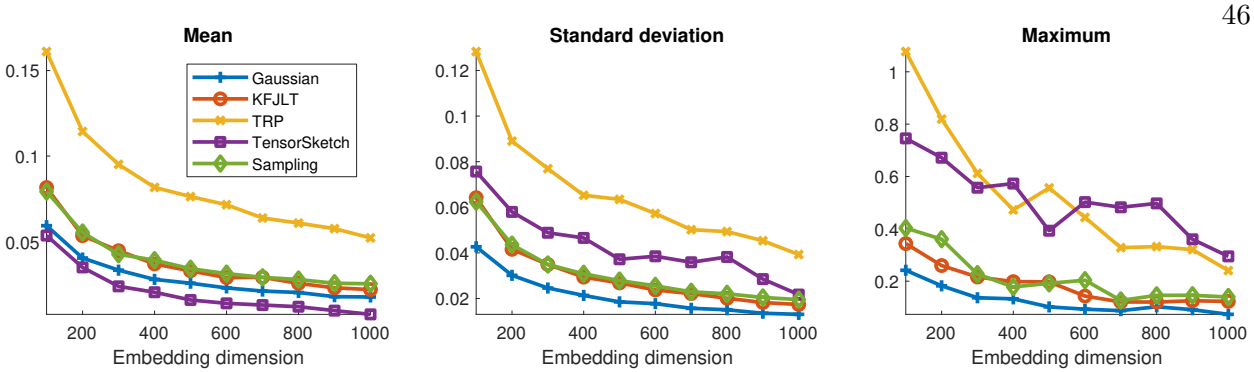

Figure 3.2: Mean, standard deviation, and maximum of the quantity in (3.12) over 1000 trials when the test vectors are Kronecker products of vectors with three nonzero elements which are independent and normally distributed with mean zero and standard deviation 100.

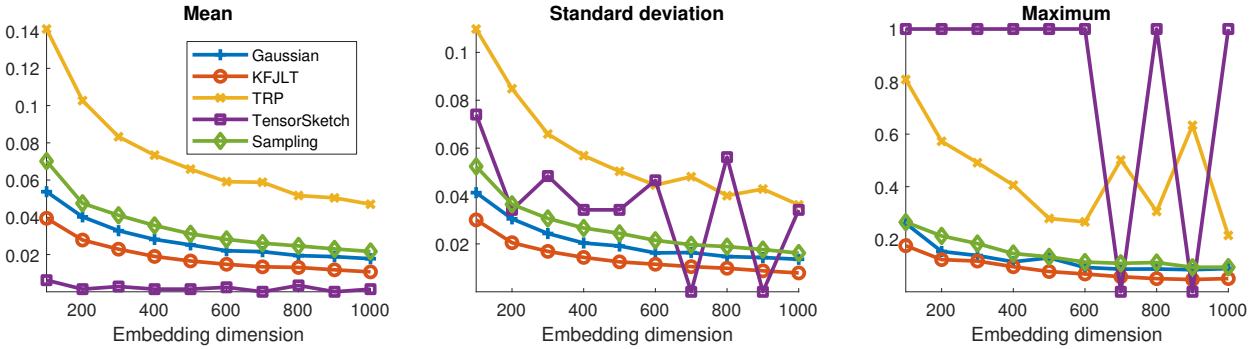

Figure 3.3: Mean, standard deviation, and maximum of the quantity in (3.12) over 1000 trials when the test vectors are Kronecker products of vectors containing a single nonzero entry equal to 100.

inputs, the KFJLT seems to strike the best balance between mean and worst case performance. TRP does poorly for all three distribution types.

3.5.2 Experiment 2: MNIST handwritten digits

In this experiment, we consider a subset of the MNIST Handwritten Digits dataset [125], which is a standard benchmark dataset in machine learning.[5] The dataset consists of images of handwritten digits between 0 and 9. Each image is in gray scale and of size 28 by 28 pixels. To make the image width and height powers of two, we pad the images with zeros so that their size is 32 by 32 pixels. We arrange 100 images depicting fours into a tensor $\mathcal{X} \in \mathbb{R}^{32\times32\times100}$ and 100

[5] We downloaded the MNIST dataset using the scripts provided at `https://github.com/sunsided/mnist-matlab`.

images depicting nines into another tensor $\mathcal{Y} \in \mathbb{R}^{32\times32\times100}$. Handwritten fours and nines can look quite similar and can be difficult to distinguish, which is why we choose this particular pair of digits. We then compute a rank-10 approximate CP decomposition of each tensor using `cp_als` in the Tensor Toolbox for Matlab [10, 11]. These take the form

$$\hat{\mathcal{X}} \stackrel{\text{def}}{=} \sum_{r=1}^{10} \boldsymbol{A}_{:r}^{(1)} \circ \boldsymbol{A}_{:r}^{(2)} \circ \boldsymbol{A}_{:r}^{(3)} \approx \mathcal{X},$$
$$\hat{\mathcal{Y}} \stackrel{\text{def}}{=} \sum_{r=1}^{10} \boldsymbol{B}_{:r}^{(1)} \circ \boldsymbol{B}_{:r}^{(2)} \circ \boldsymbol{B}_{:r}^{(3)} \approx \mathcal{Y},$$

where $\circ$ denotes outer product, and each $\boldsymbol{A}^{(1)}, \boldsymbol{B}^{(1)}, \boldsymbol{A}^{(2)}, \boldsymbol{B}^{(2)} \in \mathbb{R}^{32\times10}$ and $\boldsymbol{A}^{(3)}, \boldsymbol{B}^{(3)} \in \mathbb{R}^{100\times10}$ are called factor matrices; see [116] for further details on tensor decomposition. Notice that the factor matrices require much less storage than the original tensors. Figure 3.4 shows an example of a four and a nine in the MNIST dataset, and their corresponding approximations in the CP tensors.

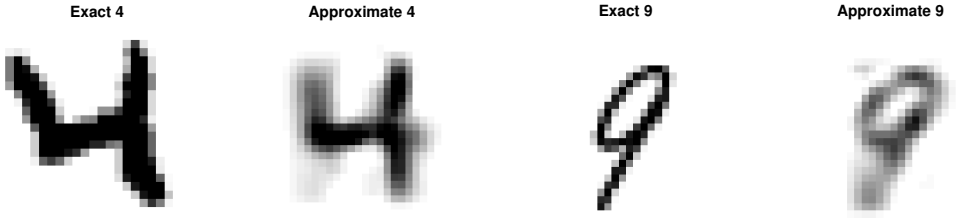

Figure 3.4: Example of a four and a nine with their corresponding approximations.

In this experiment, we apply the KFJLT, TRP, TensorSketch and sampling sketches to $\hat{\mathcal{X}}$ and $\hat{\mathcal{Y}}$ to see how well they preserve the distance between these tensors. The four sketches can be applied efficiently to $\hat{\mathcal{X}}$ and $\hat{\mathcal{Y}}$ in their decomposed form. To see this, define $\boldsymbol{X} \stackrel{\text{def}}{=} \boldsymbol{A}^{(1)} \odot \boldsymbol{A}^{(2)} \odot \boldsymbol{A}^{(3)} \in \mathbb{R}^{102400\times10}$ and $\boldsymbol{Y} \stackrel{\text{def}}{=} \boldsymbol{B}^{(1)} \odot \boldsymbol{B}^{(2)} \odot \boldsymbol{B}^{(3)} \in \mathbb{R}^{102400\times10}$, and let $\boldsymbol{u} \in \mathbb{R}^{20}$ denote a column vector with elements $\boldsymbol{u}_i = 1$ if $1 \le i \le 10$ and $\boldsymbol{u}_i = -1$ if $11 \le i \le 20$. Then the following relation holds:

$$\|\hat{\mathcal{X}} - \hat{\mathcal{Y}}\|_{\text{F}} = \|[\boldsymbol{X}, \boldsymbol{Y}]\,\boldsymbol{u}\|_2,$$

where $\|\cdot\|_{\text{F}}$ denotes the tensor Frobenius norm. Since the columns of $[\boldsymbol{X}, \boldsymbol{Y}]$ are Kronecker products, each of the four sketches under consideration can be applied efficiently to this matrix. The Gaussian sketch requires too much memory and is therefore not considered.

For any sketch $\boldsymbol{M}$ with the property

$$\|[\boldsymbol{X}, \boldsymbol{Y}]\,\boldsymbol{u}\|_2 \approx \|\boldsymbol{M}\,[\boldsymbol{X}, \boldsymbol{Y}]\,\boldsymbol{u}\|_2, \tag{3.14}$$

we may use $\|\boldsymbol{M}\,[\boldsymbol{X}, \boldsymbol{Y}]\,\boldsymbol{u}\|_2$ as an estimate for $\|\hat{\boldsymbol{\mathcal{X}}} - \hat{\boldsymbol{\mathcal{Y}}}\|_{\mathrm{F}}$. In the case of KFJLT, a guarantee of the form (3.14) follows from Theorem 15 when J is large enough. For each of the four sketches and some embedding dimension J, we compute the quantity

$$\left|\frac{\|\boldsymbol{M}\,[\boldsymbol{X}, \boldsymbol{Y}]\,\boldsymbol{u}\|_2}{\|\hat{\boldsymbol{\mathcal{X}}} - \hat{\boldsymbol{\mathcal{Y}}}\|_{\mathrm{F}}} - 1\right| \tag{3.15}$$

as a measure of performance. For each embedding dimension

$$J \in \{100, 200, \cdots, 5000\} \tag{3.16}$$

we repeat this 1000 times and compute the mean, standard deviation and maximum of (3.15) over those 1000 trials. The pair $(\hat{\boldsymbol{\mathcal{X}}}, \hat{\boldsymbol{\mathcal{Y}}})$ remains the same in all trials. For J as in (3.16), applying one of the sketches to $[\boldsymbol{X}, \boldsymbol{Y}]$ reduces the number of rows by between 95% (for $J = 5000$) and 99.9% (for $J = 100$). Figure 3.5 presents the results of the experiment. All methods have similar performance.

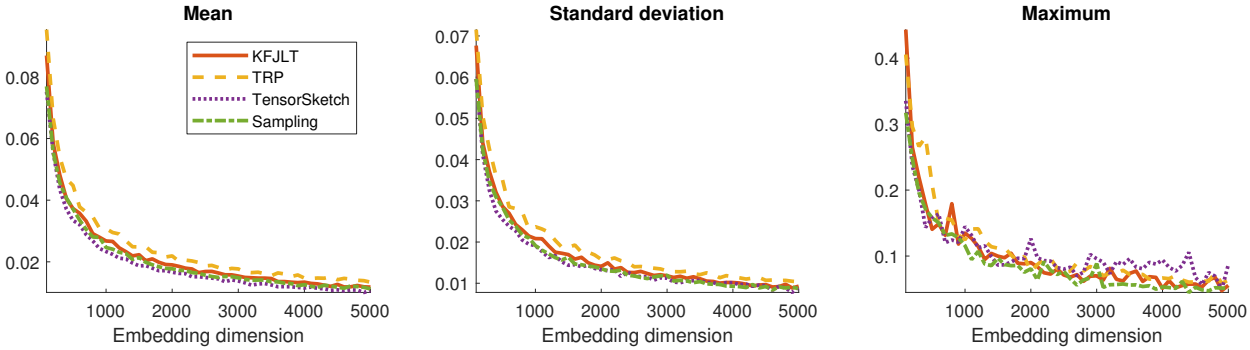

Figure 3.5: Mean, standard deviation, and maximum of the quantity in (3.15) over 1000 trials.

The fact that the factor matrices are mostly dense (some rows are zero due to the padding) may explain why the occasional large errors for TensorSketch observed in Figure 3.3 are avoided. Our results here indicate that the different sketches may have more similar performance on vectors like $[\boldsymbol{X}, \boldsymbol{Y}]\,\boldsymbol{u}$ which have less structure than the Kronecker vectors in the synthetic experiment. Jin et al. [105] made the related observation that KFJLT does a better job of embedding unstructured vectors than Kronecker structured ones; see Section 5.2 in their paper for further details.

3.6 Conclusion

We have presented a coherence and sampling argument for showing that the KFJLT is a Johnson–Lindenstrauss transform on vectors with Kronecker structure. Since our bound on the embedding dimension is different from the one in the recent paper by Jin et al. [105], it can be combined with the bound from that paper to yield a better bound overall. As a stepping stone to proving our result, we also showed that the KFJLT is a subspace embedding for matrices whose columns are Kronecker products.

We provided results from numerical experiments which compare five different sketches, four of which are designed to be particularly efficient for sketching of Kronecker structured vectors. The first experiment was done on Kronecker vectors with three different random distributions. The second experiment was done on two CP tensors, each approximating a tensor containing digits from the MNIST dataset. In the first experiment, there was a clear difference in performance between different sketches, although no single method outperformed all others for all three vector distributions. In the second experiment, all methods performed similarly except the unstructured Gaussian sketch which was not included in the experiment due to its high memory usage. We believe that there is a need for a more comprehensive comparison of sketches for structured data to help practitioners choose the best sketch for their particular needs.

Chapter 4

Fast randomized matrix and tensor interpolative decomposition using CountSketch

4.1 Introduction

Matrix decomposition is a fundamental tool used to compress and analyze data, and to improve the speed of computations. For data and computational problems involving more than two dimensions, analogous tools in the form of tensors and associated decompositions have been developed [116]. In many modern applications, matrices and tensors can be very large, which makes decomposing them especially challenging. One approach to dealing with this problems is to incorporate randomization in decomposition algorithms [91]. In this paper, we consider the interpolative decomposition (ID) for matrices, as well as the tensor ID problem. By tensor ID, we mean the tensor rank reduction problem as introduced by Biagioni et al. [22]; we provide an exact definition in Section 4.1.2.2. We make the following contributions in this paper:

- We propose a new fast randomized algorithm for matrix ID and provide theoretical performance guarantees.
- We propose a new randomized algorithm for tensor ID. To the best of our knowledge, we provide the first performance guarantees for any randomized tensor ID algorithm.
- We validate our algorithms on both synthetic and real data.
- We propose a small modification to the standard CountSketch formulation which helps avoid certain rank deficiency issues and slightly strengthen our matrix ID results.

4.1.1 Tensors and the CP decomposition

For a more complete introduction to tensors and their decompositions, see the review paper by Kolda and Bader [116]. A **tensor** $\mathcal{X} \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_N}$ is an N-dimensional array of real numbers, also called an N-way tensor. The number of elements in such a tensor is denoted by $\tilde{I} \stackrel{\text{def}}{=} \prod_{n=1}^{N} I_n$. Boldface Euler script letters, e.g. $\mathcal{X}$, denote tensors of dimension 3 or greater; bold capital letters, e.g. $\boldsymbol{X}$, denote matrices; bold lowercase letters, e.g. $\boldsymbol{x}$, denote vectors; and lowercase letters, e.g. x, denote scalars. Uppercase letters, e.g. I, are used to denote scalars indicating dimension size. A colon is used to denote all elements along a certain dimension. For example, $\boldsymbol{x}_{m:}$ and $\boldsymbol{x}_{:n}$ are the mth row and nth column of the matrix $\boldsymbol{X}$, respectively. If $\boldsymbol{j}$ is a vector of column indices, then $\boldsymbol{X}_{:\boldsymbol{j}}$ denotes the submatrix of $\boldsymbol{X}$ consisting of the columns of $\boldsymbol{X}$ whose indices are listed in $\boldsymbol{j}$. $\boldsymbol{I}^{(K)}$ denotes the $K \times K$ identity matrix. For a matrix $\boldsymbol{X}$, $\sigma_i(\boldsymbol{X})$ denotes its ith singular value, and $\sigma_{\max}(\boldsymbol{X})$ and $\sigma_{\min}(\boldsymbol{X})$ denote the maximum and minimum singular values, respectively. The condition number of $\boldsymbol{X}$ is defined as $\kappa(\boldsymbol{X}) \stackrel{\text{def}}{=} \sigma_{\max}(\boldsymbol{X})/\sigma_{\min}(\boldsymbol{X})$. The number of nonzero elements of $\boldsymbol{X}$ is denoted by $\operatorname{nnz}(\boldsymbol{X})$. For positive integers m and $n > m$, let $[m] \stackrel{\text{def}}{=} \{1, 2, \ldots, m\}$ and $[m:n] \stackrel{\text{def}}{=} \{m, m+1, \ldots, n\}$. The **Hadamard product**, or element-wise product, of matrices is denoted by $\circledast$. The **Khatri–Rao product** of matrices is denoted by $\odot$. The **tensor Frobenius norm** is denoted by $\|\mathcal{X}\|_{\mathrm{F}} \stackrel{\text{def}}{=} \|\operatorname{vec}(\mathcal{X})\|_2$, where $\operatorname{vec}(\mathcal{X})$ flattens the tensor $\mathcal{X}$ into a column vector. A norm $\|\cdot\|$ with no subscript will always denote the matrix spectral norm.

The singular value decomposition (SVD) decomposes matrices into a sum of rank-1 matrices [83]. Similarly, the **CP decomposition** decomposes a tensor $\mathcal{X} \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_N}$ into a sum of rank-1 tensors:

$$\mathcal{X} = \sum_{r=1}^{R} \lambda_r \boldsymbol{a}_{:r}^{(1)} \circ \boldsymbol{a}_{:r}^{(2)} \circ \cdots \circ \boldsymbol{a}_{:r}^{(N)} = \sum_{r=1}^{R} \lambda_r \mathcal{X}^{(r)}, \tag{4.1}$$

where $\circ$ denotes outer product, and each $\mathcal{X}^{(r)}$ is a rank-1 tensor. Each λ_r is called an **s-value**, each $\boldsymbol{A}^{(n)} = [\boldsymbol{a}_{:1}^{(n)} \;\; \boldsymbol{a}_{:2}^{(n)} \;\; \cdots \;\; \boldsymbol{a}_{:R}^{(n)}]$ is called a **factor matrix**, and all vectors $\boldsymbol{a}_{:r}^{(n)}$ have unit 2-norm. Usually, a tensor $\mathcal{X}$ is said to be of rank-R if R is the smallest possible number of terms required in a representation of the form (4.1). We will use the term "rank" in a looser sense to mean the (not

necessarily minimal) number of rank-1 terms in a representation of the form (4.1).

4.1.2 Interpolative decomposition

4.1.2.1 Matrix interpolative decomposition

For a matrix $\boldsymbol{A} \in \mathbb{R}^{I \times R}$, a rank-$K$ **interpolative decomposition** (ID) takes the form $\boldsymbol{A} \approx \boldsymbol{A}_{:\boldsymbol{j}}\boldsymbol{P}$, where $\boldsymbol{A}_{:\boldsymbol{j}} \in \mathbb{R}^{I \times K}$ consists of a subset of $K < R$ columns from $\boldsymbol{A}$, and $\boldsymbol{P} \in \mathbb{R}^{K \times R}$ is a coefficient matrix which is well-conditioned in some sense. The fact that the decomposition is expressed in terms of the columns of $\boldsymbol{A}$ means that $\boldsymbol{A}_{:\boldsymbol{j}}$ inherits properties such as sparsity and non-negativity from $\boldsymbol{A}$. Moreover, expressing the decomposition in terms of columns of $\boldsymbol{A}$ can increase interpretability. Algorithm 4 outlines one method to compute a matrix ID.

Algorithm 4: Matrix ID via QR [206]

input : $\boldsymbol{A} \in \mathbb{R}^{I \times R}$, target rank K
output: $\boldsymbol{P} \in \mathbb{R}^{K \times R}$, $\boldsymbol{j} \in [R]^K$

1 Perform rank-K QR factorization $\boldsymbol{A}\boldsymbol{\Pi} \approx \boldsymbol{Q}^{(1)}\boldsymbol{R}^{(1)}$
2 Define $\boldsymbol{j} \in [R]^K$ via $\boldsymbol{I}^{(R)}_{:\boldsymbol{j}} = \boldsymbol{\Pi}_{:[K]}$
3 Partition $\boldsymbol{R}^{(1)}$ into two parts: $\boldsymbol{R}^{(11)} = \boldsymbol{R}^{(1)}_{:[K]}$, $\boldsymbol{R}^{(12)} = \boldsymbol{R}^{(1)}_{:[K+1:R]}$
4 Compute $\boldsymbol{P}^\top = \boldsymbol{\Pi}\begin{bmatrix}\boldsymbol{I}^{(K)} & (\boldsymbol{R}^{(11)})^{-1}\boldsymbol{R}^{(12)\top}\end{bmatrix}$

Fact 18. If the partial QR factorization on line 3 in Algorithm 4 is done using the strongly rank-revealing QR (SRRQR) decomposition developed by Gu and Eisenstat [87], then Algorithm 4 has complexity $O(IR^2)$ [47]. Moreover, the decomposition it produces satisfies the following properties [150]:

(i) Some subset of the columns of $\boldsymbol{P}$ makes up the $K \times K$ identity matrix,

(ii) no entry of $\boldsymbol{P}$ has an absolute value exceeding 2,

(iii) $\|\boldsymbol{P}\| \leq \sqrt{4K(R-K)+1}$,

(iv) $\sigma_{\min}(\boldsymbol{P}) \geq 1$,

(v) $\boldsymbol{A}_{:\boldsymbol{j}}\boldsymbol{P} = \boldsymbol{A}$ when $K = I$ or $K = R$, and

(vi) $\|\boldsymbol{A}_{:\boldsymbol{j}}\boldsymbol{P} - \boldsymbol{A}\| \leq \sigma_{K+1}(\boldsymbol{A})\sqrt{4K(R-K)+1}$ when $K < \min(I, R)$.

In practice, using a variant of column pivoted QR instead of the SRRQR on line 3 of Algorithm 4 works just as well, and reduces the complexity of the algorithm to $O(KIR)$ [47].

There have been subsequent proposals for randomized versions of matrix ID [132]. Martinsson et al. [150] propose a variant which incorporates Gaussian random sketching. It computes a sketch $\boldsymbol{Y} = \boldsymbol{\Omega}\boldsymbol{A}$, where $\boldsymbol{\Omega} \in \mathbb{R}^{L\times I}$ $(K < L < I)$ is a matrix with iid standard normal entries, and then computes an ID $\boldsymbol{Y} \approx \boldsymbol{Y}_{:\boldsymbol{j}}\boldsymbol{P}$. The same $\boldsymbol{j}$ and $\boldsymbol{P}$ then give an ID of $\boldsymbol{A} \approx \boldsymbol{A}_{:\boldsymbol{j}}\boldsymbol{P}$. Woolfe et al. [213] propose a similar fast randomized algorithm which uses a subsampled randomized fast Fourier transform (SRFT) instead of a Gaussian matrix. It computes a sketch $\boldsymbol{Y} = \boldsymbol{S}_{\text{sub}}\boldsymbol{F}\boldsymbol{D}\boldsymbol{A}$, where $\boldsymbol{D} \in \mathbb{R}^{I\times I}$ is a diagonal matrix with each diagonal entry iid and equal to $+1$ or -1 with equal probability, $\boldsymbol{F} \in \mathbb{R}^{I\times I}$ is the fast Fourier transform (FFT), and $\boldsymbol{S}_{\text{sub}} \in \mathbb{R}^{L\times I}$ is a subsampling operator that randomly samples L rows.

4.1.2.2 Tensor interpolative decomposition

Biagioni et al. [22] consider the problem of rank reduction of a CP tensor, which they call tensor ID. Suppose $\boldsymbol{\mathcal{X}} \in \mathbb{R}^{I_1\times I_2\times\cdots\times I_N}$ is an N-way tensor with CP decomposition (4.1). Computing a rank-K, $K < R$, tensor ID of $\boldsymbol{\mathcal{X}}$ amounts to finding a representation

$$\hat{\boldsymbol{\mathcal{X}}} = \sum_{k=1}^{K} \hat{\lambda}_k \boldsymbol{\mathcal{X}}^{(j_k)} \approx \boldsymbol{\mathcal{X}}, \tag{4.2}$$

where $\boldsymbol{j} \in [R]^K$ contains K unique indices. Tensor ID has many applications. For example, in various algorithms, the rank of discretized separated representations of multivariate functions grows with each iteration, requiring repeated rank reduction of CP tensors [20, 21]. Another example is the algorithm by Reynolds et al. [180] for finding the element of maximum magnitude in a CP tensor which also requires repeated rank reduction.

Biagioni et al. [22] approach the tensor ID problem by considering the matrix

$$\boldsymbol{M} = \begin{bmatrix} \lambda_1 \text{vec}(\boldsymbol{\mathcal{X}}^{(1)}) & \cdots & \lambda_R \text{vec}(\boldsymbol{\mathcal{X}}^{(R)}) \end{bmatrix} = \left(\bigodot_{n=1}^{N} \boldsymbol{A}^{(n)} \right) \text{diag}(\lambda_1, \ldots, \lambda_R), \tag{4.3}$$

where $\text{diag}(\lambda_1, \ldots, \lambda_R) \in \mathbb{R}^{R \times R}$ is a diagonal matrix with entries $\lambda_1, \ldots, \lambda_R$. The tensor ID problem can now be reduced to identifying columns of $\boldsymbol{M}$ using matrix ID. However, when the factor matrices have no special structure, $\boldsymbol{M}$ has $R\tilde{I}$ elements and is therefore typically infeasible to form. One way to tackle this problem is by forming the much smaller Gram matrix $\boldsymbol{M}^\top \boldsymbol{M} \in \mathbb{R}^{R \times R}$, which can be done using $O(R^2 \sum_n I_n)$ flops since

$$\boldsymbol{M}^\top \boldsymbol{M} = (\boldsymbol{A}^{(1)\top} \boldsymbol{A}^{(1)}) \circledast \cdots \circledast (\boldsymbol{A}^{(N)\top} \boldsymbol{A}^{(N)}), \tag{4.4}$$

compute its symmetric matrix ID, and use it to compute an ID of $\boldsymbol{M}$. This approach, however, can lead to accuracy issues since the Gram matrix can be ill-conditioned, since $\kappa(\boldsymbol{M}^\top \boldsymbol{M}) = \kappa^2(\boldsymbol{M})$ [22]. Biagioni et al. [22] therefore propose a randomized method which avoids the ill-conditioning issue and reduces the complexity. This is done by applying a kind of Gaussian sketch to $\boldsymbol{M}$, but instead of forming a full Gaussian matrix of size $L \times \tilde{I}$, a matrix of the form

$$\boldsymbol{\Omega} = \left(\bigodot_{n=1}^{N} \boldsymbol{\Omega}^{(n)} \right)^\top \in \mathbb{R}^{L \times \tilde{I}}, \tag{4.5}$$

is used, where each $\boldsymbol{\Omega}^{(n)} \in \mathbb{R}^{I_n \times L}$ is a matrix with elements that are iid standard normal random variables. The sketch $\boldsymbol{Y} = \boldsymbol{\Omega} \boldsymbol{M}$ can then be computed efficiently without ever forming $\boldsymbol{\Omega}$ or $\boldsymbol{M}$, since $y_{lr} = \lambda_r \prod_{n=1}^{N} \langle \boldsymbol{\omega}_{:l}^{(n)}, \boldsymbol{a}_{:r}^{(n)} \rangle$. Note that the elements of $\boldsymbol{\Omega}$ in (4.5) are not independent. This means that the theory for Gaussian matrix ID, which requires independence, cannot be used to provide guarantees for sketched matrix ID using $\boldsymbol{\Omega}$.

4.1.3 Basics of CountSketch

Our proposed method uses a type of sketching called **CountSketch** [44, 50], which we now describe. Let $h : [I] \to [L]$ be a random map such that each $h(i)$ is iid and $(\forall i \in [I])(\forall l \in [L])$ $\mathbb{P}(h(i) = l) = 1/L$, let $\boldsymbol{\Phi} \in \mathbb{R}^{L \times I}$ be a matrix with $\phi_{h(i)i} = 1$ and all other entries equal to 0, and let $\boldsymbol{D} \in \mathbb{R}^{I \times I}$ be a diagonal matrix with each diagonal entry iid and equal to $+1$ or -1 with equal

probability. The CountSketch operator $\boldsymbol{S} \in \mathbb{R}^{L \times I}$ is then defined as $\boldsymbol{S} = \boldsymbol{\Phi}\boldsymbol{D}$. Applying $\boldsymbol{S}$ to $\boldsymbol{A} \in \mathbb{R}^{I \times R}$ does the following: The matrix $\boldsymbol{D}$ changes the sign of each row of $\boldsymbol{A}$ with probability $1/2$, and the matrix $\boldsymbol{\Phi}$ then randomly adds each row of $\boldsymbol{D}\boldsymbol{A}$ to one of L target rows. Due to the special structure of $\boldsymbol{S}$, it can be applied implicitly with complexity $O(\text{nnz}(\boldsymbol{A}))$ [50].

Suppose $\boldsymbol{A}$ has the special structure $\boldsymbol{A} = \bigodot_{n=1}^{N} \boldsymbol{A}^{(n)} \in \mathbb{R}^{\tilde{I} \times R}$, where each $\boldsymbol{A}^{(n)} \in \mathbb{R}^{I_n \times R}$. For such matrices, there is a variant of CountSketch which allows computing the sketch of $\boldsymbol{A}$ without ever having to form the full matrix, which can be prohibitively large to store explicitly. This variant is called TensorSketch and is developed by Pagh [170], Pham and Pagh [174], Avron et al. [9] and Diao et al. [64]. It works as follows:

- Define n independent random maps $h_n : [I_n] \to [L]$ such that each $h(i)$ is iid and $(\forall i \in [I_n])(\forall l \in [L])\ \mathbb{P}(h_n(i) = l) = 1/L$; and
- define n independent random sign functions $s_n : [I_n] \to \{+1, -1\}$ such that $(\forall i \in [I_n])$ $\mathbb{P}(s_n(i) = +1) = \mathbb{P}(s_n(i) = -1) = 1/2$.

Next, define $H : [I_1] \times [I_2] \times \cdots \times [I_N] \to [L]$ as

$$H(i_1, i_2, \ldots, i_N) \stackrel{\text{def}}{=} \Big(\sum_{n=1}^{N} (h_n(i_n) - 1) \mod L \Big) + 1,$$

and $S : [I_1] \times [I_2] \times \cdots \times [I_N] \to \{+1, -1\}$ as

$$S(i_1, i_2, \ldots, i_N) \stackrel{\text{def}}{=} \prod_{n=1}^{N} s_n(i_n).$$

Notice that each row index of $\boldsymbol{A}$ corresponds to a unique N-tuple $(i_1, \ldots, i_N)$. H and S can therefore be considered functions on $[\tilde{I}]$. With this in mind, let $\boldsymbol{D}_S \in \mathbb{R}^{\tilde{I} \times \tilde{I}}$ denote a diagonal matrix with the ith diagonal entry equal to $S(i)$. If H and $\boldsymbol{D}_S$ are used instead of h and $\boldsymbol{D}$ in the definition of CountSketch above, we get TensorSketch, which we will denote by $\boldsymbol{T} \in \mathbb{R}^{L \times \tilde{I}}$. The reason for choosing this formulation is that it can be computed efficiently using the following formula:

$$\boldsymbol{T}\boldsymbol{A} = \text{FFT}^{-1}\Big(\bigoplus_{n=1}^{N} \text{FFT}(\boldsymbol{S}^{(n)}\boldsymbol{A}^{(n)}) \Big), \tag{4.6}$$

where each $\boldsymbol{S}^{(n)} \in \mathbb{R}^{L \times I_n}$ is a CountSketch operator defined using h_n and the diagonal matrix $\text{diag}(s_n(1), \ldots, s_n(I_n))$. The formula (4.6) follows from the discussion in Section A in the supplementary material of Diao et al. [64]. Other good sources for further details on TensorSketch are Pagh [170], Pham and Pagh [174] and Avron et al. [9].

4.2 Other related work

We provided an overview of existing ID algorithms in Section 4.1.2. The matrix ID is related to the CX and CUR decompositions [23, 31, 67, 69, 71, 139, 208], also known as skeleton approximations [84, 85, 200], and the column subset selection problem [32, 33, 61, 62, 63, 79, 90]. Like ID, the CX decomposition takes the form $\boldsymbol{A} \approx \boldsymbol{C}\boldsymbol{X}$, where $\boldsymbol{C}$ contains a subset of the columns of $\boldsymbol{A}$. The crucial feature that distinguishes ID from a CX decomposition is the additional conditioning requirements on the coefficient matrix $\boldsymbol{P}$ in ID; the matrix $\boldsymbol{X}$ in a CX decomposition is not required to have the properties (i)–(iv) listed in Fact 18 [71]. A CUR decomposition takes the form $\boldsymbol{A} \approx \boldsymbol{C}\boldsymbol{U}\boldsymbol{R}$, where $\boldsymbol{C}$ and $\boldsymbol{R}$ contain a subset of the columns and rows of $\boldsymbol{A}$, respectively. Consequently, setting $\boldsymbol{X} = \boldsymbol{U}\boldsymbol{R}$ would yield a CX decomposition. It is well-known that the matrix $\boldsymbol{X}$ defined in this manner is typically ill-conditioned [206]. Since we require the coefficient matrix $\boldsymbol{P}$ in our decomposition to be well-conditioned, the available algorithms for CX and CUR decomposition are not useful to us.

Various randomized algorithms have been utilized in the context of tensor decomposition before. Examples include the works of Wang et al. [210], Battaglino et al. [15], and Yang et al. [215] for the CP decomposition; Drineas and Mahoney [68], Tsourakakis [199], da Costa et al. [53] and Malik and Becker [142] for the Tucker decomposition; and Zhang et al. [221] and Tarzanagh and Michailidis [196] for t-product based decompositions. Other notable works that use CUR-type algorithms or sampling are e.g. those by Mahoney et al. [140], Caiafa and Cichocki [42], Oseledets et al. [168] and Friedland et al. [78]. The tensor ID which we consider is different from the various problems solved in these previous papers. The goal of tensor ID is not to compute a tensor decomposition from an arbitrary data tensor. Instead, the purpose of tensor ID is to **compress a tensor which is already in CP format** in an efficient and principled manner. To the best of our

knowledge, the only work aside from that by Biagioni et al. [22] which considers randomized tensor ID is the paper by Reynolds et al. [179]. They introduce a randomized alternating least-squares (ALS) algorithm, which is better conditioned but slower than the standard ALS algorithm for CP decomposition. Biagioni et al. [22] conclude that standard ALS is much slower than their Gaussian sketching algorithm. We therefore do not compare our proposed tensor ID to the randomized ALS by Reynolds et al. [179] since it is even slower.

To the best of our knowledge, TensorSketch was the first sketch with theoretical guarantees that could be applied particularly efficiently to matrices like $\boldsymbol{M}$ in (4.3) with Kronecker structured columns. Recently, a number of works have appeared that provide guarantees for other methods designed for efficient sketching of Kronecker structured vectors. Sun et al. [193] consider sketches of the form (4.5) where each $\boldsymbol{\Omega}^{(n)}$ has sub-Gaussian entries. They provide Johnson–Lindenstrauss (JL) style guarantees for the case when the sketch is a Khatri–Rao product of two smaller matrices, i.e., for $N = 2$ in (4.5). Rakhshan and Rabusseau [175] consider sketches which have tensor train or CP tensor structure, and with core tensors and factor matrices that have Gaussian entries. They provide JL style guarantees for these sketches for arbitrary orders of the random tensors. Their sketch with CP tensor structure includes the sketch in (4.5) as a special case. Another line of work considers the Kronecker fast JL transform, which is a structured variant of the fast JL transform of Ailon and Chazelle [3]. It was first proposed by Battaglino et al. [15] with theoretical guarantees later provided by Jin et al. [105] and Malik and Becker [143]. A variant of this transform is also considered by Iwen et al. [102].

4.3 Fast randomized matrix ID using CountSketch

Algorithm 5 explains our proposal for CountSketch matrix ID. Proposition 19 provides guarantees for the method. A proof is provided in Section 4.7.1, which also contains a more detailed version of the bound in (4.8).

Proposition 19 (CountSketch matrix ID)**.** *Suppose I, R and $K < R$ are defined as in Algorithm 5.*

Algorithm 5: CountSketch matrix ID (proposal)

input : $\boldsymbol{A} \in \mathbb{R}^{I \times R}$, target rank K, sketch dimension L
output : $\boldsymbol{P} \in \mathbb{R}^{K \times R}$, $\boldsymbol{j} \in [R]^K$

1 Draw CountSketch matrix $\boldsymbol{S} \in \mathbb{R}^{L \times I}$
2 Compute sketch $\boldsymbol{Y} = \boldsymbol{S}\boldsymbol{A} \in \mathbb{R}^{L \times R}$ implicitly
3 Compute $[\boldsymbol{P}, \boldsymbol{j}] = \text{Matrix ID}(\boldsymbol{Y}, K)$ using Algorithm 4

Let $\beta > 1$ be a real number and L a positive integer such that

$$2\beta(K^2 + K) \leq L < I. \tag{4.7}$$

Suppose that the matrix ID on line 3 of Algorithm 5 utilizes SRRQR. Then, the output $\boldsymbol{P}$ of Algorithm 5 satisfies properties (i)–(iv) in Fact 18. Moreover, the outputs $[\boldsymbol{P}, \boldsymbol{j}]$ satisfy

$$\|\boldsymbol{A}_{:\boldsymbol{j}}\boldsymbol{P} - \boldsymbol{A}\| \lesssim 2\sigma_{K+1}(\boldsymbol{A})\sqrt{KIR} \tag{4.8}$$

with probability at least $1 - \frac{1}{\beta}$.

The condition in (4.7) is very similar to that for SRFT matrix ID by Woolfe et al. [213]. The only difference is that instead of a term of the form $(K^2 + K)$, their work only has a factor K^2. In practice, the condition in (4.7) is very conservative. We find that a small oversampling factor, e.g. $L = K + 10$, works well in practice, producing errors of the same size as the other randomized ID methods.

Remark 20. The semi-coherent matrices defined by Avron et al. [8] are adversarial to CountSketch, and therefore to our proposed method. For such matrices, using $L = K + 10$ may result in a large error for our method. Some care is therefore necessary when applying our method together this choice of L. In Section 4.6.1.2, we do extensive testing of our method on real-world matrices to demonstrate that using $L = K + 10$ works well in practice. We also provide an example of a matrix with semi-coherent structure on which our method fails when choosing L like this.

Remark 21. In cases when the target rank K is quite large (e.g. $K = R/2$), an issue we encountered is that $\boldsymbol{S}\boldsymbol{A}$ can be rank deficient due to rank deficiency of $\boldsymbol{S}$. This issue can be dealt with easily by

defining $\boldsymbol{S}$ slightly differently to ensure that each row contains at least one nonzero element. This is done by the following straightforward modification of the map h in the definition of CountSketch in Section 4.1.3: Let each $h(i) = v_i$, where $\boldsymbol{v} \in \mathbb{R}^I$ is a uniform random permutation of the elements of the vector $[1, \cdots, L, x_{L+1}, \cdots, x_I]$, where each $x_i \in [L]$ is iid uniformly random. With this modification, the guarantees of Proposition 19 still hold. In fact, the condition in (4.7) is slightly improved. We give a precise statement with proof in Section 4.7.2.

4.4 Extending the results to tensor ID

Let $\mathcal{X}$ and $\hat{\mathcal{X}}$ be defined as in (4.1) and (4.2), respectively. Our approach to the tensor ID problem is similar to that of Biagioni et al. [22]: We sketch the matrix $\boldsymbol{M}$ in (4.3) without forming it and compute a matrix ID of this sketch. The approximation $\hat{\mathcal{X}}$ is then constructed using the rank-1 components of $\mathcal{X}$ corresponding to the columns of $\boldsymbol{M}$ used in the ID of that matrix. The s-values $\hat{\lambda}_1, \ldots, \hat{\lambda}_K$ used in the representation of $\hat{\mathcal{X}}$ are then computed as $\hat{\lambda}_k = \lambda_{j_k} \sum_{r=1}^{R} p_{kr}$, for $k \in [K]$. The sketch we use is the efficient TensorSketch variant of CountSketch. Algorithm 6 outlines our proposed method for tensor ID. Proposition 22 provides guarantees for the method. A proof is provided in Section 4.7.3. To the best of our knowledge, there are no previous results like Proposition 22 for randomized tensor ID.

Algorithm 6: TensorSketch tensor ID (proposal)

input : CP tensor $\mathcal{X} \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_N}$, target rank K, sketch dimension L
output: rank-K approximation $\hat{\mathcal{X}}$

1 Draw TensorSketch operator $\boldsymbol{T} \in \mathbb{R}^{L \times \tilde{I}}$
2 Define $\boldsymbol{M}$ implicitly as in (4.3)
3 Compute sketch $\boldsymbol{Y} = \boldsymbol{T}\boldsymbol{M} \in \mathbb{R}^{L \times R}$ using (4.6)
4 Compute $[\boldsymbol{P}, \boldsymbol{j}] = \text{Matrix ID}(\boldsymbol{Y}, K)$ using Algorithm 4
5 Compute $\hat{\lambda}_k = \lambda_{j_k} \sum_{r=1}^{R} p_{kr}$ for $k \in [K]$
6 Define CP tensor $\hat{\mathcal{X}}$ as in (4.2)

Proposition 22 (TensorSketch tensor ID)**.** *Suppose $I_1, \ldots, I_N$, R and $K < R$ are defined as in Algorithm 6. Let $\beta > 1$ be a real number and L a positive integer such that $2(2 + 3^N)\beta K^2 \leq L < \tilde{I}$.*

Suppose that the matrix ID on line 4 of Algorithm 6 utilizes SRRQR. Then, the output of Algorithm 6 satisfies

$$\|\hat{\mathcal{X}} - \mathcal{X}\|_{\mathrm{F}} \lesssim 2\sigma_{K+1}(\boldsymbol{M})R\sqrt{KR\tilde{I}}$$

with probability at least $1 - \frac{1}{\beta}$.

As mentioned in Section 4.1.2.2, an issue with forming and then decomposing $\boldsymbol{M}^\top \boldsymbol{M}$ is that it can be ill-conditioned. Biagioni et al. [22] point out that the sketched matrix $\boldsymbol{\Omega M}$ typically is much better conditioned since $\kappa(\boldsymbol{\Omega M}) \leq \kappa(\boldsymbol{\Omega})\kappa(\boldsymbol{M})$ and Gaussian matrices are well-conditioned. As Proposition 23 demonstrates, the matrix $\boldsymbol{TM}$ is also well-conditioned with high probability, when the sketch dimension L is sufficiently large. A proof of Proposition 23 is provided in Section 4.7.4.

Proposition 23. *Let* $\beta > 1$ *be a real number, and let* L, R *and* $I_1, \ldots, I_N$ *be positive integers such that* $2(2 + 3^N)\beta R^2 \leq L$. *Suppose* $\boldsymbol{T} \in \mathbb{R}^{L \times \tilde{I}}$ *is a TensorSketch matrix, and* $\boldsymbol{M} \in \mathbb{R}^{\tilde{I} \times R}$ *is an arbitrary matrix. Then* $\kappa(\boldsymbol{TM}) \leq 7\kappa(\boldsymbol{M})$ *with probability at least* $1 - \frac{1}{\beta}$.

4.5 Complexity analysis

In this section, we compare the complexity of our proposed methods with the other algorithms. We assume all QR factorizations are done using column pivoted QR instead of SRRQR, and ignore the cost of generating random variables. We also assume that $L = K + C$ where C is a small positive integer (e.g. $L = K + 10$) since this choice works well in practice. Since $K < R$, and we assume $L = K + C$ for a small constant C, we also make the assumption $L < R$.

The costs of the different steps of Algorithm 5 are as follows:

- Computing the sketch $\boldsymbol{Y} = \boldsymbol{SA}$: $O(\mathrm{nnz}(\boldsymbol{A}))$.
- Computing Matrix ID of $\boldsymbol{Y} \in \mathbb{R}^{L \times R}$, where $L < R$: $O(L^2 R)$.

The total cost is therefore $O(\mathrm{nnz}(\boldsymbol{A}) + K^2 R)$. The cost of standard matrix ID can be found in Remark 3 of Cheng et al. [47], and the cost of SRFT matrix ID can be found in Remark 5.4 of

Woolfe et al. [213]. The cost of Gaussian matrix ID is straightforward to compute similarly to our computation above. Table 4.1 summarize these matrix ID complexities.

For the tensor ID algorithms, we assume the input is an N-way rank-R CP tensor of size $I \times \cdots \times I$, and that each factor matrix has the same number of nonzeros, which we denote by $\mathrm{nnz}(\boldsymbol{A})$. The costs of the different steps of Algorithm 6 are as follows:

- Computing the TensorSketched matrix $\boldsymbol{Y}$: $O(N(\mathrm{nnz}(\boldsymbol{A}) + RL \log L))$.
- Computing Matrix ID of $\boldsymbol{Y} \in \mathbb{R}^{L \times R}$, where $L < R$: $O(L^2 R)$.
- Computing $\hat{\lambda}_1, \ldots, \hat{\lambda}_K$: $O(RK)$.

The total cost is therefore $O(N(\mathrm{nnz}(\boldsymbol{A}) + RK \log K) + K^2 R)$. Although Biagioni et al. [22] do not specify these, the complexities for the Gram matrix approach and Gaussian tensor ID can be computed from the descriptions in their paper. Table 4.2 summarize the complexities for the different tensor ID algorithms. The constant C_{mult} is the cost of computing one Gram matrix $\boldsymbol{A}^{(n)\top}\boldsymbol{A}^{(n)}$ in (4.4), which we assume is the same for each n, e.g. $C_{\mathrm{mult}} = IR^2$ if the factor matrices were dense.

Table 4.1: Comparison of the complexity for matrix ID algorithms.

Algorithm for matrix ID	Complexity
Standard (Alg. 4)	KIR
Gaussian	$K\,\mathrm{nnz}(\boldsymbol{A}) + K^2 R$
SRFT	$IR \log(K) + K^2 R$
CountSketch (Proposal, Alg. 5)	$\mathrm{nnz}(\boldsymbol{A}) + K^2 R$

Table 4.2: Comparison of the complexity for tensor ID algorithms.

Algorithm for tensor ID	Complexity
Gram matrix	$NC_{\mathrm{mult}} + R^3$
Gaussian	$NK\,\mathrm{nnz}(\boldsymbol{A}) + K^2 R$
CountSketch (Proposal, Alg. 6)	$N(\mathrm{nnz}(\boldsymbol{A}) + RK \log K) + K^2 R$

4.6 Numerical experiments

The numerical experiments are done in Matlab R2018b and C. All results are averages over ten runs in an environment using four cores of an Intel Xeon E5-2680 v3 @2.50GHz CPU and 19 GB of RAM. All code used to generate our results can be found at `https://github.com/OsmanMalik/countsketch-matrix-tensor-id`, including implementations of our proposed methods. For all randomized methods, we use an oversampling parameter equal to 10 (i.e., $L = K+10$ in Algorithms 5 and 6).

4.6.1 Matrix ID experiments

We compare the four methods in Table 4.1. For standard matrix ID, we use the implementation in RSVDPACK[1] . For the remaining methods, we use our own Matlab implementations which utilize Matlab's column pivoted QR function. RSVDPACK only supports dense matrices. Moreover, since it is challenging to efficiently construct partial QR decompositions of sparse matrices, we did not attempt to write our own implementation of standard matrix ID for sparse matrices; see Section 11.1.8 of Golub and Van Loan [83] for a discussion about the challenges of sparse QR. We therefore have to convert each sparse input matrix to dense format before applying standard matrix ID from RSVDPACK. Similarly, it is challenging to implement an efficient algorithm for FFT for sparse matrices, or for the accelerated FFT by Woolfe et al. [213]. We therefore also have to convert the input matrix to dense format before applying standard FFT in our implementation of SRFT matrix ID. However, by only sketching a subset of columns of the input matrix at a time, we can avoid having to convert all columns of the matrix to dense format at the same time. In the experiments, we use the modification described in Remark 21 when implementing our proposed CountSketch matrix ID.

Computing the spectral norm of the matrices we consider is not feasible due to their size. Therefore, when computing the error for each matrix decomposition, we utilize the randomized

[1] Available at `https://github.com/sergeyvoronin/LowRankMatrixDecompositionCodes`.

scheme for estimating the spectral norm suggested in Section 3.4 of Woolfe et al. [213]. Letting E be the true error in spectral norm, our estimates $\tilde{E}$ satisfy the following properties: $\tilde{E} \leq E$, and $\mathbb{P}(\tilde{E} \geq E/100) = 1 - q$ where $0 < q \ll 1$. In other words, the estimate is smaller than the true spectral norm, but it is unlikely to be much smaller (with "much smaller" meaning more than two orders of magnitude smaller). This is good enough for our purposes, since we are primarily interested in comparing the performance of the different methods rather than establishing the exact errors. In the first experiment, $q <$ 2e−2, and in the second $q <$ 2e−5.

4.6.1.1 Experiment 1: Synthetic matrices

We generate sparse matrices $\boldsymbol{A} \in \mathbb{R}^{I \times R}$ with $R =$ 1e+4, and density $\text{nnz}(\boldsymbol{A})/(IR) \approx 0.5\%$. We use different values of $I \in$ [1e+4, 1e+6]. The matrices have a true rank of $2K$, where $K =$ 1e+3. Similarly to experiments by Martinsson et al. [150], we let $\sigma_i(\boldsymbol{A})$, $i \in [K]$, decay exponentially to 10^{-8}, and then remain constant at $\sigma_i(\boldsymbol{A}) \approx 10^{-8}$, $i \in [K+1 : 2K]$.

The results for the first experiment are presented in Figure 4.1. Standard matrix ID encountered memory issues when $I \geq$ 5e+4. For the matrix sizes the standard method could handle, it was more accurate but much slower than the randomized methods. The accuracy of all randomized methods is comparable. Our proposed CountSketch matrix ID is the fastest, achieving a speed-up of about 18× and 12× when $I =$ 1e+6 compared to Gaussian and SRFT ID, respectively.

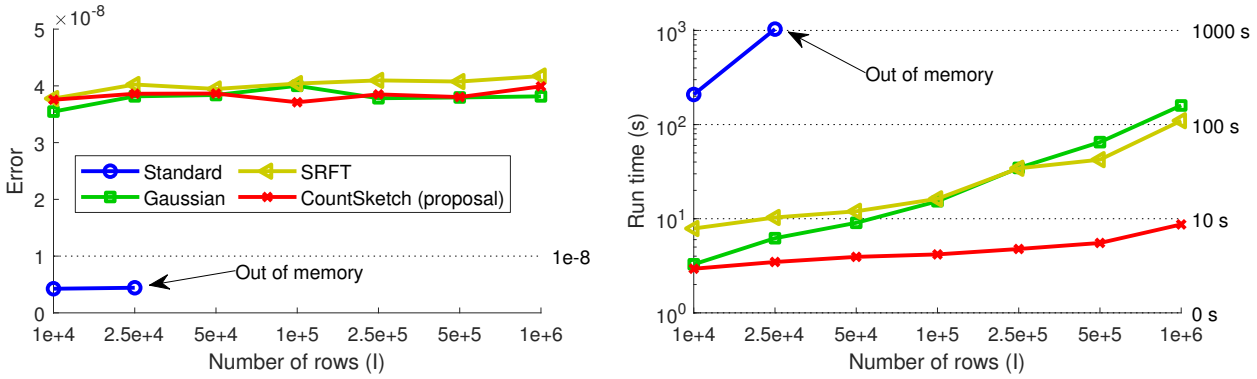

Figure 4.1: Errors (left) and run times (right) in the synthetic matrix ID experiment. The errors are computed using the randomized spectral norm by Woolfe et al. [213].

4.6.1.2 Experiment 2: Real-world matrices

We decompose a sparse matrix which comes from a computer vision problem and is part of the SuiteSparse Matrix Collection[2] . The matrix is of size 477,976 by 1,600, contains 7,647,040 nonzero elements, and has a rank of 1,442. We set the target rank to $K = 1{,}442$. Ideally, the methods should be able to produce decompositions with a very small error. We only attempt this with the three randomized methods, since the matrix is too large for standard matrix ID. Table 4.3 shows the result. All methods produce good approximations with a small error. Our proposed CountSketch matrix ID method is much faster than the other algorithms, achieving a speed-up of about 35× and 31× compared to Gaussian and SRFT matrix ID, respectively.

Table 4.3: Errors and run times in the real-world matrix ID experiment. The errors are computed using the randomized spectral norm by Woolfe et al. [213].

Algorithm for matrix ID	Error	Run time (s)
Gaussian	1.505e−15	20.38
SRFT	1.507e−15	18.40
CountSketch (proposal)	1.504e−15	0.59

To further support our claim that $L = K + 10$ works well in practice, we have done additional experiments. We consider 20 matrices from the SuiteSparse Matrix Collection[3] , and 3 different target ranks (10%, 50% and 90% of the number of columns). The matrices are of different sizes and come from different application areas. We compare the performance of Gaussian, SRFT and CountSketch (proposed method) matrix ID, all with $L = K + 10$, repeating each experiment 10 times and reporting averages. The results are in Table 4.4; on average, our method is **the most accurate** even compared to the Gaussian method which it outperforms in 34 of the 60 tests. Our method outperforms the SRFT method in 57 of the 60 tests. Out of the 26 cases when the Gaussian method outperforms our method, the difference is no more than 7% in 25 of those cases, and 31% in one case. Out of the 58 cases when the Gaussian method outperforms the SRFT method, the

[2] The matrix can be downloaded from `https://sparse.tamu.edu/Brogan/specular`.

[3] They are `landmark`, `Franz7`, `ch7-8-b2`, `ch7-9-b2`, `ch8-8-b2`, `mk12-b2`, `shar_te2-b1`, `rel7`, `relat7b`, `relat7`, `abtaha2`, `abtaha1`, `specular`, `photogrammetry2`, `GL7d12`, `ch7-6-b2`, `ch7-7-b2`, `cis-n4c6-b3`, `mk11-b2`, `n4c6-b3`.

difference is no more than 14% in 56 of those cases, and 36%–45% in two cases.

Table 4.4: Number of experiments out of 60 for which method A is more accurate than method B.

	Method B		
Method A	Our Proposal	SRFT	Gaussian
Our Proposal	-	57	34
SRFT	3	-	2
Gaussian	26	58	-

As mentioned in Remark 20, semi-coherent matrices are adversarial to CountSketch and our proposed method. The matrix `soc-sign-bitcoin-otc` in the SuiteSparse Matrix Collection, which is the adjacency matrix of a graph, is a concrete example of when choosing $L = K + 10$ results in a large error for our method. The semi-coherent structure of this matrix can be revealed by rearranging it so that rows and columns corresponding to nodes in the same strongly connected components of the graph are adjacent in the matrix. Some care is therefore necessary when applying our method together with the rule of thumb $L = K + 10$.

4.6.2 Tensor ID experiments

We compare the three methods in Table 4.2. We have implemented all methods ourselves in Matlab and C.

4.6.2.1 Experiment 1: Synthetic tensors

We generate sparse 5-way tensors $\boldsymbol{\mathcal{X}} \in \mathbb{R}^{I \times \cdots \times I}$ using (4.1), where each factor matrix column $\boldsymbol{a}_{:r}^{(n)}$ is a random sparse vector with a density of 1%, and we use different values of $I \in [1\text{e}{+3}, 1\text{e}{+5}]$. The number of rank-1 terms is $R = 10{,}000$, and we use a target rank of 1,000. The values of λ_r in (4.1) are defined as $\lambda_r \stackrel{\text{def}}{=} 10^{-\frac{r-1}{R}8}$ for $r \in [1000]$, and $\lambda_r \stackrel{\text{def}}{=} 10^{-8}$ for $r \in [1001 : R]$. The results for the experiment are presented in Figure 4.2. Gaussian and CountSketch tensor ID achieve similar accuracy. Although the Gram matrix approach has a better accuracy here, it can have issues reaching an error below the square root of machine precision due to poor conditioning; see the

example in Section 5.1.1 of Biagioni et al. [22]. Our proposed method is much faster than both other methods for the larger tensors, achieving a speed-up of 46× over the Gram matrix approach (for I = 2.5e+4) and 14× over Gaussian tensor ID (for I = 1e+5).

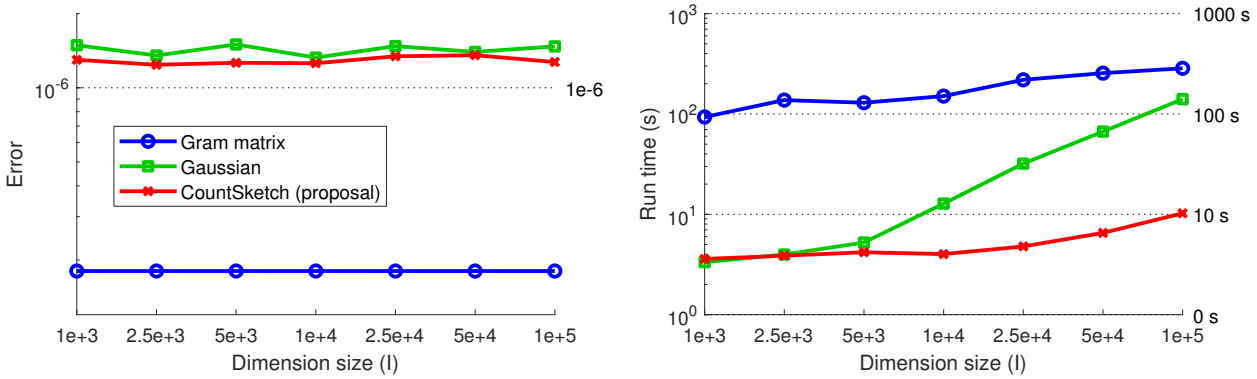

Figure 4.2: Errors (left) and run times (right) in the synthetic tensor ID experiment. The errors are in Frobenius norm.

4.6.2.2 Experiment 2: Real-world tensor

The purpose of this experiment is to show how tensor ID can be useful in a data analysis task. We implement Algorithm 2 by Reynolds et al. [180], which requires repeated rank reduction, and use it to find the maximum magnitude element in a CP tensor which comes from decomposing streamed data. The rank reduction step is done using tensor ID. The data we consider is a decomposed version of the Enron data set[4] of size 6,066 × 5,699 × 244,268 × 1,176. The Enron data set keeps track of email correspondence between employees at Enron, and the four modes represent sender, receiver, keyword and date. The decomposition has rank 100, and was constructed using the streamed version of SPLATT[5] [189], with the data streamed along the fourth mode (time). As suggested in the documentation of SPLATT-stream, we apply an additional Frobenius norm regularizer with regularization coefficient 1e−2 to the mode-4 factor matrix. We threshold the factor matrices outputted by SPLATT-stream by first normalizing them so that each column have unit 2-norm (the normalization constant is absorbed into the s-values) and then setting all elements with magnitude

[4] The data set is available at `http://frostt.io/tensors/enron`.
[5] The streamed version of SPLATT is available at `https://github.com/ShadenSmith/splatt-stream`.

less than 1e−6 to zero. The relative error introduced by this thresholding is less than 2e−5.

Unlike the previous experiments, the matrices being sketched in this experiment have many rows containing only zeros. We therefore could speed up Gaussian tensor ID by only generating those columns of the Gaussian sketch matrices which are actually multiplied by nonzero elements. We used this improved version of Gaussian tensor ID in the experiment for a more fair comparison. The same modification does not yield a speed-up of Gaussian matrix or tensor ID in the previous experiments since there most rows of the matrices being sketched contain nonzero elements.

Finding the maximum magnitude element using a brute force approach would require computing every nonzero element in the tensor, which would be costly. Using the algorithm by Reynolds et al. [180] together with our CountSketch tensor ID, we find the maximum in 11 seconds. The sketching portion of the algorithm takes 2.6× more time if Gaussian tensor ID is used instead. We do not compare with the Gram matrix approach since it takes very long to run. With the results in the previous subsection in mind, we believe the speed-up would be more substantial for higher rank tensors. For all ten trials, and both when using CountSketch and Gaussian tensor ID for rank reduction, the same position for the maximum magnitude element is identified each time.

4.7 Proofs

4.7.1 Proof of Proposition 19

Our proof of Proposition 19 is an adaption of the proof for SRFT matrix ID provided by Woolfe et al. [213]. We show that their arguments hold when a CountSketch matrix is used for sketching instead of an SRFT matrix. Although much of our proof is identical to that provided by Woolfe et al. [213], we choose to include it in detail. The reason for doing this is that the proofs of Propositions 22 and 33 rely on adapting the proof in the present section. Having a detailed proof here therefore makes those subsequent proofs easier to follow.

The following facts will be useful in the proof.

Fact 24 (Lemma 3.7 in [150])**.** Let I and R be positive integers with $I \geq R$. Suppose $\boldsymbol{A} \in \mathbb{R}^{I \times R}$ is

a matrix such that $\boldsymbol{A}^\top \boldsymbol{A}$ is invertible. Then

$$\|(\boldsymbol{A}^\top \boldsymbol{A})^{-1}\boldsymbol{A}^\top\| = \frac{1}{\sigma_R(\boldsymbol{A})}.$$

Fact 25 (Lemma 3.7 in [213])**.** Let K, L, I and R be positive integers such that $K \leq R$. Suppose $\boldsymbol{A} \in \mathbb{R}^{I\times R}$, $\boldsymbol{B} \in \mathbb{R}^{I\times K}$ is a matrix whose columns constitute a subset of the columns of $\boldsymbol{A}$, $\boldsymbol{P} \in \mathbb{R}^{K\times R}$, $\boldsymbol{X} \in \mathbb{R}^{I\times L}$, and $\boldsymbol{S} \in \mathbb{R}^{L\times I}$. Then

$$\|\boldsymbol{BP} - \boldsymbol{A}\| \leq \|\boldsymbol{XSA} - \boldsymbol{A}\|(\|\boldsymbol{P}\| + 1) + \|\boldsymbol{X}\|\|\boldsymbol{SBP} - \boldsymbol{SA}\|.$$

Fact 26 (Lemma 3.9 in [150])**.** Let L, I and R be positive integers. Suppose $\boldsymbol{A} \in \mathbb{R}^{I\times R}$, and $\boldsymbol{S} \in \mathbb{R}^{L\times I}$. Then $\sigma_j(\boldsymbol{SA}) \leq \|\boldsymbol{S}\|\sigma_j(\boldsymbol{A})$ for all $j \in [\min(L, I, R)]$.

Fact 27 is a special case of a more general statement in Atkinson and Han [6].

Fact 27 (Theorem 2.3.1 of Atkinson and Han [6])**.** Let $\boldsymbol{L} \in \mathbb{R}^{K\times K}$ be a matrix and assume $\|\boldsymbol{L}\| < 1$. Then $(\boldsymbol{I}^{(K)} - \boldsymbol{L})$ is invertible and

$$\|(\boldsymbol{I}^{(K)} - \boldsymbol{L})^{-1}\| \leq \frac{1}{1 - \|\boldsymbol{L}\|}.$$

Lemma 28 is an adaption of Lemma 4.2 by Woolfe et al. [213].

Lemma 28. *Let K, L and I be positive integers such that $K \leq I$. Suppose $\boldsymbol{S} = \boldsymbol{\Phi D} \in \mathbb{R}^{L\times I}$ is a CountSketch matrix, and $\boldsymbol{U} \in \mathbb{R}^{I\times K}$ is a matrix with orthonormal columns. Define $\boldsymbol{C} \in \mathbb{R}^{K\times K}$ as*

$$\boldsymbol{C} \stackrel{\text{def}}{=} (\boldsymbol{SU})^\top(\boldsymbol{SU}),$$

and define $\boldsymbol{E} \in \mathbb{R}^{K\times K}$ elementwise as

$$e_{kk'} \stackrel{\text{def}}{=} \sum_{\substack{i,i'\in[I]\\ i\neq i'}} d_{ii}d_{i'i'}u_{ik}u_{i'k'}\Big(\sum_{l\in[L]}\phi_{li}\phi_{li'}\Big). \tag{4.9}$$

Then $\boldsymbol{C} = \boldsymbol{I}^{(K)} + \boldsymbol{E}$.

Proof. For $k, k' \in [K]$,

$$c_{kk'} = \sum_{l\in[L]}(\boldsymbol{SU})_{lk}(\boldsymbol{SU})_{lk'}. \tag{4.10}$$

Since

$$(\boldsymbol{SU})_{lk} = \sum_{i\in[I]} \phi_{li} d_{ii} u_{ik},$$

we can rewrite (4.10) as

$$\begin{aligned} c_{kk'} &= \sum_{l\in[L]} \Big(\sum_{i\in[I]} \phi_{li} d_{ii} u_{ik} \Big) \Big(\sum_{i'\in[I]} \phi_{li'} d_{i'i'} u_{i'k'} \Big) \\ &= \sum_{i\in[I]} \sum_{l\in[L]} \phi_{li}^2 d_{ii}^2 u_{ik} u_{ik'} + \sum_{\substack{i,i'\in[I] \\ i\neq i'}} d_{ii} d_{i'i'} u_{ik} u_{i'k'} \Big(\sum_{l\in[L]} \phi_{li} \phi_{li'} \Big). \end{aligned}$$

The second term on the last line in the equation above is just $e_{kk'}$. Since

$$\phi_{li}^2 = \begin{cases} 1 & \text{if } h(i) = l, \\ 0 & \text{otherwise,} \end{cases}$$

and $d_{ii}^2 = 1$, the first term is just

$$\sum_{i\in[I]} \sum_{l\in[L]} \phi_{li}^2 d_{ii}^2 u_{ik} u_{ik'} = \sum_{i\in[I]} u_{ik} u_{ik'} = \langle \boldsymbol{u}_{:k}, \boldsymbol{u}_{:k'} \rangle = \begin{cases} 1 & \text{if } k = k', \\ 0 & \text{otherwise.} \end{cases}$$

It follows that $\boldsymbol{C} = \boldsymbol{I}^{(K)} + \boldsymbol{E}$. □

Lemma 29 is an adaption of Lemma 4.3 by Woolfe et al. [213].

Lemma 29. *Let α and β be real numbers such that $\alpha, \beta > 1$, and let K, L and I be positive integers such that*

$$\Big(\frac{\alpha}{\alpha - 1}\Big)^2 \beta(K^2 + K) \leq L < I. \tag{4.11}$$

Suppose $\boldsymbol{S} = \boldsymbol{\Phi D} \in \mathbb{R}^{L\times I}$ is a CountSketch matrix, $\boldsymbol{U} \in \mathbb{R}^{I\times K}$ is a matrix with orthonormal columns, and $\boldsymbol{E} \in \mathbb{R}^{K\times K}$ is the matrix defined in (4.9)*. Then*

$$\|\boldsymbol{E}\| \leq 1 - \frac{1}{\alpha} \tag{4.12}$$

with probability at least $1 - \frac{1}{\beta}$.

Proof. Using the definition in (4.9), we have

$$\mathbb{E}[e_{kk'}^2] = \mathbb{E}\Big[\sum_{\substack{i,i' \in [I] \\ i \neq i'}} \sum_{\substack{j,j' \in [I] \\ j \neq j'}} d_{ii} d_{i'i'} d_{jj} d_{j'j'} u_{ik} u_{i'k'} u_{jk} u_{j'k'} \Big(\sum_{l \in [L]} \phi_{li} \phi_{li'} \Big) \Big(\sum_{l \in [L]} \phi_{lj} \phi_{lj'} \Big) \Big]. \tag{4.13}$$

Note that for each term in the sum above, $i \neq i'$ and $j \neq j'$. This means that unless ($i = j$ and $i' = j'$) or ($i = j'$ and $i' = j$), we have

$$\mathbb{E}\Big[d_{ii} d_{i'i'} d_{jj} d_{j'j'} u_{ik} u_{i'k'} u_{jk} u_{j'k'} \Big(\sum_{l \in [L]} \phi_{li} \phi_{li'} \Big) \Big(\sum_{l \in [L]} \phi_{lj} \phi_{lj'} \Big) \Big] = 0,$$

since each d_{ii} is independent from all other random variables, and since $\mathbb{E}[d_{ii}] = 0$ for all $i \in [I]$. We can therefore rewrite (4.13) as

$$\begin{aligned} \mathbb{E}[e_{kk'}^2] = &\sum_{\substack{i,i' \in [I] \\ i \neq i'}} \mathbb{E}\Big[d_{ii}^2 d_{i'i'}^2 u_{ik} u_{i'k'} u_{ik} u_{i'k'} \Big(\sum_{l \in [L]} \phi_{li} \phi_{li'} \Big)^2 \Big] \\ &+ \sum_{\substack{i,i' \in [I] \\ i \neq i'}} \mathbb{E}\Big[d_{ii}^2 d_{i'i'}^2 u_{ik} u_{i'k'} u_{i'k} u_{ik'} \Big(\sum_{l \in [L]} \phi_{li} \phi_{li'} \Big)^2 \Big]. \end{aligned} \tag{4.14}$$

The matrix $\mathbf{\Phi}$ has exactly one nonzero entry which is equal to 1 in each column. Consequently,

$$\Big(\sum_{l \in [L]} \phi_{li} \phi_{li'} \Big)^2 = \begin{cases} 1 & \text{if } h(i) = h(i'), \\ 0 & \text{otherwise.} \end{cases}$$

The event $h(i) = h(i')$ happens with probability $\frac{1}{L}$ when $i \neq i'$. If follows that

$$\mathbb{E}\Big[\Big(\sum_{l \in [L]} \phi_{li} \phi_{li'} \Big)^2 \Big] = 1 \times \frac{1}{L} + 0 \times \Big(1 - \frac{1}{L} \Big) = \frac{1}{L}. \tag{4.15}$$

Using this fact, and the fact that each $d_{ii}^2 = 1$, (4.14) simplifies to

$$\mathbb{E}[e_{kk'}^2] = \frac{1}{L} \sum_{\substack{i,i' \in [I] \\ i \neq i'}} u_{ik}^2 u_{i'k'}^2 + \frac{1}{L} \sum_{\substack{i,i' \in [I] \\ i \neq i'}} u_{ik} u_{i'k'} u_{i'k} u_{ik'}. \tag{4.16}$$

Note that

$$\sum_{\substack{i,i' \in [I] \\ i \neq i'}} u_{ik}^2 u_{i'k'}^2 = \sum_{i \in [I]} u_{ik}^2 \sum_{\substack{i' \in [I] \\ i' \neq i}} u_{i'k'}^2 \leq \|\boldsymbol{u}_{:k}\|^2 \|\boldsymbol{u}_{:k'}\|^2 = 1. \tag{4.17}$$

Moreover,

$$\begin{aligned}\sum_{\substack{i,i'\in[I]\\ i\neq i'}} u_{ik}u_{i'k'}u_{i'k}u_{ik'} &= \sum_{i\in[I]} u_{ik}u_{ik'} \sum_{\substack{i'\in[I]\\ i'\neq i}} u_{i'k'}u_{i'k} \\ &= \sum_{i\in[I]} u_{ik}u_{ik'}(\langle \boldsymbol{u}_{:k}, \boldsymbol{u}_{:k'}\rangle - u_{ik}u_{ik'}) \\ &= \langle \boldsymbol{u}_{:k}, \boldsymbol{u}_{:k'}\rangle^2 - \sum_{i\in[I]} u_{ik}^2 u_{ik'}^2 \leq \langle \boldsymbol{u}_{:k}, \boldsymbol{u}_{:k'}\rangle^2 = \begin{cases} 1 & \text{if } k = k', \\ 0 & \text{otherwise.} \end{cases}\end{aligned} \tag{4.18}$$

Combining (4.16), (4.17) and (4.18) yields

$$\mathbb{E}[e_{kk'}^2] \leq \begin{cases} \frac{2}{L} & \text{if } k = k', \\ \frac{1}{L} & \text{otherwise.} \end{cases}$$

Since

$$\|\boldsymbol{E}\|^2 \leq \|\boldsymbol{E}\|_{\mathrm{F}}^2 = \sum_{k,k'\in[K]} e_{kk'}^2,$$

we have

$$\mathbb{E}[\|\boldsymbol{E}\|^2] \leq \sum_{k\in[K]} \mathbb{E}[e_{kk}^2] + \sum_{\substack{k,k'\in[K]\\ k\neq k'}} \mathbb{E}[e_{kk'}^2] \leq \frac{2K}{L} + \frac{K^2-K}{L} = \frac{K^2+K}{L}.$$

Using Markov's inequality and the condition in (4.11), we have

$$\mathbb{P}\Big(\|\boldsymbol{E}\| \geq 1 - \frac{1}{\alpha}\Big) \leq \mathbb{P}\Big(\|\boldsymbol{E}\|^2 \geq \frac{\beta(K^2+K)}{L}\Big) \leq \frac{L}{\beta(K^2+K)}\mathbb{E}[\|\boldsymbol{E}\|^2] \leq \frac{1}{\beta}.$$

Consequently,

$$\mathbb{P}\Big(\|\boldsymbol{E}\| \leq 1 - \frac{1}{\alpha}\Big) \geq 1 - \frac{1}{\beta}.$$

□

Lemma 30 is an adaption of Lemma 4.4 by Woolfe et al. [213].

Lemma 30. *Let α, β, K, L and I satisfy the same properties as in Lemma 29. Furthermore, suppose $\boldsymbol{S}$, $\boldsymbol{U}$, and $\boldsymbol{E}$ are defined as in Lemma 29, and let $\boldsymbol{C} \stackrel{\text{def}}{=} (\boldsymbol{SU})^\top(\boldsymbol{SU})$. If* (4.12) *is true, then the following hold:*

$$\sigma_1(\boldsymbol{SU}) = \sqrt{\|\boldsymbol{C}\|} \leq \sqrt{2 - \frac{1}{\alpha}},$$

$\boldsymbol{C}$ is invertible, and

$$\sigma_K(\boldsymbol{SU}) = \frac{1}{\sqrt{\|\boldsymbol{C}^{-1}\|}} \geq \frac{1}{\sqrt{\alpha}}.$$

Proof. Using Lemma 28 and (4.12), we then have

$$\sigma_1(\boldsymbol{SU}) = \sqrt{\|\boldsymbol{C}\|} = \sqrt{\|\boldsymbol{I}^{(K)} + \boldsymbol{E}\|} \leq \sqrt{\|\boldsymbol{I}\| + \|\boldsymbol{E}\|} \leq \sqrt{1 + 1 - \frac{1}{\alpha}} = \sqrt{2 - \frac{1}{\alpha}}.$$

Since $\boldsymbol{C} = \boldsymbol{I}^{(K)} + \boldsymbol{E}$ and $\|\boldsymbol{E}\| < 1$, it follows from Fact 27 that $\boldsymbol{C}$ is invertible and

$$\|\boldsymbol{C}^{-1}\| = \|(\boldsymbol{I} + \boldsymbol{E})^{-1}\| \leq \frac{1}{1 - \|\boldsymbol{E}\|} \leq \alpha,$$

where the last inequality follows from (4.12). Consequently,

$$\sigma_K(\boldsymbol{SU}) = \frac{1}{\sqrt{\|\boldsymbol{C}^{-1}\|}} \geq \frac{1}{\sqrt{\alpha}}.$$

□

Lemma 31 is an adaption of Lemma 4.5 by Woolfe et al. [213].

Lemma 31. *Let L and I be positive integers with $L < I$. Suppose $\boldsymbol{S} \in \mathbb{R}^{L \times I}$ is a CountSketch matrix. Then $\|\boldsymbol{S}\| \leq \sqrt{I}$.*

Proof. The matrix $\boldsymbol{S}$ contains I nonzero elements, all of magnitude 1. It follows that $\|\boldsymbol{S}\|_{\mathrm{F}}^2 = I$, and hence $\|\boldsymbol{S}\| \leq \|\boldsymbol{S}\|_{\mathrm{F}} \leq \sqrt{I}$. □

Lemma 32 is an adaption of Lemma 4.6 by Woolfe et al. [213].

Lemma 32. *Let α, β, K, L and I satisfy the same properties as in Lemma 29. Suppose $\boldsymbol{S} \in \mathbb{R}^{L \times I}$ is a CountSketch matrix, and $\boldsymbol{A} \in \mathbb{R}^{I \times R}$ is an arbitrary matrix. Then, with probability at least $1 - \frac{1}{\beta}$, there exists a matrix $\boldsymbol{X} \in \mathbb{R}^{I \times L}$ such that*

$$\|\boldsymbol{XSA} - \boldsymbol{A}\| \leq \sigma_{K+1}(\boldsymbol{A})\sqrt{\alpha I + 1} \tag{4.19}$$

and

$$\|\boldsymbol{X}\| \leq \sqrt{\alpha}. \tag{4.20}$$

Proof. Let $\boldsymbol{A} = \boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}^\top$ be the SVD of $\boldsymbol{A}$, where $\boldsymbol{U} \in \mathbb{R}^{I\times I}$ and $\boldsymbol{V} \in \mathbb{R}^{R\times R}$ are unitary, and $\boldsymbol{\Sigma} \in \mathbb{R}^{I\times R}$ is diagonal with non-negative entries. Split $\boldsymbol{U}$ into two matrices $\boldsymbol{U}^{(1)} \in \mathbb{R}^{I\times K}$ and $\boldsymbol{U}^{(2)} \in \mathbb{R}^{I\times(I-K)}$ so that $\boldsymbol{U} = \begin{bmatrix}\boldsymbol{U}^{(1)} & \boldsymbol{U}^{(2)}\end{bmatrix}$. Let $\boldsymbol{Z}^{(1)} = \boldsymbol{S}\boldsymbol{U}^{(1)} \in \mathbb{R}^{L\times K}$ and $\boldsymbol{Z}^{(2)} = \boldsymbol{S}\boldsymbol{U}^{(2)} \in \mathbb{R}^{L\times(I-K)}$. Then

$$\boldsymbol{S}\boldsymbol{U} = \begin{bmatrix}\boldsymbol{Z}^{(1)} & \boldsymbol{Z}^{(2)}\end{bmatrix} \in \mathbb{R}^{L\times I}. \tag{4.21}$$

Define $\boldsymbol{C} = \boldsymbol{Z}^{(1)\top}\boldsymbol{Z}^{(1)} \in \mathbb{R}^{K\times K}$ and let $\boldsymbol{E}$ be the corresponding matrix defined in (4.9), but in terms of $\boldsymbol{U}^{(1)}$ instead of $\boldsymbol{U}$. Then $\boldsymbol{C} = \boldsymbol{I}^{(K)} + \boldsymbol{E}$ according to Lemma 28. For the remainder of the proof, we will assume that $\|\boldsymbol{E}\| \le 1 - \frac{1}{\alpha}$, which happens with probability at least $1 - \frac{1}{\beta}$ according to Lemma 29. Then $\boldsymbol{C}$ is invertible according to Lemma 30. Define $\boldsymbol{G}^{(-1)} \stackrel{\text{def}}{=} \boldsymbol{C}^{-1}\boldsymbol{Z}^{(1)\top} = (\boldsymbol{Z}^{(1)\top}\boldsymbol{Z}^{(1)})^{-1}\boldsymbol{Z}^{(1)\top} \in \mathbb{R}^{K\times L}$ and

$$\boldsymbol{X} \stackrel{\text{def}}{=} \boldsymbol{U}\begin{bmatrix}\boldsymbol{G}^{(-1)} \\ \boldsymbol{0}\end{bmatrix} \in \mathbb{R}^{I\times L}. \tag{4.22}$$

According to Fact 24 and Lemma 30, it follows that

$$\|\boldsymbol{G}^{(-1)}\| = \frac{1}{\sigma_K(\boldsymbol{Z}^{(1)})} = \frac{1}{\sigma_K(\boldsymbol{S}\boldsymbol{U}^{(1)})} \le \sqrt{\alpha}. \tag{4.23}$$

Combining (4.22) and (4.23), we have

$$\|\boldsymbol{X}\| = \|\boldsymbol{G}^{(-1)}\| \le \sqrt{\alpha}.$$

So (4.20) is satisfied. Next, let $\boldsymbol{\Theta} \in \mathbb{R}^{K\times K}$ and $\boldsymbol{\Psi} \in \mathbb{R}^{(I-K)\times(I-K)}$ be the matrices in the upper left and lower right corners of $\boldsymbol{\Sigma}$, respectively, so that

$$\boldsymbol{\Sigma} = \begin{bmatrix}\boldsymbol{\Theta} & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{\Psi}\end{bmatrix}. \tag{4.24}$$

It is easy to verify that

$$\boldsymbol{X}\boldsymbol{S}\boldsymbol{A} - \boldsymbol{A} = \boldsymbol{U}\Big(\begin{bmatrix}\boldsymbol{G}^{(-1)} \\ \boldsymbol{0}\end{bmatrix}\begin{bmatrix}\boldsymbol{Z}^{(1)} & \boldsymbol{Z}^{(2)}\end{bmatrix} - \boldsymbol{I}^{(I)}\Big)\boldsymbol{\Sigma}\boldsymbol{V}^\top. \tag{4.25}$$

Using (4.24), we can further rewrite

$$\left(\begin{bmatrix} \boldsymbol{G}^{(-1)} \\ \boldsymbol{0} \end{bmatrix} \begin{bmatrix} \boldsymbol{Z}^{(1)} & \boldsymbol{Z}^{(2)} \end{bmatrix} - \boldsymbol{I}^{(I)} \right) \boldsymbol{\Sigma} = \begin{bmatrix} \boldsymbol{0} & \boldsymbol{G}^{(-1)} \boldsymbol{Z}^{(2)} \boldsymbol{\Psi} \\ \boldsymbol{0} & -\boldsymbol{\Psi} \end{bmatrix}. \tag{4.26}$$

Note that

$$\left\| \begin{bmatrix} \boldsymbol{0} & \boldsymbol{G}^{(-1)} \boldsymbol{Z}^{(2)} \boldsymbol{\Psi} \\ \boldsymbol{0} & -\boldsymbol{\Psi} \end{bmatrix} \right\|^2 \leq \|\boldsymbol{G}^{(-1)} \boldsymbol{Z}^{(2)} \boldsymbol{\Psi}\|^2 + \|\boldsymbol{\Psi}\|^2 \leq \|\boldsymbol{G}^{(-1)}\|^2 \|\boldsymbol{Z}^{(2)}\|^2 \|\boldsymbol{\Psi}\|^2 + \|\boldsymbol{\Psi}\|^2. \tag{4.27}$$

From (4.24), we know that

$$\|\boldsymbol{\Psi}\| = \sigma_{K+1}(\boldsymbol{A}). \tag{4.28}$$

Moreover, using (4.21), the fact that $\boldsymbol{U}$ is unitary, and Lemma 31, we have

$$\|\boldsymbol{Z}^{(2)}\| \leq \| \begin{bmatrix} \boldsymbol{Z}^{(1)} & \boldsymbol{Z}^{(2)} \end{bmatrix} \| = \|\boldsymbol{S}\boldsymbol{U}\| = \|\boldsymbol{S}\| \leq \sqrt{I}. \tag{4.29}$$

Combining (4.25), (4.26), (4.27), (4.28), (4.29) and (4.23) we have

$$\|\boldsymbol{X}\boldsymbol{S}\boldsymbol{A} - \boldsymbol{A}\| \leq \sigma_{K+1}(\boldsymbol{A}) \sqrt{\alpha I + 1},$$

which proves (4.19). □

We can now prove Proposition 19 in the main manuscript. The proof is an adaption of the discussion in Section 5.1 of Woolfe et al. [213].

Proof of Proposition 19. According to Fact 18, the outputs $\boldsymbol{P}$ and $\boldsymbol{j}$ computed on line 3 of Algorithm 5 satisfy the following: $\boldsymbol{P}$ satisfies properties (i)–(iv) in Fact 18, including

$$\|\boldsymbol{P}\| \leq \sqrt{4K(R-K)+1}, \tag{4.30}$$

and

$$\|\boldsymbol{Y}_{:\boldsymbol{j}} \boldsymbol{P} - \boldsymbol{Y}\| \leq \sigma_{K+1}(\boldsymbol{Y}) \sqrt{4K(R-K)+1}, \tag{4.31}$$

since $K \leq \min(L, R)$. Applying Fact 25, we have

$$\|\boldsymbol{A}_{:\boldsymbol{j}} \boldsymbol{P} - \boldsymbol{A}\| \leq \|\boldsymbol{X}\boldsymbol{S}\boldsymbol{A} - \boldsymbol{A}\|(\|\boldsymbol{P}\| + 1) + \|\boldsymbol{X}\| \|\boldsymbol{S}\boldsymbol{A}_{:\boldsymbol{j}} \boldsymbol{P} - \boldsymbol{S}\boldsymbol{A}\|, \tag{4.32}$$

where $\boldsymbol{X} \in \mathbb{C}^{I \times L}$ is an arbitrary matrix. From Lemma 32, with probability at least $1 - \frac{1}{\beta}$, we can choose $\boldsymbol{X}$ such that the bounds in (4.19) and (4.20) hold. Moreover, since $\boldsymbol{Y} = \boldsymbol{SA}$, it follows that $\boldsymbol{Y}_{:\boldsymbol{j}} = \boldsymbol{SA}_{:\boldsymbol{j}}$, and consequently,

$$\|\boldsymbol{SA}_{:\boldsymbol{j}}\boldsymbol{P} - \boldsymbol{SA}\| = \|\boldsymbol{Y}_{:\boldsymbol{j}}\boldsymbol{P} - \boldsymbol{Y}\|. \tag{4.33}$$

Combining (4.19), (4.20), (4.30), (4.31), (4.32), (4.33), and Fact 26 gives that

$$\|\boldsymbol{A}_{:\boldsymbol{j}}\boldsymbol{P} - \boldsymbol{A}\| \leq \sigma_{K+1}(\boldsymbol{A})((\sqrt{4K(R-K)+1}+1)\sqrt{\alpha I+1} + \sqrt{4K(R-K)+1}\sqrt{\alpha I})$$

with probability at least $1 - \frac{1}{\beta}$. Setting $\alpha = 4$ then yields the same bounds as in the statement in Proposition 19. □

4.7.2 Formal statement and proof of claim in Remark 21

We express the statement in Remark 21 in slightly different terms here. Let $f : [I] \to [L]$ be a hybrid deterministic/random function defined as

$$f(i) \stackrel{\text{def}}{=} \begin{cases} i & \text{if } i \in [L], \\ x_i & \text{if } i \in [L+1 : I], \end{cases}$$

where all x_i are iid random variables that are uniformly distributed in $[L]$. Furthermore, let $\pi : [I] \to [I]$ be a uniform random permutation function. We then define $\tilde{h} : [I] \to [L]$ as $\tilde{h}(i) \stackrel{\text{def}}{=} f(\pi(i))$. Using $\tilde{h}$ instead of h in the definition of CountSketch ensures that $\boldsymbol{S}$ is of full rank. The guarantees of Proposition 19 still hold for this modified CountSketch, and in fact the bound in (4.7) is slightly improved.

Proposition 33. *If $\tilde{h}$ defined in this way is used instead of h when defining $\boldsymbol{S}$ on line 1 in Algorithm 5, then Proposition 19 still holds, but with the condition in* (4.7) *improved to*

$$2\Big(1 - \frac{L(L-1)}{I(I-1)}\Big)\beta(K^2+K) \leq L < I.$$

We have not seen anyone else consider this kind of modified CountSketch.

Proof of Proposition 33. When using the modified CountSketch matrix proposed in Remark 21, the only thing that will change in the proof in Section 4.7.1 is Lemma 29. Notice that going from h to $\tilde{h}$ only impacts $\boldsymbol{\Phi}$ and not $\boldsymbol{D}$, and since $\boldsymbol{\Phi}$ and $\boldsymbol{D}$ remain independent, the argument that takes us from (4.13) to (4.14) remains valid for $\tilde{h}$. Indeed, the key conditions that each d_{ii} is independent from all other random variables and $\mathbb{E}[d_{ii}] = 0$ remain true when we use $\tilde{h}$ instead of h. However, the expectation in (4.15) will change, which impacts (4.16), due to the fact that the probability of the event $\tilde{h}(i) = \tilde{h}(i')$ when $i \neq i'$ is not $\frac{1}{L}$. Note that

$$\mathbb{P}(\tilde{h}(i) = \tilde{h}(i')) = \sum_{l\in[L]} \mathbb{P}(\tilde{h}(i) = l, \tilde{h}(i') = l) = \sum_{l\in[L]} \mathbb{P}(f(\pi(i)) = l, f(\pi(i')) = l). \tag{4.34}$$

We can rewrite

$$\begin{aligned}\mathbb{P}(f(\pi(i)) = l, f(\pi(i')) = l) = \mathbb{P}(f(\pi(i)) = l, f(\pi(i')) = l, \pi(i) \in [L], \pi(i') \in [L]) \\ + \mathbb{P}(f(\pi(i)) = l, f(\pi(i')) = l, \pi(i) \notin [L], \pi(i') \in [L]) \\ + \mathbb{P}(f(\pi(i)) = l, f(\pi(i')) = l, \pi(i) \in [L], \pi(i') \notin [L]) \\ + \mathbb{P}(f(\pi(i)) = l, f(\pi(i')) = l, \pi(i) \notin [L], \pi(i') \notin [L]).\end{aligned} \tag{4.35}$$

Notice that the first term on the right hand side of (4.35) is zero, since f then will map $\pi(i)$ and $\pi(i')$ to distinct elements. The second and third term in (4.35) are equal. Considering the second term, we have

$$\begin{aligned}&\mathbb{P}(f(\pi(i)) = l, f(\pi(i')) = l, \pi(i) \notin [L], \pi(i') \in [L]) \\ &= \sum_{j\in[L]} \mathbb{P}(f(\pi(i)) = l, f(\pi(i')) = l, \pi(i) \notin [L], \pi(i') \in [L], \pi(i') = j) \\ &= \mathbb{P}(f(\pi(i)) = l, \pi(i) \notin [L], \pi(i') = l),\end{aligned} \tag{4.36}$$

since if $\pi(i') \in [L]$, then $f(\pi(i')) = l$ if and only if $\pi(i') = l$. Furthermore,

$$\begin{aligned}&\mathbb{P}(f(\pi(i)) = l, \pi(i) \notin [L], \pi(i') = l) \\ &= \mathbb{P}(f(\pi(i)) = l \mid \pi(i) \notin [L], \pi(i') = l)\, \mathbb{P}(\pi(i) \notin [L], \pi(i') = l) \\ &= \mathbb{P}(x_I = l)\, \mathbb{P}(\pi(i) \notin [L], \pi(i') = l) \\ &= \frac{1}{L}\frac{I-L}{I(I-1)},\end{aligned} \tag{4.37}$$

where the second equality is true since each x_i, $i \in [L+1 : I]$ is iid. For the fourth term in the right hand side of (4.35), we have

$$\begin{aligned}
&\mathbb{P}(f(\pi(i)) = l, f(\pi(i')) = l, \pi(i) \notin [L], \pi(i') \notin [L]) \\
&= \mathbb{P}(x_{\pi(i)} = l, x_{\pi(i')} = l, \pi(i) \notin [L], \pi(i') \notin [L]) \\
&= \mathbb{P}^2(x_I = l)\mathbb{P}(\pi(i) \notin [L], \pi(i') \notin [L]) \\
&= \frac{1}{L^2}\frac{(I-L)(I-L-1)}{I(I-1)},
\end{aligned} \tag{4.38}$$

where the second equality again holds since each x_i, $i \in [L+1 : I]$ is iid and $\pi(i) \neq \pi(i')$. Combining (4.34), (4.35), (4.36), (4.37), (4.38), and using the fact that the second and third term in (4.35) are equal, we get

$$\mathbb{P}(\tilde{h}(i) = \tilde{h}(i')) = \sum_{l \in [L]} 2\frac{1}{L}\frac{I-L}{I(I-1)} + \frac{1}{L^2}\frac{(I-L)(I-L-1)}{I(I-1)} = \frac{1}{L} - \frac{L-1}{I(I-1)}.$$

Proceeding with the remainder of the proof of Lemma 29 as before, we now get a bound

$$\mathbb{E}[\|\boldsymbol{E}\|^2] \leq (K^2+K)\Big(\frac{1}{L} - \frac{L-1}{I(I-1)}\Big).$$

Using this new bound and the new condition

$$\Big(\frac{\alpha}{\alpha-1}\Big)^2\Big(1 - \frac{L(L-1)}{I(I-1)}\Big)\beta(K^2+K) \leq L < I \tag{4.39}$$

together with Markov's inequality, we get

$$\mathbb{P}\Big(\|\boldsymbol{E}\| \geq 1 - \frac{1}{\alpha}\Big) \leq \mathbb{P}\Big(\|\boldsymbol{E}\|^2 \geq \frac{1}{L}\Big(1 - \frac{L(L-1)}{I(I-1)}\Big)\beta(K^2+K)\Big) \leq \frac{1}{\beta},$$

and consequently

$$\mathbb{P}\Big(\|\boldsymbol{E}\| \leq 1 - \frac{1}{\alpha}\Big) \geq 1 - \frac{1}{\beta}$$

holds in this case too.

All the other lemmas will remain the same, with the only exception that Lemmas 30 and 32 now will use the new condition in (4.39) instead of the old one in (4.11). The proof of the proposition itself at the end of Section 4.7.1 will therefore remain identical. When using the modified CountSketch, the statements in Proposition 19 will therefore remain true with the new condition in (4.39). Setting $\alpha = 4$ then yields the desired bound. □

4.7.3 Proof of Proposition 22

The following fact will be useful.

Fact 34 (Theorem B.1 in the supplement of Diao et al. [64])**.** Recall that $\tilde{I} = I_1 \cdots I_N$. Let $\boldsymbol{T} \in \mathbb{R}^{L \times \tilde{I}}$ be a TensorSketch operator defined as in Section 4.1.3 in terms of N CountSketch operators.

(i) Suppose $\boldsymbol{A}$ and $\boldsymbol{B}$ are matrices with $\tilde{I}$ rows. For $L \geq (2 + 3^N)/(\varepsilon^2 \delta)$, we have

$$\mathbb{P}(\|\boldsymbol{A}^\top \boldsymbol{T}^\top \boldsymbol{T}\boldsymbol{B} - \boldsymbol{A}^\top \boldsymbol{B}\|_{\mathrm{F}}^2 \leq \varepsilon^2 \|\boldsymbol{A}\|_{\mathrm{F}}^2 \|\boldsymbol{B}\|_{\mathrm{F}}^2) \geq 1 - \delta.$$

(ii) Suppose $\boldsymbol{M} \in \mathbb{R}^{\tilde{I} \times R}$ is any matrix. If $L \geq R^2(2 + 3^N)/(\varepsilon^2 \delta)$, then the following holds with probability at least $1 - \delta$:

$$(\forall \boldsymbol{x} \in \mathbb{R}^R) \quad (1 - \varepsilon)\|\boldsymbol{M}\boldsymbol{x}\| \leq \|\boldsymbol{T}\boldsymbol{M}\boldsymbol{x}\| \leq (1 + \varepsilon)\|\boldsymbol{M}\boldsymbol{x}\|.$$

We break the proof into two parts. First, we prove Lemma 35 which is a variant of Proposition 19 for the case when a TensorSketch operator $\boldsymbol{T} \in \mathbb{R}^{L \times \tilde{I}}$ is used instead of a CountSketch operator. Then we prove the proposition itself.

Lemma 35. *Let α and β be real numbers such that $\alpha, \beta > 1$, and let K, L, R and $I_1, \ldots, I_N$ be positive integers such that $K \leq R$ and*

$$\Big(\frac{\alpha}{\alpha - 1}\Big)^2 (2 + 3^N)\beta K^2 \leq L < \prod_{n=1}^{N} I_n. \tag{4.40}$$

Suppose that the matrix ID on line 4 of Algorithm 6 utilizes SRRQR. Then the outputs $\boldsymbol{P}$ and $\boldsymbol{j}$ on that line will satisfy

$$\begin{aligned} \|\boldsymbol{M}_{:\boldsymbol{j}}\boldsymbol{P} - \boldsymbol{M}\| \leq \sigma_{K+1}(\boldsymbol{M})\big((\sqrt{4K(R-K)+1}+1)\sqrt{\alpha \textstyle\prod_{n=1}^{N} I_n + 1} \\ + \sqrt{4K(R-K)+1}\sqrt{\alpha \textstyle\prod_{n=1}^{N} I_n}\big) \end{aligned} \tag{4.41}$$

with probability at least $1 - \frac{1}{\beta}$.

Proof. Recall from Section 4.1.3 that TensorSketch is defined similarly to CountSketch, but using the hash function H instead of h, and using the diagonal matrix $\boldsymbol{D}^{(S)}$ instead of $\boldsymbol{D}$. Letting $\boldsymbol{\Phi}^{(H)} \in \mathbb{R}^{L \times (I_1 \cdots I_N)}$ be a matrix with $\phi^{(H)}_{H(i)i} = 1$ for $i \in [I_1 \cdots I_N]$, and with all other entries equal to 0, we can write $\boldsymbol{T} = \boldsymbol{\Phi}^{(H)} \boldsymbol{D}^{(S)}$. This means that the proof in Section 4.7.1 largely can be repeated to prove the present lemma. Lemma 28 remains true in its present form when TensorSketch is used instead of CountSketch.

To see that Lemma 29 remains true with the new condition when $\boldsymbol{S}$ is replaced by $\boldsymbol{T}$, let $\boldsymbol{T} \in \mathbb{R}^{L \times \tilde{I}}$ be a TensorSketch operator, and let $\boldsymbol{U} \in \mathbb{R}^{\tilde{I} \times K}$ be a matrix with orthonormal columns. Define $\boldsymbol{C} \stackrel{\text{def}}{=} (\boldsymbol{TU})^\top (\boldsymbol{TU})$, and let $\boldsymbol{E}$ be defined as in (4.9), but in terms of the corresponding quantities from TensorSketch. Then $\boldsymbol{E} = \boldsymbol{C} - \boldsymbol{I}^{(K)}$, according to Lemma 28. Using Fact 34 (i), condition (4.40), and the fact that $\|\boldsymbol{U}\|_{\mathrm{F}}^2 = K$, we have

$$\begin{aligned}
&\mathbb{P}\Big(\|\boldsymbol{E}\| \leq 1 - \frac{1}{\alpha}\Big) \geq \mathbb{P}\Big(\|\boldsymbol{E}\|_{\mathrm{F}} \leq 1 - \frac{1}{\alpha}\Big) \\
&= \mathbb{P}\bigg(\|(\boldsymbol{TU})^\top (\boldsymbol{TU}) - \boldsymbol{I}^{(K)}\|_{\mathrm{F}}^2 \leq \Big(1 - \frac{1}{\alpha}\Big)^2\bigg) \geq 1 - \frac{1}{\beta}.
\end{aligned}$$

All the other lemmas will remain the same when $\boldsymbol{T}$ is used instead of $\boldsymbol{S}$, with the only exception that Lemmas 30 and 32 now will use the new condition in (4.40) instead of the old one in (4.11). Using exactly the same arguments as in the proof of Proposition 19 at the end of Section 4.7.1 will therefore give the bound in (4.41). □

Proof of Proposition 22. Recall that $\mathcal{X}$ and $\hat{\mathcal{X}}$ are defined as in (4.1) and (4.2), respectively, and the coefficients $\hat{\lambda}_1, \ldots, \hat{\lambda}_K$ are defined as

$$\hat{\lambda}_k = \lambda_{j_k} \sum_{r=1}^{R} p_{kr}$$

for $k \in [K]$. We then have

$$\begin{aligned}
\|\hat{\mathcal{X}} - \mathcal{X}\|_{\mathrm{F}} &= \Big\| \sum_{k \in [K]} \hat{\lambda}_k \mathcal{X}^{(j_k)} - \sum_{r \in [R]} \lambda_r \mathcal{X}^{(r)} \Big\|_{\mathrm{F}} \\
&= \Big\| \sum_{k \in [K]} \Big(\lambda_{j_k} \sum_{r \in [R]} p_{kr}\Big) \mathcal{X}^{(j_k)} - \sum_{r \in [R]} \lambda_r \mathcal{X}^{(r)} \Big\|_{\mathrm{F}} \\
&= \Big\| \sum_{r \in [R]} \Big(\sum_{k \in [K]} \lambda_{j_k} \mathcal{X}^{(j_k)} p_{kr} - \lambda_r \mathcal{X}^{(r)} \Big) \Big\|_{\mathrm{F}}.
\end{aligned} \tag{4.42}$$

Letting $\mathcal{I} \stackrel{\text{def}}{=} [I_1] \times \cdots \times [I_N]$, we have

$$\begin{aligned}
\Big\| \sum_{r\in[R]} \Big(\sum_{k\in[K]} \lambda_{j_k} \boldsymbol{\mathcal{X}}^{(j_k)} p_{kr} - \lambda_r \boldsymbol{\mathcal{X}}^{(r)} \Big) \Big\|_{\mathrm{F}}^2 &= \sum_{\boldsymbol{i}\in\mathcal{I}} \Big(\sum_{r\in[R]} \Big(\sum_{k\in[K]} \lambda_{j_k} x_{\boldsymbol{i}}^{(j_k)} p_{kr} - \lambda_r x_{\boldsymbol{i}}^{(r)} \Big) \Big)^2 \\
&\le \sum_{\boldsymbol{i}\in\mathcal{I}} R \sum_{r\in[R]} \Big(\sum_{k\in[K]} \lambda_{j_k} x_{\boldsymbol{i}}^{(j_k)} p_{kr} - \lambda_r x_{\boldsymbol{i}}^{(r)} \Big)^2 \qquad (4.43) \\
&= R \|\boldsymbol{M}_{:\boldsymbol{j}} \boldsymbol{P} - \boldsymbol{M}\|_{\mathrm{F}}^2,
\end{aligned}$$

where the inequality follows from Cauchy–Schwarz inequality. Combining (4.42) and (4.43) we get

$$\|\hat{\boldsymbol{\mathcal{X}}} - \boldsymbol{\mathcal{X}}\|_{\mathrm{F}} \le \sqrt{R} \|\boldsymbol{M}_{:\boldsymbol{j}} \boldsymbol{P} - \boldsymbol{M}\|_{\mathrm{F}} \le R \|\boldsymbol{M}_{:\boldsymbol{j}} \boldsymbol{P} - \boldsymbol{M}\|, \qquad (4.44)$$

where the second inequality is a well-known relation (see e.g. equation (2.3.7) in [83]). Combining (4.44) and Lemma 35 gives that

$$\begin{aligned}
\|\hat{\boldsymbol{\mathcal{X}}} - \boldsymbol{\mathcal{X}}\|_{\mathrm{F}} \le \sigma_{K+1}(\boldsymbol{M}) R \big(&(\sqrt{4K(R-K)+1} + 1) \sqrt{\alpha \textstyle\prod_{n=1}^N I_n + 1} \\
&+ \sqrt{4K(R-K)+1} \sqrt{\alpha \textstyle\prod_{n=1}^N I_n} \big)
\end{aligned}$$

with probability at least $1 - \frac{1}{\beta}$. Setting $\alpha = 4$ then yields the same bounds as in the statement in Proposition 22. □

4.7.4 Proof of Proposition 23

Proof. Note that $\boldsymbol{TM}$ is of size $L \times R$, with $L > R$. So $\sigma_R(\boldsymbol{TM})$ is the smallest singular value of $\boldsymbol{TM}$. Suppose

$$\Big(\frac{\alpha}{\alpha - 1}\Big)^2 \beta R^2 (2 + 3^N) \le L.$$

To simplify notation, let $\varepsilon \stackrel{\text{def}}{=} 1 - 1/\alpha$. Using Theorem 8.6.1 in [83] and Fact 34 (ii), we have that with probability at least $1 - \frac{1}{\beta}$, the following hold:

$$\sigma_1(\boldsymbol{TM}) = \max_{\|\boldsymbol{x}\|=1} \|\boldsymbol{TMx}\| \le (1+\varepsilon) \max_{\|\boldsymbol{x}\|=1} \|\boldsymbol{Mx}\| = (1+\varepsilon)\sigma_{\max}(\boldsymbol{M}),$$

and

$$\sigma_R(\boldsymbol{TM}) = \min_{\|\boldsymbol{x}\|=1} \|\boldsymbol{TMx}\| \ge (1-\varepsilon) \min_{\|\boldsymbol{x}\|=1} \|\boldsymbol{Mx}\| = (1-\varepsilon)\sigma_{\min}(\boldsymbol{M}).$$

We therefore have

$$\kappa(\boldsymbol{TM}) = \frac{\sigma_1(\boldsymbol{TM})}{\sigma_R(\boldsymbol{TM})} \leq \frac{(1+\varepsilon)\sigma_{\max}(\boldsymbol{M})}{(1-\varepsilon)\sigma_{\min}(\boldsymbol{M})} = \frac{(2-\frac{1}{\alpha})\sigma_{\max}(\boldsymbol{M})}{\frac{1}{\alpha}\sigma_{\min}(\boldsymbol{M})} = (2\alpha-1)\kappa(\boldsymbol{M}),$$

with probability at least $1-\frac{1}{\beta}$. Setting $\alpha = 4$ gives us the bounds in Proposition 23. □

4.8 Conclusion

We have presented a new fast randomized algorithm for computing matrix ID, which utilizes CountSketch. We have then shown how this method can be extended to computing the tensor ID of CP tensors. For both the matrix and tensor settings, we provided performance guarantees. To the best of our knowledge, we provide the first performance guarantees for any randomized tensor ID algorithm. We conducted several numerical experiments on both synthetic and real data. These experiments showed that our algorithms maintain the same accuracy as other randomized methods, but with a much shorter run time, running at least an order of magnitude faster on the larger matrices and tensors.

Chapter 5

Randomization of approximate bilinear computation for matrix multiplication

This is an Accepted Manuscript of an article published by Taylor & Francis in International Journal of Computer Mathematics: Computer Systems Theory on 30 December 2020, available online: http://wwww.tandfonline.com/10.1080/23799927.2020.1861104

5.1 Introduction

Suppose $\boldsymbol{A}, \boldsymbol{B} \in \mathbb{R}^{n\times n}$. In this paper, we are concerned with formulas for computing $\boldsymbol{C} = \boldsymbol{AB}$ that take the form

$$c_{ij} = \sum_{r=1}^{R} w_{ijr}\Big(\sum_{k,l=1}^{n} u_{klr}a_{kl}\Big)\Big(\sum_{k',l'=1}^{n} v_{k'l'r}b_{k'l'}\Big), \tag{5.1}$$

$i, j \in [n] \stackrel{\text{def}}{=} \{1, 2, \ldots, n\}$, where each $\boldsymbol{\mathcal{U}} = (u_{klr})$, $\boldsymbol{\mathcal{V}} = (v_{k'l'r})$, and $\boldsymbol{\mathcal{W}} = (w_{ijr})$ is a tensor containing real numbers, and c_{ij} is the element at position (i, j) in $\boldsymbol{C}$, with similar notation for elements of $\boldsymbol{A}$ and $\boldsymbol{B}$. Such a formula is called a **bilinear computation** (BC) [29]. We can rewrite the expression above as

$$c_{ij} = \sum_{k,l=1}^{n}\sum_{k',l'=1}^{n} a_{kl}b_{k'l'}\sum_{r=1}^{R} u_{klr}v_{k'l'r}w_{ijr}.$$

Comparing this to the standard algorithm for matrix multiplication, we can see that for the computation (5.1) to be exact, $\boldsymbol{\mathcal{U}}$, $\boldsymbol{\mathcal{V}}$ and $\boldsymbol{\mathcal{W}}$ must satisfy

$$\sum_{r=1}^{R} u_{klr}v_{k'l'r}w_{ijr} = \delta_{ki}\delta_{l'j}\delta_{lk'} \quad \text{for all } (k, l, k', l', i, j) \in [n]^6, \tag{5.2}$$

where δ is the Kronecker delta (i.e., $\delta_{ki} = 1$ if $k = i$ and 0 otherwise) [35]. If (5.2) is satisfied, we say that (5.1) is an **exact bilinear computation** (EBC). The smallest positive integer R for which

there exist $\boldsymbol{\mathcal{U}}$, $\boldsymbol{\mathcal{V}}$ and $\boldsymbol{\mathcal{W}}$ such that (5.2) holds is referred to as the **rank** of the computation. We refer to any algorithm of the form (5.1) for which $R < n^3$ as **fast**, since the asymptotic complexity is smaller than that of standard matrix multiplication, which has complexity $O(n^3)$. Examples of fast computations of the form (5.1) include Strassen's algorithm for 2×2 matrices [191], and Laderman's algorithm for 3×3 matrices [123]. The formula (5.1) is also valid if $\boldsymbol{A}$, $\boldsymbol{B}$ and $\boldsymbol{C}$ are of size $mn \times mn$ and a_{kl}, $b_{k'l'}$ and c_{ij} are replaced by submatrices $\boldsymbol{A}_{kl}$, $\boldsymbol{B}_{k'l'}$ and $\boldsymbol{C}_{ij}$ of size $m \times m$, so that e.g.

$$\boldsymbol{A} = \begin{bmatrix} \boldsymbol{A}_{11} & \boldsymbol{A}_{12} & \cdots & \boldsymbol{A}_{1n} \\ \boldsymbol{A}_{21} & \boldsymbol{A}_{22} & \cdots & \boldsymbol{A}_{2n} \\ \vdots & \vdots & & \vdots \\ \boldsymbol{A}_{n1} & \boldsymbol{A}_{n2} & \cdots & \boldsymbol{A}_{nn} \end{bmatrix}.$$

This is why fast algorithms of the form (5.1) can be used recursively to compute the product of larger matrices.

Define 6-way tensors

$$\boldsymbol{\mathcal{X}} \stackrel{\text{def}}{=} (\delta_{ki}\delta_{l'j}\delta_{lk'})_{(k,l,k',l',i,j)\in[n]^6}, \tag{5.3}$$

$$\boldsymbol{\mathcal{Y}} \stackrel{\text{def}}{=} (\sum_{r=1}^{R} u_{klr}v_{k'l'r}w_{ijr})_{(k,l,k',l',i,j)\in[n]^6}. \tag{5.4}$$

The condition in (5.2) can be written succinctly as $\boldsymbol{\mathcal{Y}} = \boldsymbol{\mathcal{X}}$. For both matrix and tensor inputs, let $\|\cdot\|$ denote the Frobenius norm, i.e., the square root of the sum of the square of all elements. We are interested in BCs that are only approximately correct, which motivates the following definition.

Definition 36 (Approximate bilinear computation)**.** Let $\boldsymbol{A}, \boldsymbol{B} \in \mathbb{R}^{mn\times mn}$, where m is a positive integer, and let f be defined blockwise via

$$f(\boldsymbol{A},\boldsymbol{B})_{ij} \stackrel{\text{def}}{=} \sum_{r=1}^{R} w_{ijr}\Big(\sum_{k,l=1}^{n} u_{klr}\boldsymbol{A}_{kl}\Big)\Big(\sum_{k',l'=1}^{n} v_{k'l'r}\boldsymbol{B}_{k'l'}\Big) \in \mathbb{R}^{m\times m} \quad \text{for } i,j \in [n], \tag{5.5}$$

where $f(\boldsymbol{A},\boldsymbol{B})_{ij}$, $\boldsymbol{A}_{kl}$ and $\boldsymbol{B}_{k'l'}$ are submatrices of $f(\boldsymbol{A},\boldsymbol{B})$, $\boldsymbol{A}$ and $\boldsymbol{B}$, respectively, of size $m \times m$. We say that f is an **approximate bilinear computation** (ABC) with parameters n and τ, or (n,τ)-ABC for short, if $\|\boldsymbol{\mathcal{Y}} - \boldsymbol{\mathcal{X}}\| \le \tau$.

We present a method for randomizing the computation (5.5) and consider the implications of this approach when there are two kinds of error present.

- **Error due to an approximate algorithm.** Using an (n, τ)-ABC with $\tau > 0$ will introduce error even if exact arithmetic is used. ABCs are interesting since it is sometimes hard or impossible to convert an approximate algorithm found numerically to an exact algorithm. Some examples of such approximate algorithms found numerically appear in [188]. Other papers that search for fast algorithms numerically include [18, 35, 75, 106].

- **Numerical error due to using floating point arithmetic.** Although the standard algorithm for matrix multiplication also incurs numerical error, it is more severe in fast algorithms since the computation of an element c_{ij} can involve elements from $\boldsymbol{A}$ and $\boldsymbol{B}$ other than the vectors $\boldsymbol{a}_{i:}$ and $\boldsymbol{b}_{:j}$, which are the ith row of $\boldsymbol{A}$ and jth column of $\boldsymbol{B}$, respectively. In exact arithmetic, these additional terms cancel out for exact fast algorithms, but in finite precision those cancellations are typically not exact which can lead to substantial numerical error. These issues are exacerbated when the algorithm itself is only approximately correct. These considerations are especially important when using low precision environments, such as when computing on a GPU [99]. Low precision computation—using 32-bit, 16-bit, and even 8-bit precision numbers—is popular in, e.g., machine learning [89, 95, 207].

We make the following contributions in this paper:

- We propose a method for randomizing BCs for matrix multiplication which does not increase the leading order computational complexity of the algorithm.

- When exact arithmetic is used, we show that our randomized ABCs compute the correct matrix product in expectation. We also provide some performance guarantees.

- We show that these exact arithmetic results largely carry over to a setting when all computations are done in floating point arithmetic.

- When floating point arithmetic is used, we provide numerical evidence that randomizing EBCs using our scheme can reduce numerical error and improve robustness to adversarial examples.

5.1.1 Related work

Bini et al. [26, 27] introduce a concept similar to ABC called Arbitrary Precision Approximating (APA) algorithms for matrix multiplication. For an APA algorithm, the tensor $\mathcal{X}$ and $\mathcal{Y}$ defined in (5.3) and (5.4) satisfy the relationship

$$\mathcal{Y}(\varepsilon) + \mathcal{E}(\varepsilon) = \mathcal{X},$$

where

$$\mathcal{Y}(\varepsilon) \stackrel{\text{def}}{=} \Big(\sum_{r=1}^{R} u_{klr}(\varepsilon) v_{k'l'r}(\varepsilon) w_{ijr}(\varepsilon)\Big)_{(k,l,k',l',i,j)\in[n]^6} \tag{5.6}$$

is a function of ε, and $\mathcal{E}$ represents the error in the approximation $\mathcal{Y} \approx \mathcal{X}$. Each entry in $\mathcal{E}$ is assumed to be a polynomial with zero constant coefficient, hence for every entry $\mathcal{E}_{klk'l'ij}(0) = 0$. Consequently, an APA algorithm can be made arbitrarily accurate by making ε small enough. Moreover, as shown in [24], an EBC can be derived from an APA algorithm (see also the related discussion in Section 15.2 of [37]). This EBC takes the form

$$\sum_{i=1}^{d+1} \alpha_i \mathcal{Y}(\varepsilon_i) = \mathcal{X}, \tag{5.7}$$

where all ε_i are distinct, $d := \max_{klk'l'ij\in[n]^6} \deg(\mathcal{E}_{klk'l'ij}(\varepsilon))$ is the maximum degree of the polynomials describing the entries in $\mathcal{E}$, and where $[\alpha_1, \ldots, \alpha_{d+1}]$ is chosen as the solution to a certain linear system. Bini [24] uses this approach to convert the APA scheme for 12×12 matrix multiplication in [26] to an exact one, which can be used recursively to get a matrix multiplication algorithm with complexity $O(n^{2.7799})$. Fixing the error in an APA algorithm yields an ABC with some parameter τ. In some of our numerical experiments, we use an ABC which we get by fixing ε in the 12×12 APA scheme of [26]. We also consider an associated EBC derived via (5.7) in other experiments. Our paper focuses on ABCs instead of APA algorithms since, in practice, it may be difficult or impossible to express a pair $(\mathcal{Y}, \mathcal{E})$ found numerically in terms of polynomials in ε.

To the best of our knowledge, our paper is the first to consider randomization as a tool for improving BCs for matrix multiplication which are only approximately correct. Various randomized algorithms for the standard matrix multiplication algorithm have been considered in other works; see e.g. [70, 170]. For EBCs in floating point arithmetic, a patent by Castrapel and Gustafson [43] describes a randomized version of Strassen's algorithm which they claim reduces numerical error. They provide empirical support for this, but no mathematical proof. Our method generalizes their approach by randomly choosing from a wider range of equivalent algorithms. Additionally, our method can be applied to any formula of the form (5.1).

Early works that analyze the stability of fast algorithms for matrix multiplication include [25, 27, 36]. Other works, such as [12, 55, 59, 73, 108], attempt to improve the stability of fast algorithms using other approaches that do not rely on randomization. In Section 5.3.2, we compare our proposed method to the rescaling method in [12]. In these experiments, we also consider a restricted version of our method which corresponds to the method in [43].

5.2 Randomization of bilinear computation for matrix multiplication

In Section 5.2.1, we first consider a setting in which exact arithmetic is used. In Section 5.2.2, we then consider a setting in which floating point arithmetic is used.

5.2.1 In exact arithmetic

We present our randomization scheme in the setting when the input matrices are $mn \times mn$ with $m \geq 1$[1] . Accordingly, suppose $f : \mathbb{R}^{mn\times mn} \times \mathbb{R}^{mn\times mn} \to \mathbb{R}^{mn\times mn}$ is an (n, τ)-ABC. We will now define a randomized version of f, denoted by $\hat{f}$, which has the following property: For all $\boldsymbol{A}, \boldsymbol{B} \in \mathbb{R}^{mn\times mn}$, $\mathbb{E}[\hat{f}(\boldsymbol{A}, \boldsymbol{B})] = \boldsymbol{AB}$. To that end, let $\{s_i(j)\}_{(i,j)\in[3]\times[n]}$ be a collection of i.i.d. Rademacher random variables, i.e., each satisfying $\mathbb{P}[s_i(j) = +1] = \mathbb{P}[s_i(j) = -1] = 1/2$. Moreover, let $\pi_i : [n] \to [n]$, $i \in [3]$, be independent random permutation functions, each satisfying $(\forall (j, k) \in [n]^2)\ \mathbb{P}[\pi_i(j) = k] = 1/n$. Let $\boldsymbol{S}_i \in \mathbb{R}^{mn\times mn}$, $i \in [3]$, be block diagonal matrices with the

[1] We present results for square matrices, but we believe the results can be extended to rectangular matrices.

jth nonzero block equal to $s_i(j)\boldsymbol{I}_m$, where $\boldsymbol{I}_m$ is the $m \times m$ identity matrix. Also, let $\boldsymbol{P}_i \in \mathbb{R}^{mn\times mn}$, $i \in [3]$, be permutation matrices divided into $m \times m$ blocks, with blocks on position $(\pi_i(j), j)$, $j \in [n]$, equal to $\boldsymbol{I}_m$ and all other blocks equal to zero. Define $\boldsymbol{M}_i = \boldsymbol{P}_i\boldsymbol{S}_i$, $i \in [3]$. Note that each $\boldsymbol{M}_i$ is orthogonal (i.e., $\boldsymbol{M}_i^{-1} = \boldsymbol{M}_i^\top$). We propose the following definition of $\hat{f}$.

Definition 37 (Randomized approximate bilinear computation)**.** Let f be an (n, τ)-ABC. We define a corresponding **randomized approximate bilinear computation** with parameters n, τ and κ, or (n, τ, κ)-RandABC for short, via

$$\hat{f}(\boldsymbol{A}, \boldsymbol{B}) \stackrel{\text{def}}{=} (1-\kappa)^{-1}\boldsymbol{M}_1^\top f(\boldsymbol{M}_1\boldsymbol{A}\boldsymbol{M}_2^\top, \boldsymbol{M}_2\boldsymbol{B}\boldsymbol{M}_3^\top)\boldsymbol{M}_3, \tag{5.8}$$

where

$$\kappa \stackrel{\text{def}}{=} \frac{1}{n^3}\sum_{(i,j,l)\in[n]^3}\Big(1 - \sum_{r=1}^{R} u_{ilr}v_{ljr}w_{ijr}\Big) \tag{5.9}$$

is assumed to satisfy $\kappa \neq 1$.

Observe that if f was an exact algorithm, then $\kappa = 0$ and $\hat{f}(\boldsymbol{A}, \boldsymbol{B}) = \boldsymbol{A}\boldsymbol{B}$ since the $\boldsymbol{M}_i$'s would cancel out due to orthogonality. Since the cost of applying $\boldsymbol{M}_i$ to an $mn \times mn$ matrix is $O(m^2n^2)$, computing $\hat{f}(\boldsymbol{A}, \boldsymbol{B})$ has the same leading order complexity as computing $f(\boldsymbol{A}, \boldsymbol{B})$.

Proposition 38. *Let $\hat{f}$ be an (n, τ, κ)-RandABC with $\tau \geq 0$ and $\kappa \neq 1$. For all $\boldsymbol{A}, \boldsymbol{B} \in \mathbb{R}^{mn\times mn}$ we have $\mathbb{E}[\hat{f}(\boldsymbol{A}, \boldsymbol{B})] = \boldsymbol{A}\boldsymbol{B}$.*

Proof. Let $\hat{\boldsymbol{C}} \stackrel{\text{def}}{=} \hat{f}(\boldsymbol{A}, \boldsymbol{B})$. Considering the (i, j)th block of $\hat{\boldsymbol{C}}$, and going through some tedious but straightforward algebra, we get

$$\hat{\boldsymbol{C}}_{ij} = \frac{s_1(i)s_3(j)}{1-\kappa}\sum_{k,l=1}^{n}\sum_{k',l'=1}^{n} s_1(k)s_2(l)s_2(k')s_3(l')\boldsymbol{A}_{kl}\boldsymbol{B}_{k'l'}\sum_{r=1}^{R} u_{\pi_1(k)\pi_2(l)r}v_{\pi_2(k')\pi_3(l')r}w_{\pi_1(i)\pi_3(j)r}. \tag{5.10}$$

If we take the expectation of this equation with respect to the random variables $\{s_i(j)\}$, most terms will vanish: If $i = k$, $l = k'$ and $j = l'$ for a given term, then the product of the s_i's will be 1; otherwise, the expectation of that term will be zero due to independence and the fact that each $\mathbb{E}[s_i(j)] = 0$. Consequently, we have

$$\mathbb{E}[\hat{\boldsymbol{C}}_{ij}] = (1-\kappa)^{-1}\sum_{l=1}^{n}\boldsymbol{A}_{il}\boldsymbol{B}_{lj}\mathbb{E}\Big[\sum_{r=1}^{R} u_{\pi_1(i)\pi_2(l)r}v_{\pi_2(l)\pi_3(j)r}w_{\pi_1(i)\pi_3(j)r}\Big] = \sum_{l=1}^{n}\boldsymbol{A}_{il}\boldsymbol{B}_{lj} = (\boldsymbol{A}\boldsymbol{B})_{ij},$$

where the second equality is true since (5.9) implies that

$$\mathbb{E}\Big[\sum_{r=1}^{R} u_{\pi_1(i)\pi_2(l)r} v_{\pi_2(l)\pi_3(j)r} w_{\pi_1(i)\pi_3(j)r}\Big] = 1-\kappa.$$

□

It may seem surprising that Proposition 38 holds for **any** $\tau \geq 0$, since this means that even an ABC with an arbitrarily large error will be correct in expectation once randomized as in Definition 37. However, as we will see in Proposition 39, in order to be able to guarantee a small error for **any realization** of $\hat{f}(\boldsymbol{A},\boldsymbol{B})$, τ also needs to be small.

Define $B_\mu^{(n)} \stackrel{\text{def}}{=} \{\boldsymbol{M} \in \mathbb{R}^{n\times n} : \|\boldsymbol{M}\| \leq \mu\}$ and $\eta \stackrel{\text{def}}{=} (1-\kappa)^{-1} - 1 = O(\kappa)$. The following proposition provides performance guarantees for f and $\hat{f}$.

Proposition 39. *Consider matrices* $\boldsymbol{A},\boldsymbol{B} \in B_\mu^{(mn)}$.

(i) If f *is an* (n,τ)*-ABC, then* $\|f(\boldsymbol{A},\boldsymbol{B}) - \boldsymbol{AB}\| \leq \mu^2\tau$.

(ii) If $\hat{f}$ *is an* (n,τ,κ)*-RandABC, then* $\|\hat{f}(\boldsymbol{A},\boldsymbol{B}) - \boldsymbol{AB}\| \leq \mu^2|\eta|\|\mathfrak{Y}\| + \mu^2\tau \lesssim \mu^2|\kappa|\|\mathfrak{Y}\| + \mu^2\tau$.

(iii) Moreover, $\sup_{\boldsymbol{A},\boldsymbol{B}\in B_\mu^{(mn)}} \|\hat{f}(\boldsymbol{A},\boldsymbol{B}) - \boldsymbol{AB}\| \leq \sup_{\boldsymbol{A},\boldsymbol{B}\in B_\mu^{(mn)}} \|f(\boldsymbol{A},\boldsymbol{B}) - \boldsymbol{AB}\| + \mu^2|\eta|\|\mathfrak{Y}\|$.

Proof. We have

$$\begin{aligned}
\|f(\boldsymbol{A},\boldsymbol{B}) - \boldsymbol{AB}\|^2 &= \sum_{i,j=1}^{n} \Big\| \sum_{k,l=1}^{n}\sum_{k',l'=1}^{n} \boldsymbol{A}_{kl}\boldsymbol{B}_{k'l'}(y_{klk'l'ij} - x_{klk'l'ij})\Big\|^2 \\
&\leq \sum_{i,j=1}^{n}\Big(\sum_{k,l=1}^{n}\sum_{k',l'=1}^{n} \|\boldsymbol{A}_{kl}\|\|\boldsymbol{B}_{k'l'}\||y_{klk'l'ij} - x_{klk'l'ij}|\Big)^2 \\
&\leq \sum_{i,j=1}^{n}\Big(\sum_{k,l=1}^{n}\sum_{k',l'=1}^{n} (\|\boldsymbol{A}_{kl}\|\|\boldsymbol{B}_{k'l'}\|)^2\Big)\Big(\sum_{k,l=1}^{n}\sum_{k',l'=1}^{n} |y_{klk'l'ij} - x_{klk'l'ij}|^2\Big) \\
&= \Big(\sum_{k,l=1}^{n}\|\boldsymbol{A}_{kl}\|^2 \sum_{k',l'=1}^{n}\|\boldsymbol{B}_{k'l'}\|^2\Big)\Big(\sum_{i,j=1}^{n}\sum_{k,l=1}^{n}\sum_{k',l'=1}^{n} |y_{klk'l'ij} - x_{klk'l'ij}|^2\Big) \\
&= \|\boldsymbol{A}\|^2\|\boldsymbol{B}\|^2\|\mathfrak{Y} - \mathfrak{X}\|^2 \leq \mu^4\tau^2,
\end{aligned}$$

where the first inequality follows from first applying the triangle inequality and then using the sub-multiplicativity of the Frobenius norm, and the second inequality follows from applying the

Cauchy–Schwarz inequality. This proves (i). Since the Frobenius norm is invariant under unitary transformations,

$$\|\hat{f}(\boldsymbol{A},\boldsymbol{B}) - \boldsymbol{A}\boldsymbol{B}\| = \|(1-\kappa)^{-1} f(\tilde{\boldsymbol{A}},\tilde{\boldsymbol{B}}) - \tilde{\boldsymbol{A}}\tilde{\boldsymbol{B}}\|, \tag{5.11}$$

where $\tilde{\boldsymbol{A}} = \boldsymbol{M}_1\boldsymbol{A}\boldsymbol{M}_2^\top$, $\tilde{\boldsymbol{B}} = \boldsymbol{M}_2\boldsymbol{B}\boldsymbol{M}_3^\top$. Going through the same computations as in the proof of (i), but with the extra $(1-\kappa)^{-1}$ term, we therefore get

$$\|\hat{f}(\boldsymbol{A},\boldsymbol{B}) - \boldsymbol{A}\boldsymbol{B}\| \leq \|\tilde{\boldsymbol{A}}\|\|\tilde{\boldsymbol{B}}\|\|(1-\kappa)^{-1}\mathcal{Y} - \mathcal{X}\| = \|\boldsymbol{A}\|\|\boldsymbol{B}\|\|(1-\kappa)^{-1}\mathcal{Y} - \mathcal{X}\| \leq \mu^2(|\eta|\|\mathcal{Y}\| + \tau)$$

where the equality once again uses the unitary invariance of the Frobenius norm, and the last inequality follows from the definitions of η and the triangle inequality. This proves (ii). Fix $\boldsymbol{A},\boldsymbol{B} \in B_\mu^{(mn)}$. Applying the triangle inequality to (5.11) and using the definition of η, we get

$$\|\hat{f}(\boldsymbol{A},\boldsymbol{B}) - \boldsymbol{A}\boldsymbol{B}\| \leq \|f(\tilde{\boldsymbol{A}},\tilde{\boldsymbol{B}}) - \tilde{\boldsymbol{A}}\tilde{\boldsymbol{B}}\| + |\eta|\,\|f(\tilde{\boldsymbol{A}},\tilde{\boldsymbol{B}})\|. \tag{5.12}$$

By doing computations almost identical to those in the proof of (i), we get the bound

$$\|f(\tilde{\boldsymbol{A}},\tilde{\boldsymbol{B}})\| \leq \|\tilde{\boldsymbol{A}}\|\|\tilde{\boldsymbol{B}}\|\|\mathcal{Y}\| \leq \mu^2\|\mathcal{Y}\|, \tag{5.13}$$

since $\tilde{\boldsymbol{A}},\tilde{\boldsymbol{B}} \in B_\mu^{(mn)}$. Combining (5.12) and (5.13) and taking supremums appropriately proves (iii). □

Points (i) and (ii) in Proposition 39 provide absolute performance guarantees for f and $\hat{f}$, respectively. Note that a tighter version of (i) holds, with $\|\boldsymbol{A}\|$ replaced by $\sqrt{\sum_{kl}\|\boldsymbol{A}_{kl}\|_2^2}$, where $\|\cdot\|_2$ is the spectral norm. We keep (i) in its current looser form to make it easier to compare to (ii) and (iii). Point (iii) shows that the worst case performance of $\hat{f}$ is no worse than that of f plus a constant. In fact, that constant, which also appears in (ii), can be bounded as follows.

Proposition 40. *If* $|\kappa| \leq 1/2$, *then*

$$|\eta|\mu^2\|\mathcal{Y}\| \leq 2\mu^2(n^{-5/2}\tau + n^{-1})\tau.$$

Proof. Let $\boldsymbol{z} \in \mathbb{R}^{n^3}$ be the vector with elements $(\sum_{r=1}^R u_{ilr}v_{ljr}w_{ijr} - 1)$ for $(i,j,l) \in [n]^3$, i.e., containing the elements of $(\mathcal{Y} - \mathcal{X})$ in positions for which $\mathcal{X}$ has an entry 1 (the element order in $\boldsymbol{z}$

is irrelevant). Note that

$$|\kappa| = \left| n^{-3} \sum_{i=1}^{n^3} z_i \right| \leq n^{-3} \|\boldsymbol{z}\|_1 \leq n^{-3} \sqrt{n} \|\boldsymbol{z}\| \leq n^{-5/2} \|\boldsymbol{\mathcal{Y}} - \boldsymbol{\mathcal{X}}\|, \tag{5.14}$$

where the first equality follows from (5.9), the first inequality follows from the triangle inequality and the definition of the 1-norm, and the second inequality is a well known relation (see e.g. Equation (2.2.5) in [83]). Now, note that

$$\|\boldsymbol{\mathcal{Y}}\| \leq \|\boldsymbol{\mathcal{Y}} - \boldsymbol{\mathcal{X}}\| + \|\boldsymbol{\mathcal{X}}\| = \|\boldsymbol{\mathcal{Y}} - \boldsymbol{\mathcal{X}}\| + n^{3/2}. \tag{5.15}$$

Combining (5.14), (5.15), the fact that $|\eta| \leq 2|\kappa|$ when $|\kappa| \leq 1/2$, and the definition of τ gives us the desired bound. □

The upper bound $\mu^2\tau$ in Proposition 39 (i) is also an upper bound to the sup term on the right hand side of the inequality in (iii) of the same proposition. Proposition 40 shows that the size of the additional constant $|\eta|\mu^2\|\boldsymbol{\mathcal{Y}}\|$ is not much larger than the bound that we already have on this sup term, and that it will be smaller than that bound if e.g. $n \geq 3$ and $\tau < 3^{3/2}/2$.

As is clear from the proof of Proposition 38, the constant κ in (5.9) is used to rescale the computation in (5.8) so that the output of $\hat{f}$ is correct in expectation. It is important to note that κ, which can be both positive and negative, is **not** a measure of error in the algorithm. Indeed, setting $R = 1$ and all elements of $\boldsymbol{\mathcal{U}}$, $\boldsymbol{\mathcal{V}}$ and $\boldsymbol{\mathcal{W}}$ to 1 would result in $\kappa = 0$, but this would clearly be a very poor algorithm. Although the corresponding randomized algorithm $\hat{f}$ would be correct in expectation, the error guarantees in Proposition 39 (i) and (ii) would be very poor, since τ would be large.

5.2.1.1 A recursive algorithm for approximate bilinear computation

We can extend the result in Proposition 38 to a recursive version of $\hat{f}$. We denote the recursive algorithm with Q recursions for multiplication of $mn^Q \times mn^Q$ matrices by $\hat{F}^{(Q)}$. Let $\{s_i^{(q)}(j)\}_{(i,j,q)\in[3]\times[n]\times[Q]}$ be a collection of i.i.d. Rademacher random variables, and let $\pi_i^{(q)} : [n] \to [n]$, $(i, q) \in [3] \times [Q]$, be independent random permutation functions, each satisfying $(\forall (j, k) \in [n]^2)$

$\mathbb{P}[\pi_i^{(q)}(j) = k] = 1/n$. Let $\hat{F}^{(1)}$ be defined exactly as $\hat{f}$ in (5.8) but based on the random variables $\{s_i^{(1)}(j)\}_{(i,j)\in[3]\times[n]}$ and $\{\pi_i^{(1)}\}_{i\in[3]}$ and define $\hat{F}^{(q)} : \mathbb{R}^{mn^q\times mn^q} \times \mathbb{R}^{mn^q\times mn^q} \to \mathbb{R}^{mn^q\times mn^q}$, $q \in \{2, 3, \ldots, Q\}$, recursively via

$$(\hat{F}^{(q)}(\boldsymbol{A}, \boldsymbol{B}))_{ij} \stackrel{\text{def}}{=} (1-\kappa)^{-1} s_1^{(q)}(i) s_3^{(q)}(j) \sum_{r=1}^{R} w_{\pi_1^{(q)}(i)\pi_3^{(q)}(j)r} \hat{F}^{(q-1)}(\boldsymbol{A}^{(q)}, \boldsymbol{B}^{(q)}), \tag{5.16}$$

where

$$\boldsymbol{A}^{(q)} \stackrel{\text{def}}{=} \sum_{k,l=1}^{n} u_{\pi_1^{(q)}(k)\pi_2^{(q)}(l)r} s_1^{(q)}(k) s_2^{(q)}(l) \boldsymbol{A}_{kl},$$

$$\boldsymbol{B}^{(q)} \stackrel{\text{def}}{=} \sum_{k',l'=1}^{n} v_{\pi_2^{(q)}(k')\pi_3^{(q)}(l')r} s_2^{(q)}(k') s_3^{(q)}(l') \boldsymbol{B}_{k'l'},$$

and $(\hat{F}^{(q)}(\boldsymbol{A}, \boldsymbol{B}))_{ij}$ is the subblock of size $mn^{q-1} \times mn^{q-1}$ on position (i, j). If $\boldsymbol{A}$ and $\boldsymbol{B}$ are of size $p \times p$ but there is no integer m such that $p = mn^Q$, e.g. if p is prime, then we can simply pad the matrices appropriately [101]. If the recursive formula (5.16) is used Q times to compute the product of two $N \times N$ matrices, where $N \stackrel{\text{def}}{=} mn^Q$, and $\hat{F}^{(0)}(\boldsymbol{A}^{(0)}, \boldsymbol{B}^{(0)}) = \boldsymbol{A}^{(0)}\boldsymbol{B}^{(0)}$ is computed via the standard matrix multiplication formula, the asymptotic cost of the recursive algorithm is $O(m^{3-\log_n R} N^{\log_n R})$. This is the same leading order complexity as the corresponding computation without any randomization. For example, if we insert $m = 1$, $n = 2$ and $R = 7$, we recover the asymptotic cost of Strassen's algorithm: $O(N^{\log_2 7}) \approx O(N^{2.81})$.

Proposition 41. *For any positive integer Q and for all $\boldsymbol{A}, \boldsymbol{B} \in \mathbb{R}^{mn^Q\times mn^Q}$ we have $\mathbb{E}[\hat{F}^{(Q)}(\boldsymbol{A}, \boldsymbol{B})] = \boldsymbol{AB}$.*

Proof. Note that the claim is true for $Q = 1$ due to Proposition 38. Now assume it is true for some $Q \geq 1$. We will show that it is also true for $Q + 1$, i.e., that $\mathbb{E}[\hat{F}^{(Q+1)}(\boldsymbol{A}, \boldsymbol{B})] = \boldsymbol{AB}$ for $\boldsymbol{A}, \boldsymbol{B} \in \mathbb{R}^{mn^{Q+1}\times mn^{Q+1}}$. Let $\mathcal{S}$ denote the σ-algebra generated by the random variables $\{s_i^{(Q+1)}(j)\}_{(i,j)\in[3]\times[n]}$ and $\{\pi_i^{(Q+1)}(j)\}_{(i,j)\in[3]\times[n]}$, and let $\mathbb{E}_{\mathcal{S}}[\,\cdot\,] \stackrel{\text{def}}{=} \mathbb{E}[\,\cdot \mid \mathcal{S}]$. Then

$$\mathbb{E}[(\hat{F}^{(Q+1)}(\boldsymbol{A}, \boldsymbol{B}))_{ij}] = \mathbb{E}\big[\mathbb{E}_{\mathcal{S}}[(\hat{F}^{(Q+1)}(\boldsymbol{A}, \boldsymbol{B}))_{ij}]\big]$$

due to the smoothing property of expectation (property 10 in [178, p. 348]),

$$= \mathbb{E}\Big[(1-\kappa)^{-1} s_1^{(Q+1)}(i) s_3^{(Q+1)}(j) \sum_{r=1}^{R} w_{\pi_1^{(Q+1)}(i)\pi_3^{(Q+1)}(j)r} \times \mathbb{E}_{\mathcal{S}}\Big[\hat{F}^{(Q)}\Big(\sum_{k,l=1}^{n} u_{\pi_1^{(Q+1)}(k)\pi_2^{(Q+1)}(l)r} s_1^{(Q+1)}(k) s_2^{(Q+1)}(l) \boldsymbol{A}_{kl}, \sum_{k',l'=1}^{n} v_{\pi_2^{(Q+1)}(k')\pi_3^{(Q+1)}(l')r} s_2^{(Q+1)}(k') s_3^{(Q+1)}(l') \boldsymbol{B}_{k'l'} \Big)\Big]\Big]$$

since each $s_i^{(Q+1)}(j)$ and $\pi_i^{(Q+1)}(j)$ is $\mathcal{S}$-measurable,

$$= \mathbb{E}\Big[(1-\kappa)^{-1} s_1^{(Q+1)}(i) s_3^{(Q+1)}(j) \sum_{r=1}^{R} w_{\pi_1^{(Q+1)}(i)\pi_3^{(Q+1)}(j)r} \times \Big(\sum_{k,l=1}^{n} u_{\pi_1^{(Q+1)}(k)\pi_2^{(Q+1)}(l)r} s_1^{(Q+1)}(k) s_2^{(Q+1)}(l) \boldsymbol{A}_{kl} \Big) \times \Big(\sum_{k',l'=1}^{n} v_{\pi_2^{(Q+1)}(k')\pi_3^{(Q+1)}(l')r} s_2^{(Q+1)}(k') s_3^{(Q+1)}(l') \boldsymbol{B}_{k'l'} \Big)\Big]$$

due to the induction hypothesis and since all random variables $\{s_i^{(q)}(j)\}_{(i,j,q)\in[3]\times[n]\times[Q]}$ and $\{\pi_i^{(q)}(j)\}_{(i,j,q)\in[3]\times[n]\times[Q]}$ are independent of $\mathcal{S}$ (and using property 12 in [178, pp. 349–350]),

$$= \mathbb{E}\Big[(1-\kappa)^{-1} s_1^{(Q+1)}(i) s_3^{(Q+1)}(j) \sum_{k,l=1}^{n} \sum_{k',l'=1}^{n} s_1^{(Q+1)}(k) s_2^{(Q+1)}(l) s_2^{(Q+1)}(k') s_3^{(Q+1)}(l') \boldsymbol{A}_{kl} \boldsymbol{B}_{k'l'} \times \sum_{r=1}^{R} u_{\pi_1^{(Q+1)}(k)\pi_2^{(Q+1)}(l)r} v_{\pi_2^{(Q+1)}(k')\pi_3^{(Q+1)}(l')r} w_{\pi_1^{(Q+1)}(i)\pi_3^{(Q+1)}(j)r}\Big]$$

by reordering the terms,

$$= \sum_{l=1}^{n} \boldsymbol{A}_{il} \boldsymbol{B}_{lj} = (\boldsymbol{AB})_{ij}$$

which follows by doing the same analysis as in the proof of Proposition 38. So $\mathbb{E}[\hat{F}^{(Q+1)}(\boldsymbol{A}, \boldsymbol{B})] = \boldsymbol{AB}$. The claim in Proposition 41 now follows by induction. □

5.2.2 In floating point arithmetic

In floating point arithmetic, we cannot achieve guarantees like those in Propositions 38 and 41. This is illustrated in the following example.

Example 42. Let $g : \mathbb{R}^{mn\times mn} \times \mathbb{R}^{mn\times mn} \to \mathbb{R}^{mn\times mn}$ be a function that computes matrix multiplication according to some EBC in floating point arithmetic. Let $\hat{g}$ be defined analogously to $\hat{f}$ in

(5.8), but in terms of g instead of f. Since g is exact, we have $\kappa = 0$. We will use $\hat{G}^{(Q)}$ to denote the recursive version of $\hat{g}$, defined analogously to $\hat{F}^{(Q)}$. We will use $G^{(Q)}$ to denote the recursive version of g, defined analogously to $\hat{G}^{(Q)}$ but with each $s_i^{(q)}(j) = 1$ and each $\pi_i^{(q)}(j) = j$, i.e., with no randomness involved so that $G^{(Q)}$ is deterministic. We consider the same setup as in Section 2.7.10 of [83]: Let

$$\boldsymbol{A} = \boldsymbol{B} = \begin{bmatrix} 0.99 & 0.0010 \\ 0.0010 & 0.99 \end{bmatrix},$$

and suppose we are computing on a machine using 2-digit floating point arithmetic. Taking g to be Strassen's algorithm computed on this machine, we get $\|g(\boldsymbol{A}, \boldsymbol{B}) - \boldsymbol{AB}\| \approx 0.0286$. In the definition of $\hat{g}$, there are a total of 64 possible sign functions and 8 possible permutation functions. Each combination of these has the same probability of occurring, so we can readily compute $\mathbb{E}[\hat{g}(\boldsymbol{A}, \boldsymbol{B})]$. Doing this, we find that $\|\mathbb{E}[\hat{g}(\boldsymbol{A}, \boldsymbol{B})] - \boldsymbol{AB}\| \approx 0.0024$. So we cannot guarantee $\mathbb{E}[\hat{g}(\boldsymbol{A}, \boldsymbol{B})] = \boldsymbol{AB}$ in this finite precision setting. We are not even guaranteed to have $\|\mathbb{E}[\hat{g}(\boldsymbol{A}, \boldsymbol{B})] - \boldsymbol{AB}\| \leq \|g(\boldsymbol{A}, \boldsymbol{B}) - \boldsymbol{AB}\|$: Let $\boldsymbol{A}^{(r)}$ and $\boldsymbol{B}^{(r)}$ be the same as $\boldsymbol{A}$ and $\boldsymbol{B}$, respectively, but with the order of the columns reversed, i.e., $\boldsymbol{A}^{(r)} \stackrel{\text{def}}{=} \begin{bmatrix} \boldsymbol{a}_{:2} & \boldsymbol{a}_{:1} \end{bmatrix}$ and $\boldsymbol{B}^{(r)} \stackrel{\text{def}}{=} \begin{bmatrix} \boldsymbol{b}_{:2} & \boldsymbol{b}_{:1} \end{bmatrix}$. We then get $\|g(\boldsymbol{A}^{(r)}, \boldsymbol{B}^{(r)}) - \boldsymbol{A}^{(r)}\boldsymbol{B}^{(r)}\| \approx 0.0001$, but $\|\mathbb{E}[\hat{g}(\boldsymbol{A}^{(r)}, \boldsymbol{B}^{(r)})] - \boldsymbol{A}^{(r)}\boldsymbol{B}^{(r)}\| \approx 0.0024$. Despite this, randomization seems to work remarkably well in practice when an exact algorithm is computed in finite precision arithmetic, as we will see in the numerical experiments.

We now consider ABCs (which include EBCs as a special case) in finite precision arithmetic. As in [83], we use $\text{fl}(x)$ to denote the representation of $x \in \mathbb{R}$ as a floating point number, and $\text{fl}(f(x))$ to denote the result of computing $f(x)$ in floating point arithmetic. When computing the latter, the algorithm used to compute f matters. We make the standard assumption that $\text{fl}(x \operatorname{op} y) = (x \operatorname{op} y)(1 + \Delta)$ where x and y are floating point numbers, op is scalar addition, subtraction or multiplication, and $|\Delta| \leq \varepsilon_{\text{machine}}$, where $\varepsilon_{\text{machine}}$ is the machine epsilon or unit roundoff. For numerical summations, e.g. $\text{fl}(\sum_{n=1}^{N} x_n)$, we assume that the summation is simply

done sequentially, e.g.

$$\mathrm{fl}\Big(\sum_{n=1}^{3} x_n\Big) = \mathrm{fl}\Big(\mathrm{fl}\big(\mathrm{fl}(x_1) + \mathrm{fl}(x_2)\big) + \mathrm{fl}(x_3)\Big).$$

We first present a series of results in Propositions 43–47 with the goal of understanding how algorithmic **and** numerical error combined impact the deterministic and randomized ABCs described by f and $\hat{f}$, respectively. Throughout these results, we make the reasonable assumption that the input matrices $\boldsymbol{A}$ and $\boldsymbol{B}$, as well as the tensors $\boldsymbol{\mathcal{U}}$, $\boldsymbol{\mathcal{V}}$ and $\boldsymbol{\mathcal{W}}$ defining the BC, are already stored in floating point format, so that e.g. $\mathrm{fl}(\boldsymbol{A}) = \boldsymbol{A}$. Proposition 43 provides an upper bound on the numerical error for the (approximate or exact) BC f defined as in (5.1).

Proposition 43. *Suppose* $(4n + R - 1)\varepsilon_{\text{machine}} \leq 0.01$. *For* $\boldsymbol{A}, \boldsymbol{B} \in B_{\mu}^{(n)}$ *and an* (n, τ)*-ABC* f *computed according to* (5.5)*, we have*

$$\|\mathrm{fl}(f(\boldsymbol{A}, \boldsymbol{B})) - f(\boldsymbol{A}, \boldsymbol{B})\| \leq 1.01(4n + R - 1)\sqrt{R}\varepsilon_{\text{machine}}\mu^2 \sqrt{\sum_{r=1}^{R} \|\boldsymbol{U}_{::r}\|^2 \|\boldsymbol{V}_{::r}\|^2 \|\boldsymbol{W}_{::r}\|^2},$$

where e.g. $\boldsymbol{U}_{::r} \in \mathbb{R}^{n \times n}$ *is the* r*th frontal slice of* $\boldsymbol{\mathcal{U}}$*, so that* $\|\boldsymbol{U}_{::r}\|^2 = \sum_{k,l} u_{klr}^2$.

Proof. Throughout this proof, constants γ, θ, λ, $\tilde{\gamma}$, $\tilde{\theta}$, $\tilde{\lambda}$, ϕ and ψ with subscripts are real numbers of magnitude less than or equal to $\varepsilon_{\text{machine}}$. Define

$$s_{kpr} \stackrel{\text{def}}{=} \mathrm{fl}\Big(\sum_{l=1}^{p} u_{klr} a_{kl}\Big).$$

Then

$$s_{k1r} = \mathrm{fl}(u_{k1r} a_{k1}) = u_{k1r} a_{k1}(1 + \gamma_{k1r}),$$

$$s_{k2r} = \mathrm{fl}(s_{k1r} + u_{k2r} a_{k2}) = (s_{k1r} + u_{k2r} a_{k2}(1 + \gamma_{k2r}))(1 + \theta_{k2r}),$$

etc. More generally,

$$s_{kpr} = \sum_{l=1}^{p} u_{klr} a_{kl}(1 + \gamma_{klr}) \prod_{\alpha=l}^{p} (1 + \theta_{k\alpha r}), \quad \theta_{k1r} \stackrel{\text{def}}{=} 0.$$

Next, define

$$\hat{s}_{pr} \stackrel{\text{def}}{=} \mathrm{fl}\Big(\sum_{k=1}^{p} \sum_{l=1}^{n} u_{klr} a_{kl}\Big).$$

Then

$$\hat{s}_{1r} = \mathrm{fl}\Big(\sum_{l=1}^{n} u_{1lr}a_{1l}\Big) = s_{1nr},$$

$$\hat{s}_{2r} = \mathrm{fl}\Big(\hat{s}_{1r} + \mathrm{fl}\Big(\sum_{l=1}^{n} u_{2lr}a_{2l}\Big)\Big) = (s_{1nr} + s_{2nr})(1+\lambda_{2r}),$$

etc. More generally,

$$\hat{s}_{pr} = \sum_{k=1}^{p} s_{knr} \prod_{\alpha=k}^{p}(1+\lambda_{\alpha r}), \quad \lambda_{1r} \stackrel{\text{def}}{=} 0.$$

Consequently,

$$P_r \stackrel{\text{def}}{=} \mathrm{fl}\Big(\sum_{k=1}^{n}\sum_{l=1}^{n} u_{klr}a_{kl}\Big) = \hat{s}_{nr} = \sum_{k=1}^{n}\sum_{l=1}^{n} u_{klr}a_{kl}(1+\gamma_{klr})\Big(\prod_{\alpha=l}^{n}(1+\theta_{k\alpha r})\Big)\Big(\prod_{\alpha=k}^{n}(1+\lambda_{\alpha r})\Big).$$

Similarly,

$$Q_r \stackrel{\text{def}}{=} \mathrm{fl}\Big(\sum_{k'=1}^{n}\sum_{l'=1}^{n} v_{k'l'r}b_{k'l'}\Big) = \sum_{k'=1}^{n}\sum_{l'=1}^{n} v_{k'l'r}b_{k'l'}(1+\tilde{\gamma}_{k'l'r})\Big(\prod_{\alpha=l'}^{n}(1+\tilde{\theta}_{k'\alpha r})\Big)\Big(\prod_{\alpha=k'}^{n}(1+\tilde{\lambda}_{\alpha r})\Big).$$

Now, define

$$z_p \stackrel{\text{def}}{=} \mathrm{fl}\Big(\sum_{r=1}^{p} w_{ijr}\Big(\sum_{k,l=1}^{n} u_{klr}a_{kl}\Big)\Big(\sum_{k',l'=1}^{n} v_{k'l'r}b_{k'l'}\Big)\Big).$$

We have

$$\begin{aligned} z_1 &= \mathrm{fl}\Big(w_{ij1}\Big(\sum_{k,l=1}^{n} u_{kl1}a_{kl}\Big)\Big(\sum_{k',l'=1}^{n} v_{k'l'1}b_{k'l'}\Big)\Big) \\ &= w_{ij1}P_1Q_1(1+\phi_{11})(1+\phi_{12}), \\ z_2 &= \mathrm{fl}\Big(z_1 + \mathrm{fl}\Big(w_{ij2}\Big(\sum_{k,l=1}^{n} u_{kl2}a_{kl}\Big)\Big(\sum_{k',l'=1}^{n} v_{k'l'2}b_{k'l'}\Big)\Big)\Big) \\ &= (z_1 + w_{ij2}P_2Q_2(1+\phi_{21})(1+\phi_{22}))(1+\psi_2), \end{aligned}$$

etc. From this, it follows that

$$\mathrm{fl}(f(\boldsymbol{A},\boldsymbol{B})_{ij}) = z_R = \sum_{r=1}^{R} w_{ijr}P_rQ_r(1+\phi_{r1})(1+\phi_{r2})\prod_{\zeta=r}^{R}(1+\psi_\zeta), \quad \psi_1 \stackrel{\text{def}}{=} 0.$$

Writing this out in full and rearranging, we have

$$\begin{aligned} \mathrm{fl}(f(\boldsymbol{A},\boldsymbol{B})_{ij}) = \sum_{k,l=1}^{n}\sum_{k',l'=1}^{n} a_{kl}b_{k'l'}\sum_{r=1}^{R} u_{klr}v_{k'l'r}w_{ijr}(1+\gamma_{klr})(1+\tilde{\gamma}_{k'l'r}) \\ \times\Big(\prod_{\alpha=l}^{n}(1+\theta_{k\alpha r})\Big)\Big(\prod_{\alpha=l'}^{n}(1+\tilde{\theta}_{k'\alpha r})\Big)\Big(\prod_{\alpha=k}^{n}(1+\lambda_{\alpha r})\Big) \\ \times\Big(\prod_{\alpha=k'}^{n}(1+\tilde{\lambda}_{\alpha r})\Big)(1+\phi_{r1})(1+\phi_{r2})\prod_{\zeta=r}^{R}(1+\psi_\zeta). \end{aligned}$$

Since we have assumed $(4n + R - 1)\varepsilon_{\text{machine}} \leq 0.01$, it follows from Lemma 2.7.1 in [83] that there exist constants $\varepsilon_{klk'l'r}$, for $(k, l, k', l', r) \in [n]^4 \times [R]$, such that $|\varepsilon_{klk'l'r}| \leq 1.01(4n + R - 1)\varepsilon_{\text{machine}}$ and

$$\begin{aligned}\text{fl}(f(\boldsymbol{A}, \boldsymbol{B})_{ij}) &= \sum_{k,l=1}^{n} \sum_{k',l'=1}^{n} a_{kl} b_{k'l'} \sum_{r=1}^{R} u_{klr} v_{k'l'r} w_{ijr} (1 + \varepsilon_{klk'l'r}) \\ &= f(\boldsymbol{A}, \boldsymbol{B})_{ij} + \sum_{k,l=1}^{n} \sum_{k',l'=1}^{n} a_{kl} b_{k'l'} \sum_{r=1}^{R} u_{klr} v_{k'l'r} w_{ijr} \varepsilon_{klk'l'r}.\end{aligned}$$

We have

$$\begin{aligned}|\text{fl}(f(\boldsymbol{A}, \boldsymbol{B})_{ij}) - f(\boldsymbol{A}, \boldsymbol{B})_{ij}|^2 &= \Big| \sum_{k,l=1}^{n} \sum_{k',l'=1}^{n} a_{kl} b_{k'l'} \sum_{r=1}^{R} u_{klr} v_{k'l'r} w_{ijr} \varepsilon_{klk'l'r} \Big|^2 \\ &\leq \Big(\sum_{k,l=1}^{n} \sum_{k',l'=1}^{n} (a_{kl} b_{k'l'})^2 \Big) \Big(\sum_{k,l=1}^{n} \sum_{k',l'=1}^{n} \Big(\sum_{r=1}^{R} u_{klr} v_{k'l'r} w_{ijr} \varepsilon_{klk'l'r} \Big)^2 \Big) \\ &\leq \Big(\sum_{k,l=1}^{n} \sum_{k',l'=1}^{n} (a_{kl} b_{k'l'})^2 \Big) \Big(\sum_{k,l=1}^{n} \sum_{k',l'=1}^{n} \Big(\sum_{r=1}^{R} (u_{klr} v_{k'l'r} w_{ijr})^2 \Big) \Big(\sum_{r=1}^{R} \varepsilon_{klk'l'r}^2 \Big) \Big) \\ &\leq \|\boldsymbol{A}\|^2 \|\boldsymbol{B}\|^2 R (1.01(4n + R - 1)\varepsilon_{\text{machine}})^2 \sum_{r=1}^{R} \sum_{k,l=1}^{n} \sum_{k',l'=1}^{n} (u_{klr} v_{k'l'r} w_{ijr})^2,\end{aligned}$$

where the first and second inequality follows from the Cauchy–Schwarz inequality, and the third inequality follows from the bound on $|\varepsilon_{klk'l'r}|$. Summing over all i, j, taking a square root, and using that $\|\boldsymbol{A}\|, \|\boldsymbol{B}\| \leq \mu$, we arrive at

$$\|\text{fl}(f(\boldsymbol{A}, \boldsymbol{B})) - f(\boldsymbol{A}, \boldsymbol{B})\| \leq 1.01(4n + R - 1)\sqrt{R}\varepsilon_{\text{machine}}\mu^2 \sqrt{\sum_{r=1}^{R} \|\boldsymbol{U}_{::r}\|^2 \|\boldsymbol{V}_{::r}\|^2 \|\boldsymbol{W}_{::r}\|^2}.$$

□

Proposition 44 generalizes Proposition 43 to the case when the input matrices are of size $mn \times mn$.

Proposition 44. *Suppose* $(4n + m + R - 2)\varepsilon_{\text{machine}} \leq 1.01$. *For* $\boldsymbol{A}, \boldsymbol{B} \in B_{\mu}^{(mn)}$ *and an* (n, τ)*-ABC* f *computed according to* (5.5)*, we have*

$$\|\text{fl}(f(\boldsymbol{A}, \boldsymbol{B})) - f(\boldsymbol{A}, \boldsymbol{B})\| \leq 1.01(4n + m + R - 2)\sqrt{R}\varepsilon_{\text{machine}}\mu^2 \sqrt{\sum_{r=1}^{R} \|\boldsymbol{U}_{::r}\|^2 \|\boldsymbol{V}_{::r}\|^2 \|\boldsymbol{W}_{::r}\|^2}.$$

Proof. Note that all computations when calculating $\sum_{k,l} u_{klr}\boldsymbol{A}_{kl}$ and $\sum_{k',l'} v_{k'l'r}\boldsymbol{B}_{k'l'}$ are done elementwise. The error accumulated for each entry will therefore be the same as in the case when $\boldsymbol{A}_{kl}$ and $\boldsymbol{B}_{k'l'}$ are scalars. More precisely,

$$\mathrm{fl}\Big(\sum_{k,l=1}^{n} u_{klr}\boldsymbol{A}_{kl}\Big) = \sum_{k,l=1}^{n} u_{klr}\boldsymbol{A}_{kl} \circledast \boldsymbol{E}_{klr}, \tag{5.17}$$

where $\circledast$ is the Hadamard (elementwise) product, and $\boldsymbol{E}_{klr} \in \mathbb{R}^{m\times m}$ is a matrix with entries which are the product of at most $2n-1$ terms of the form $(1+\varepsilon)$ with $|\varepsilon| \leq \varepsilon_{\text{machine}}$. A similar statement is true for $\mathrm{fl}(\sum_{k',l'=1}^{n} v_{k'l'r}\boldsymbol{B}_{k'l'})$. When computing each

$$\mathrm{fl}\Big(w_{ijr}\Big(\sum_{k,l=1}^{n} u_{klr}\boldsymbol{A}_{kl}\Big)\Big(\sum_{k',l'=1}^{n} v_{k'l'r}\boldsymbol{B}_{k'l'}\Big)\Big),$$

we now have additional error compared to the scalar case due to the inner product computations. In order to avoid cumbersome notation, we do not write out the following computations in full. Using the result in Section 2.7.6 of [83] for the rounding error in inner products, Equation (5.17), Lemma 2.7.1 in [83], and letting $(f(\boldsymbol{A},\boldsymbol{B})_{ij})_{\alpha\beta}$ denote the element on position (α,β) in matrix block (i,j), we can compute

$$\mathrm{fl}((f(\boldsymbol{A},\boldsymbol{B})_{ij})_{\alpha\beta}) = (f(\boldsymbol{A},\boldsymbol{B})_{ij})_{\alpha\beta} + \sum_{r=1}^{R} w_{ijr} \sum_{k,l=1}^{n}\sum_{k',l'=1}^{n}\sum_{z=1}^{m} u_{klr}v_{k'l'r}(\boldsymbol{A}_{kl})_{\alpha z}(\boldsymbol{B}_{k'l'})_{z\beta}\Theta_{klk'l'r\alpha\beta z},$$

where each $|\Theta_{klk'l'r\alpha\beta z}| \leq \hat{\theta} \overset{\text{def}}{=} 1.01(4n+m+R-2)\varepsilon_{\text{machine}}$. Rearranging this, we have

$$\begin{aligned}
&|\mathrm{fl}((f(\boldsymbol{A},\boldsymbol{B})_{ij})_{\alpha\beta}) - (f(\boldsymbol{A},\boldsymbol{B})_{ij})_{\alpha\beta}|^2 \\
&= \Big|\sum_{r=1}^{R}\sum_{k,l=1}^{n}\sum_{k',l'=1}^{n}\sum_{z=1}^{m} u_{klr}v_{k'l'r}w_{ijr}(\boldsymbol{A}_{kl})_{\alpha z}(\boldsymbol{B}_{k'l'})_{z\beta}\Theta_{klk'l'r\alpha\beta z}\Big|^2 \\
&\leq \hat{\theta}^2\Big|\sum_{r=1}^{R}\sum_{k,l=1}^{n}\sum_{k',l'=1}^{n} |u_{klr}v_{k'l'r}w_{ijr}| \sum_{z=1}^{m} |(\boldsymbol{A}_{kl})_{\alpha z}(\boldsymbol{B}_{k'l'})_{z\beta}|\Big|^2
\end{aligned}$$

due to the triangle inequality and definition of $\hat{\theta}$,

$$\begin{aligned}
&\leq R\hat{\theta}^2 \sum_{r=1}^{R} \Big| \sum_{k,l=1}^{n} \sum_{k',l'=1}^{n} |u_{klr} v_{k'l'r} w_{ijr}| \sum_{z=1}^{m} |(\boldsymbol{A}_{kl})_{\alpha z} (\boldsymbol{B}_{k'l'})_{z\beta}| \Big|^2 \\
&\leq R\hat{\theta}^2 \sum_{r=1}^{R} \Big(\sum_{k,l=1}^{n} \sum_{k',l'=1}^{n} |u_{klr} v_{k'l'r} w_{ijr}|^2 \Big) \Big(\sum_{k,l=1}^{n} \sum_{k',l'=1}^{n} \Big| \sum_{z=1}^{m} |(\boldsymbol{A}_{kl})_{\alpha z} (\boldsymbol{B}_{k'l'})_{z\beta}| \Big|^2 \Big) \\
&\leq R\hat{\theta}^2 \sum_{r=1}^{R} \Big(\sum_{k,l=1}^{n} \sum_{k',l'=1}^{n} |u_{klr} v_{k'l'r} w_{ijr}|^2 \Big) \Big(\sum_{k,l=1}^{n} \sum_{k',l'=1}^{n} \Big(\sum_{z=1}^{m} |(\boldsymbol{A}_{kl})_{\alpha z}|^2 \Big) \Big(\sum_{z=1}^{m} |(\boldsymbol{B}_{k'l'})_{z\beta}|^2 \Big) \Big) \\
&= R(1.01(4n + m + R - 2)\varepsilon_{\text{machine}})^2 \\
&\quad \times \Big(\sum_{k,l=1}^{n} \sum_{z=1}^{m} (\boldsymbol{A}_{kl})_{\alpha z}^2 \Big) \Big(\sum_{k',l'=1}^{n} \sum_{z=1}^{m} (\boldsymbol{B}_{k'l'})_{z\beta}^2 \Big) \Big(\sum_{r=1}^{R} \sum_{k,l=1}^{n} \sum_{k',l'=1}^{n} u_{klr}^2 v_{k'l'r}^2 w_{ijr}^2 \Big)
\end{aligned}$$

where each inequality follows from an application of the Cauchy–Schwarz inequality and the final equality follows from a rearrangement of terms. Finally, since

$$\|\text{fl}(f(\boldsymbol{A}, \boldsymbol{B})) - f(\boldsymbol{A}, \boldsymbol{B})\|^2 = \sum_{\alpha,\beta=1}^{m} \sum_{i,j=1}^{n} |\text{fl}((f(\boldsymbol{A}, \boldsymbol{B})_{ij})_{\alpha\beta}) - (f(\boldsymbol{A}, \boldsymbol{B})_{ij})_{\alpha\beta}|^2$$

and $\|\boldsymbol{A}\|, \|\boldsymbol{B}\| \leq \mu$, it follows that

$$\|\text{fl}(f(\boldsymbol{A}, \boldsymbol{B})) - f(\boldsymbol{A}, \boldsymbol{B})\| \leq 1.01(4n + m + R - 2)\sqrt{R}\varepsilon_{\text{machine}}\mu^2 \sqrt{\sum_{r=1}^{R} \|\boldsymbol{U}_{::r}\|^2 \|\boldsymbol{V}_{::r}\|^2 \|\boldsymbol{W}_{::r}\|^2}.$$

□

Proposition 45 shows how the error in Proposition 44 changes when a randomized algorithm $\hat{f}$ is used instead of its deterministic counterpart. In particular, the bound is worse by only a factor $(1 - \kappa)^{-1}$.

Proposition 45. *Suppose* $(4n + m + R - 2)\varepsilon_{\text{machine}} \leq 1.01$ *and* $1 - \kappa > 0$. *For* $\boldsymbol{A}, \boldsymbol{B} \in B_{\mu}^{(mn)}$ *and an* (n, τ, κ)*-RandABC* $\hat{f}$ *computed according to* (5.8)*, we have*

$$\begin{aligned}
&\|\text{fl}(\hat{f}(\boldsymbol{A}, \boldsymbol{B})) - \hat{f}(\boldsymbol{A}, \boldsymbol{B})\| \\
&\leq 1.01(4n + m + R - 2)\sqrt{R}\varepsilon_{\text{machine}}(1 - \kappa)^{-1}\mu^2 \sqrt{\sum_{r=1}^{R} \|\boldsymbol{U}_{::r}\|^2 \|\boldsymbol{V}_{::r}\|^2 \|\boldsymbol{W}_{::r}\|^2}.
\end{aligned}$$

Proof. From (5.10), it follows that

$$\begin{aligned}\hat{f}(\boldsymbol{A},\boldsymbol{B})_{ij} &= (1-\kappa)^{-1}s_1(i)s_3(j)\sum_{r=1}^{R} w_{\pi_1(i)\pi_3(j)r} \\ &\times \Big(\sum_{k,l=1}^{n} u_{\pi_1(k)\pi_2(l)r}s_1(k)s_2(l)\boldsymbol{A}_{kl}\Big)\Big(\sum_{k',l'=1}^{n} v_{\pi_2(k')\pi_3(l')r}s_2(k')s_3(l')\boldsymbol{B}_{k'l'}\Big) \\ &= \sum_{r=1}^{R}\hat{w}_{ijr}\Big(\sum_{k,l=1}^{n}\hat{u}_{klr}\boldsymbol{A}_{kl}\Big)\Big(\sum_{k',l'=1}^{n}\hat{v}_{k'l'r}\boldsymbol{B}_{k'l'}\Big),\end{aligned} \tag{5.18}$$

where

$$\begin{aligned}\hat{u}_{klr} &\stackrel{\text{def}}{=} u_{\pi_1(k)\pi_2(l)r}s_1(k)s_2(l), \\ \hat{v}_{k'l'r} &\stackrel{\text{def}}{=} v_{\pi_2(k')\pi_3(l')r}s_2(k')s_3(l'), \\ \hat{w}_{ijr} &\stackrel{\text{def}}{=} (1-\kappa)^{-1}s_1(i)s_3(j)w_{\pi_1(i)\pi_3(j)r}.\end{aligned}$$

Note that $\|\hat{\boldsymbol{U}}_{::r}\| = \|\boldsymbol{U}_{::r}\|$, $\|\hat{\boldsymbol{V}}_{::r}\| = \|\boldsymbol{V}_{::r}\|$ and $\|\hat{\boldsymbol{W}}_{::r}\| = (1-\kappa)^{-1}\|\boldsymbol{W}_{::r}\|$. Using this fact, and applying Proposition 44 to the computation in (5.18), gives us the desired result. □

Propositions 46 and 47 give upper bounds for the total error, both due to algorithmic error and numerical rounding, for the deterministic and randomized algorithms, respectively.

Proposition 46. *Suppose* $(4n+m+R-2)\varepsilon_{\text{machine}} \leq 1.01$. *For* $\boldsymbol{A},\boldsymbol{B} \in B_\mu^{(mn)}$ *and an* (n,τ)*-ABC* f *computed according to* (5.5)*, we have*

$$\begin{aligned}&\|\text{fl}(f(\boldsymbol{A},\boldsymbol{B})) - \boldsymbol{A}\boldsymbol{B}\| \\ &\leq 1.01(4n+m+R-2)\sqrt{R}\varepsilon_{\text{machine}}\mu^2\sqrt{\sum_{r=1}^{R}\|\boldsymbol{U}_{::r}\|^2\|\boldsymbol{V}_{::r}\|^2\|\boldsymbol{W}_{::r}\|^2} + \mu^2\tau.\end{aligned} \tag{5.19}$$

Proof. Follows by applying the triangle inequality and using Propositions 44 and 39 (i). □

Proposition 47. *Suppose* $(4n+m+R-2)\varepsilon_{\text{machine}} \leq 1.01$. *For* $\boldsymbol{A},\boldsymbol{B} \in B_\mu^{(mn)}$ *and an* (n,τ,κ)*-RandABC* $\hat{f}$ *computed according to* (5.8)*, we have*

$$\begin{aligned}&\|\mathbb{E}[\text{fl}(\hat{f}(\boldsymbol{A},\boldsymbol{B}))] - \boldsymbol{A}\boldsymbol{B}\| \\ &\leq 1.01(4n+m+R-2)\sqrt{R}\varepsilon_{\text{machine}}(1-\kappa)^{-1}\mu^2\sqrt{\sum_{r=1}^{R}\|\boldsymbol{U}_{::r}\|^2\|\boldsymbol{V}_{::r}\|^2\|\boldsymbol{W}_{::r}\|^2}.\end{aligned} \tag{5.20}$$

Proof. Follows by taking expectations of the inequality in Proposition 45 and applying Jensen's inequality [183]. □

Note that Proposition 47 implies that

$$\mathbb{E}[\text{fl}(\hat{f}(\boldsymbol{A}, \boldsymbol{B}))] \to \boldsymbol{AB} \quad \text{as} \quad \varepsilon_{\text{machine}} \to 0,$$

which means that as the numerical precision increases, $\mathbb{E}[\text{fl}(\hat{f}(\boldsymbol{A}, \boldsymbol{B}))]$ approaches $\boldsymbol{AB}$. This is consistent with Proposition 38.

To make the bounds in (5.19) and (5.20) easier to compare, using the fact that $(1-\kappa)^{-1} = 1 + \kappa + O(\kappa^2)$, we can rewrite (5.20) as

$$\begin{aligned}
&\|\mathbb{E}[\text{fl}(\hat{f}(\boldsymbol{A}, \boldsymbol{B}))] - \boldsymbol{AB}\| \\
&\leq 1.01(4n + m + R - 2)\sqrt{R}\varepsilon_{\text{machine}}\mu^2 \sqrt{\sum_{r=1}^{R} \|\boldsymbol{U}_{::r}\|^2 \|\boldsymbol{V}_{::r}\|^2 \|\boldsymbol{W}_{::r}\|^2} \\
&+ 1.01(4n + m + R - 2)\sqrt{R}\varepsilon_{\text{machine}}\kappa\mu^2 \sqrt{\sum_{r=1}^{R} \|\boldsymbol{U}_{::r}\|^2 \|\boldsymbol{V}_{::r}\|^2 \|\boldsymbol{W}_{::r}\|^2} + O(\varepsilon_{\text{machine}}\kappa^2).
\end{aligned} \tag{5.21}$$

The first term on the right hand side of (5.19) and (5.21) are identical. Recall from (5.14) that $|\kappa| \leq n^{-5/2}\|\boldsymbol{\mathcal{Y}} - \boldsymbol{\mathcal{X}}\| \leq n^{-5/2}\tau$. Consequently, if the quantity

$$\sqrt{\sum_{r=1}^{R} \|\boldsymbol{U}_{::r}\|^2 \|\boldsymbol{V}_{::r}\|^2 \|\boldsymbol{W}_{::r}\|^2} \tag{5.22}$$

is not too large, and m, n and R are of moderate size, the second term in (5.21) will be smaller than the second term in (5.19), showing that the result in Proposition 38 largely carries over to a setting with floating point arithmetic. The following example illustrates this.

Example 48. For the sake of this example, suppose we are using an approximate variant of Strassen's algorithm. Then $R = 7$ and $n = 2$. For Strassen's algorithm, the quantity in (5.22) is approximately 6, so we will assume that it is less than 10 for our approximate variant of the algorithm. Suppose we are multiplying two 100,000 × 100,000 matrices in single precision, so that $m = 50{,}000$ and $\varepsilon_{\text{machine}} \sim 10^{-8}$. Then, the second term in (5.21) is upper bounded by a quantity

which is on the order of

$$10 \cdot 1.01 \cdot (4 \cdot 2 + 50000 + 7 - 2) \cdot \frac{\sqrt{7}}{2^{5/2}} \cdot 10^{-8} \mu^2 \tau \approx 0.0024 \mu^2 \tau,$$

which is much smaller than the second term in (5.19). The implementation of Strassen's algorithm in [99] achieves a speed-up over an efficient implementation of the standard matrix multiplication algorithm for square matrices with as few as 1,536 rows/columns. So multiplication of matrices with 100,000 rows/columns is well beyond the problem size for which fast algorithms can outperform the standard algorithm.

5.3 Experiments

In this section we present some results from experiments, with additional results provided in Section A.1. We implement all experiments in Matlab with certain parts implemented in C. All our code is available online at

`https://github.com/OsmanMalik/random-approximate-matrix-multiplication`.

In our experiments, we draw the matrices $\boldsymbol{A}, \boldsymbol{B}$ from different random distributions. By **Gaussian matrix**, we mean a matrix whose elements are realizations of i.i.d. standard normal random variables. Similarly, a **uniform matrix** is one whose elements are realizations of i.i.d. Uniform$(0, 1)$ random variables. We also consider three types of random **adversarial matrices**, which were proposed by [12] and are designed to be challenging for Strassen's algorithm.

Definition 49 (Adversarial matrices)**.** Consider a matrix pair $\boldsymbol{A}, \boldsymbol{B} \in \mathbb{R}^{n \times n}$. We say that it is **type 1 adversarial** if

$$a_{ij} \sim \begin{cases} \text{Uniform}(0, 1/n^2) & \text{if } j > n/2, \\ \text{Uniform}(0, 1) & \text{otherwise,} \end{cases} \qquad b_{ij} \sim \begin{cases} \text{Uniform}(0, 1/n^2) & \text{if } i < n/2, \\ \text{Uniform}(0, 1) & \text{otherwise.} \end{cases}$$

We say that the matrix pair is **type 2 adversarial** if

$$a_{ij} \sim \begin{cases} \text{Uniform}(0, n^2) & \text{if } i < n/2 \text{ and } j > n/2, \\ \text{Uniform}(0, 1) & \text{otherwise,} \end{cases} \qquad b_{ij} \sim \begin{cases} \text{Uniform}(0, 1/n^2) & \text{if } j < n/2, \\ \text{Uniform}(0, 1) & \text{otherwise.} \end{cases}$$

We say that the matrix pair is **type 3 adversarial** if

$$a_{ij}, b_{ij} \sim \begin{cases} \text{Uniform}(0, 1/n^2) & \text{if } i < n/2 \text{ and } j > n/2, \text{ or if } i \geq n/2 \text{ and } j \leq n/2 \\ \text{Uniform}(0, 1) & \text{otherwise,} \end{cases}$$

Here, all the entries are assumed to be independent.

We will also consider the Hilbert matrix $\boldsymbol{H} \in \mathbb{R}^{n \times n}$, which has entries $h_{ij} = 1/(i + j - 1)$, since it appears to be a particularly challenging matrix to the rescaling method which we consider in Section 5.3.2.

For an ABC, the corresponding randomized computation with Q recursions, $\hat{F}^{(Q)}$, was defined in (5.16). In this section, we will use $F^{(Q)}$ to denote the deterministic counterpart. It is defined in the same way, but with each $s_i^{(q)}(j) = 1$ and each $\pi_i^{(q)}(j) = j$, i.e., with no randomness involved. As in Example 42, for EBCs we will use $\hat{G}^{(Q)}$ and $G^{(Q)}$ to denote the corresponding randomized and deterministic computations with Q recursions.

5.3.1 Approximate algorithm

We create an ABC of the form (5.5) by taking the tensors $\boldsymbol{\mathcal{U}}$, $\boldsymbol{\mathcal{V}}$ and $\boldsymbol{\mathcal{W}}$ corresponding to Strassen's algorithm and perturbing them: For each of the three tensors, we add i.i.d. mean zero Gaussian noise with standard deviation 10^{-3} to each element in the tensor equal to 1 as well as to five randomly selected elements that are equal to 0. Since we do the computations in double precision, and since we add a considerable amount of noise, these experiments are designed to test Propositions 38–41, i.e., we can ignore any floating point error.

In the first experiment, we draw two Gaussian matrices $\boldsymbol{A}, \boldsymbol{B} \in \mathbb{R}^{80 \times 80}$ and compute

$$\Big\| \frac{1}{n} \sum_{i=1}^{n} \hat{F}_i^{(Q)}(\boldsymbol{A}, \boldsymbol{B}) - \boldsymbol{AB} \Big\| / \|\boldsymbol{AB}\| \tag{5.23}$$

for $n \in [10^4]$ and $Q \in [3]$. Here, $\hat{F}_i^{(Q)}$ is the ith realization of $\hat{F}^{(Q)}$. Figure 5.1 shows the results. As expected from Propositions 38 and 41, the quantity in (5.23) becomes smaller as n increases.

In the second experiment, we draw two Gaussian matrices $\boldsymbol{A}, \boldsymbol{B} \in \mathbb{R}^{320\times 320}$ and compute

$$\|\hat{F}_i^{(Q)}(\boldsymbol{A}, \boldsymbol{B}) - \boldsymbol{AB}\|/\|\boldsymbol{AB}\| \tag{5.24}$$

for $i \in [100]$ and $Q \in [5]$, i.e., the relative error for each of 100 trials. The box plots in Figure 5.2 compares the empirical distribution of (5.24) to the relative error for the deterministic approximate algorithm, i.e., $\|F^{(Q)}(\boldsymbol{A}, \boldsymbol{B}) - \boldsymbol{AB}\|/\|\boldsymbol{AB}\|$. In this particular case, randomization does not impact the median error and there is very little variation between trials. Figure 5.3 repeats this experiment, but with $\boldsymbol{A} = \boldsymbol{B} = \boldsymbol{H}$, where $\boldsymbol{H}$ is the 320×320 Hilbert matrix. In this case, the randomized scheme frequently results in a slightly larger error. In Figure A.1–A.5 in Section A.1 we provide additional results for when $\boldsymbol{A}, \boldsymbol{B}$ are Gaussian, uniform and type 1–3 adversarial. Those results demonstrate that randomization can both increase and decrease the error in this setting. However, as expected from Propositions 39 and 40, the difference in error between the randomized and deterministic variants is typically not substantial.

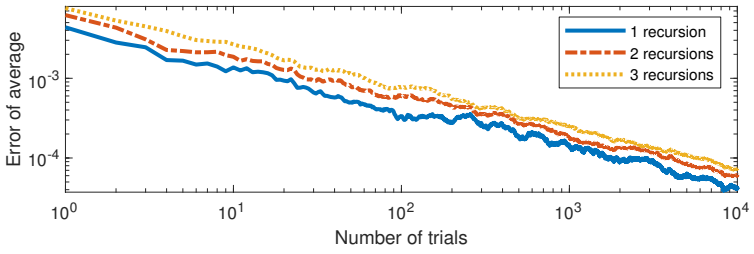

Figure 5.1: Error of average for randomized ABC. $\boldsymbol{A}, \boldsymbol{B} \in \mathbb{R}^{80\times 80}$ are Gaussian and remain fixed throughout the experiment. The ABC is a perturbed variant of Strassen's algorithm.

To see how these results carry over to other ABCs, we also present results in Figure 5.4 from experiments which use an instance of the 12×12 APA algorithm in [26] with $\varepsilon = 1\text{e}{-4}$ in the representation (5.6). Results are shown for single recursion experiments on 12×12 matrices drawn from different distributions. As for the ABC based on the perturbed variant of Strassen's algorithm, the randomized variant of this ABC sometimes results in a smaller and sometimes in a larger error than the deterministic counterpart.

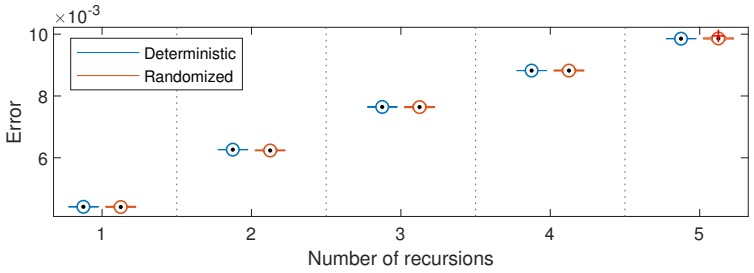

Figure 5.2: Error for deterministic ABC compared to the error of the randomized counterpart, over 100 realizations of the randomized algorithm. $\boldsymbol{A}, \boldsymbol{B} \in \mathbb{R}^{320\times 320}$ are Gaussian and remain fixed over all realizations. Note that, unlike Figure 5.1, this figure shows the distribution of errors, rather than the errors of averages. The ABC is a perturbed variant of Strassen's algorithm.

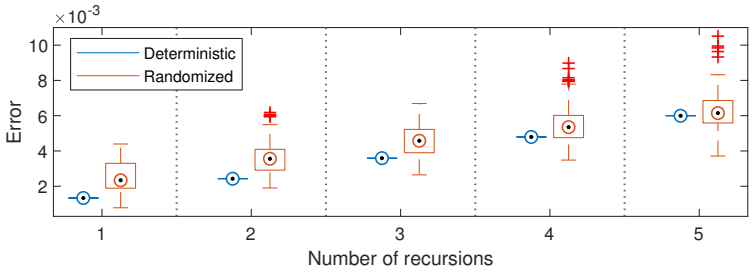

Figure 5.3: Same as Figure 5.2, but with $\boldsymbol{A}$ and $\boldsymbol{B}$ equal to the 320×320 Hilbert matrix.

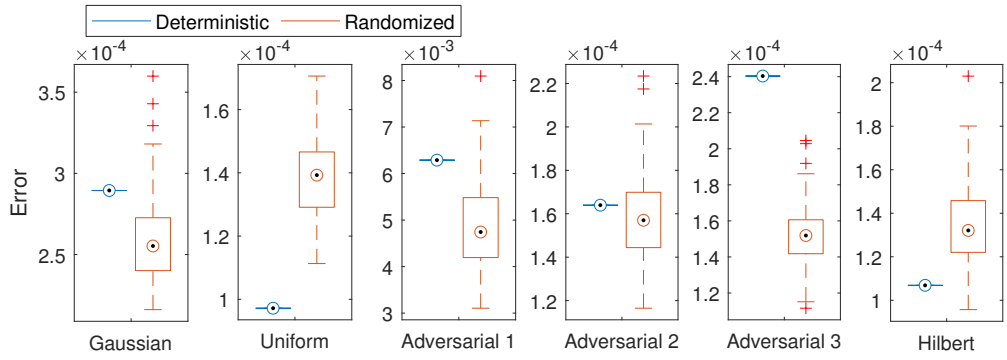

Figure 5.4: Error for deterministic ABC compared to the error of the randomized counterpart, over 100 realizations of the randomized algorithm. $\boldsymbol{A}, \boldsymbol{B} \in \mathbb{R}^{12\times 12}$ are drawn from six different random distributions, one in each subplot, and remain fixed over all realizations. The ABC is an instance of the 12×12 APA algorithm of [26].

5.3.2 Exact algorithm in single precision floating point arithmetic

We first consider Strassen's algorithm, without any perturbations so that it is exact, when the computations are done in single precision floating point arithmetic. In error computations, we use the double precision product for $\boldsymbol{AB}$ computed using the standard algorithm as the true value of the product. Recall that by "standard algorithm" we mean the $O(n^3)$ algorithm.

In the first experiment, we draw two Gaussian matrices $\boldsymbol{A}, \boldsymbol{B} \in \mathbb{R}^{80\times 80}$ and compute the quantity in (5.23), but with each $\hat{F}_i^{(Q)}$ replaced by $\hat{G}_i^{(Q)}$, where $\hat{G}_i^{(Q)}$ is the ith realization of $\hat{G}^{(Q)}$, for $n \in [10^4]$ and $Q \in [3]$. Figure 5.5 shows the results, where we also have included the error for the standard algorithm computed in single precision as a reference. Although it is clear that the randomized algorithms do not converge to the exact correct answer, it seems like their expectations perform better than the standard algorithm. Although the figure only shows the result for a specific random pair $(\boldsymbol{A}, \boldsymbol{B})$, we get qualitatively similar results every time we draw a new Gaussian matrix pair. The fact that the average of a single, or a few, outcomes of the randomized algorithms performs worse than the standard algorithm is to be expected, since Strassen's algorithm is more susceptible to numerical error that the standard algorithm. Figure 5.6 repeats this experiment, but with $\boldsymbol{A} = \boldsymbol{B} = \boldsymbol{H}$, where $\boldsymbol{H}$ is the 80×80 Hilbert matrix. The results are similar to those in Figure 5.5. Figures A.6–A.10 in the appendix provide additional results for when $\boldsymbol{A}, \boldsymbol{B}$ are Gaussian, uniform and type 1–3 adversarial.

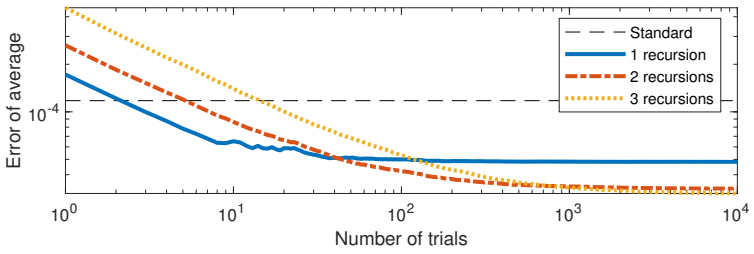

Figure 5.5: Error of average for randomized EBC compared to the standard $O(n^3)$ algorithm in single precision floating point arithmetic. $\boldsymbol{A}, \boldsymbol{B} \in \mathbb{R}^{80\times 80}$ are Gaussian and remain fixed throughout the experiment.

In the second experiment, we first compare the deterministic and three randomized versions

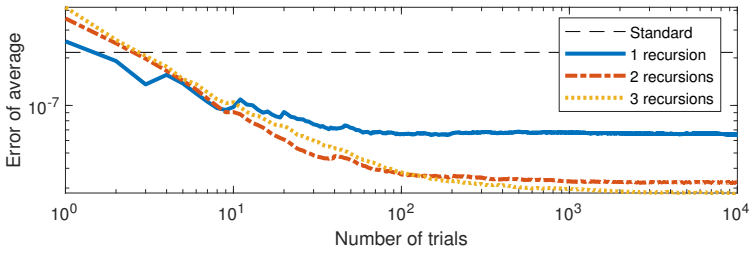

Figure 5.6: Same as Figure 5.5, but with $\boldsymbol{A}$ and $\boldsymbol{B}$ equal to the 80×80 Hilbert matrix.

of Strassen's algorithm, as well as the rescaling scheme proposed in [12]. The first randomized version uses both random permutations and random signs. The two other randomized versions use only random signs and only random permutations, respectively. We include these two variants to better understand how random signs and random permutations each impact the performance. These three randomized methods will be referred to as "fully randomized," "random sign" and "random permutation," respectively. The algorithm that only uses random permutations corresponds to the method suggested in [43].

The purpose of the rescaling scheme in [12] is to improve numerical stability of fast EBCs, and has two steps: Outside scaling and inside scaling. With outside scaling, $\boldsymbol{C} = \boldsymbol{AB}$ is computed via

$$\boldsymbol{C}_{\text{outside}} \stackrel{\text{def}}{=} \boldsymbol{D}_{\boldsymbol{A}} G^{(Q)}(\boldsymbol{D}_{\boldsymbol{A}}^{-1}\boldsymbol{A}, \boldsymbol{B}\boldsymbol{D}_{\boldsymbol{B}}^{-1})\boldsymbol{D}_{\boldsymbol{B}}, \tag{5.25}$$

where $\boldsymbol{D}_{\boldsymbol{A}} \stackrel{\text{def}}{=} \text{diag}(\max_j |a_{ij}|)$ and $\boldsymbol{D}_{\boldsymbol{B}} \stackrel{\text{def}}{=} \text{diag}(\max_i |b_{ij}|)$. With inside scaling, $\boldsymbol{C}$ is instead computed via

$$\boldsymbol{C}_{\text{inside}} \stackrel{\text{def}}{=} G^{(Q)}(\boldsymbol{AD}, \boldsymbol{D}^{-1}\boldsymbol{B}), \tag{5.26}$$

where $\boldsymbol{D} \stackrel{\text{def}}{=} \text{diag}(\sqrt{\max_j |b_{kj}| / \max_i |a_{ik}|})$. In exact arithmetic, the scaling matrices in (5.25) and (5.26) will cancel out, in which case $\boldsymbol{C}_{\text{outside}} = \boldsymbol{C}_{\text{inside}} = \boldsymbol{C}$. Both outside and inside scaling can be applied at the same time, as well as multiple times in an alternating fashion. The rescaling scheme that works best in numerical experiments in [12] does outside-inside rescaling twice. We use the same rescaling scheme in our experiments, which we refer to as "Rescaled 2x O-I." See Section 6 in [12], in particular Algorithm 3, for further details on the rescaling method.

For the experiment, we draw two random matrices $\boldsymbol{A}, \boldsymbol{B} \in \mathbb{R}^{320\times 320}$ and compute the quantity in (5.24), but with each $\hat{F}_i^{(Q)}$ replaced by $\hat{G}_i^{(Q)}$, for $i \in [100]$ and $Q \in [5]$. Figures 5.7–5.11 show the results for Gaussian, uniform and type 1–3 adversarial matrices. Figure 5.12 shows the result when $\boldsymbol{A}$ and $\boldsymbol{B}$ are both Hilbert matrices. These figures include the error of the standard algorithm computed in single precision as a reference.

When the matrices are Gaussian (Figure 5.7), all algorithms perform roughly the same with little variation between trials. For uniform matrices (Figure 5.8), the rescaling method performs about the same as the deterministic method. The fully randomized method has a lower error than the deterministic method, and it seems like this improvement comes from the random signs. For type 1 adversarial matrices (Figure 5.9), the rescaling method does remarkably well, achieving the same accuracy as the standard algorithm. The fully randomized algorithm outperforms the deterministic algorithm, and it seems like this improvement is coming from the random signs. For type 2 adversarial matrices (Figure 5.10), the fully randomized method will sometimes perform worse than the deterministic algorithm, but has a lower median error for 2 or more recursions. The rescaling method also improves on the deterministic method, although the median error for the fully randomized method is lower for 4 recursions or more. Type 3 adversarial matrices were specifically proposed in [12] to show a situation when rescaling does not work. This is clear in Figure 5.11, where the rescaling method has the same error as the deterministic method. Our fully randomized method, however, has a lower error, and it seems like both the random signs and random permutations contribute to this performance improvement. When the matrices are Hilbert matrices (Figure 5.12), the rescaling method does very poorly, with a much larger error than the deterministic method. Once again, our randomized method reduces the error compared to the deterministic method, with both the random signs and random permutations contributing to the improved performance. Figures A.11–A.15 in the appendix provide additional results for when $\boldsymbol{A}, \boldsymbol{B}$ are Gaussian, uniform, and type 1–3 adversarial.

In Figure 5.13, we give the results for some experiments which use an EBC for 12×12 matrix multiplication derived as in [24] from the 12×12 APA algorithm in [26] via the approach discussed

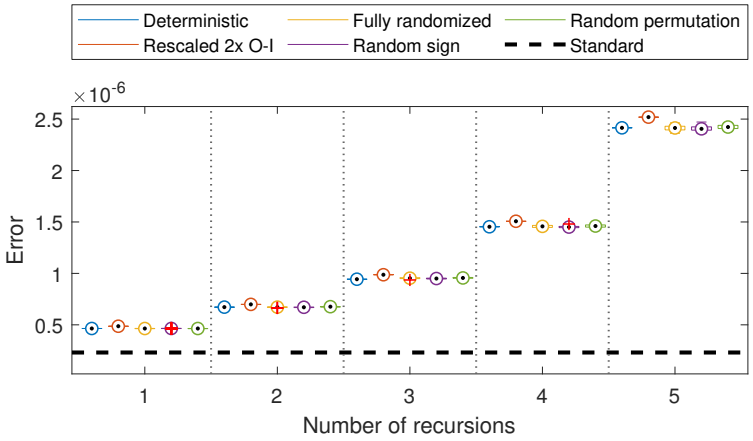

Figure 5.7: Error for different variants of the Strassen EBC in single precision floating point arithmetic, over 100 realizations of the randomized algorithms. $\boldsymbol{A}, \boldsymbol{B} \in \mathbb{R}^{320\times 320}$ are Gaussian and remain fixed over all realizations.

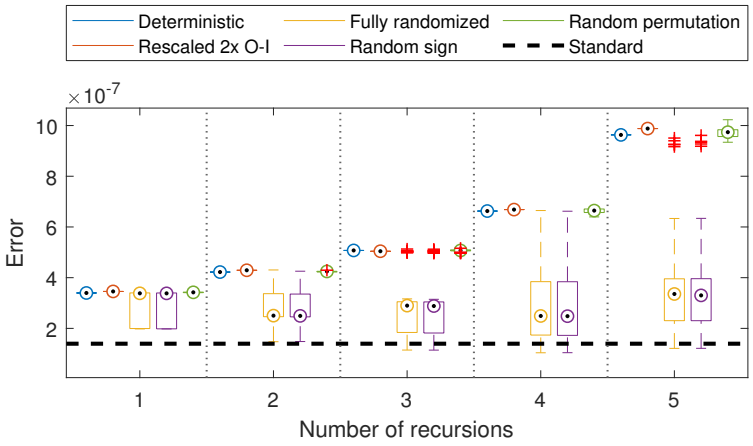

Figure 5.8: Same as Figure 5.7, but with uniform matrices.

in Section 5.1.1.[2] The APA algorithm in question has an error tensor whose entries are polynomials with maximum degree $d = 6$. Consequently, 7 distinct values of ε are required in (5.7) to derive an exact scheme. For this purpose, we choose $\varepsilon_1, \varepsilon_2, \ldots, \varepsilon_7$ to be $0.1, 0.2, \ldots, 0.7$. Our results are for single recursion experiments on 12×12 matrices drawn from different distributions. The results are similar to those for the Strassen EBC, except for the Gaussian case where the variability of the

[2] There appears to be a few typos in the definition of $w_r^{(s)}$ in Equation (5.2) in [24], which defines the APA scheme. We encourage the reader to consult our code for a corrected definition.

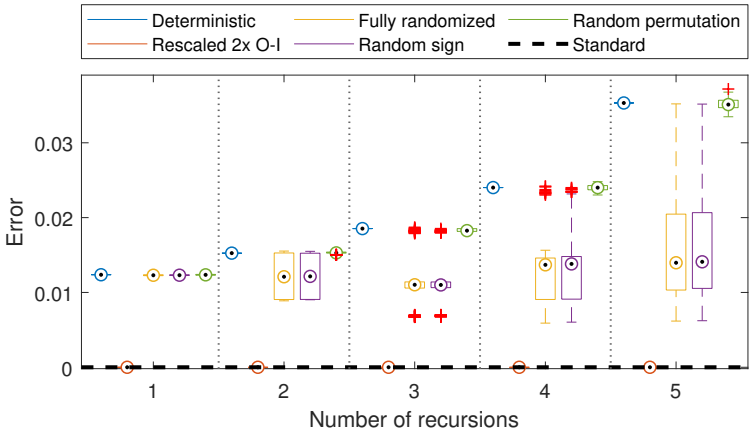

Figure 5.9: Same as Figure 5.7, but with type 1 adversarial matrices.

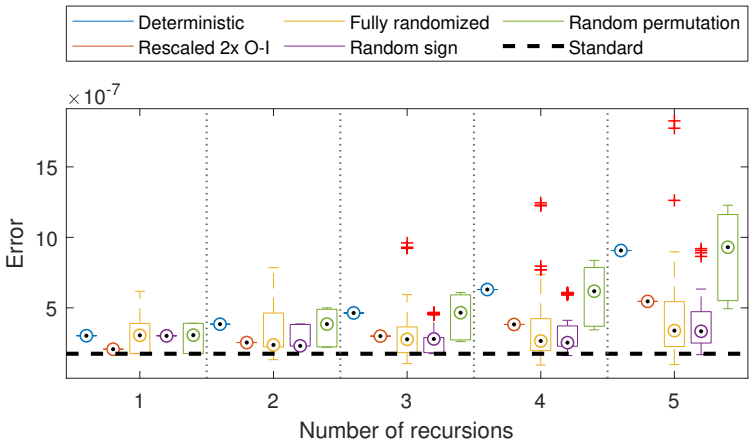

Figure 5.10: Same as Figure 5.7, but with type 2 adversarial matrices.

randomized algorithms is much higher. The variability of the randomized methods also appears to be somewhat higher on the other matrix types as well. Overall, our randomized methods perform favorably compared to the deterministic method, especially for the adversarial matrices and the Hilbert matrix. The rescaling method once again performs very well on type 1 adversarial matrices, but poorly on the Hilbert matrix.

Although our method has some variability in performance due to being randomized, it seems to reduce the error compared to the deterministic method more reliably on a wider range of matrices than the rescaling method does, especially when more recursions are used. Our method also works

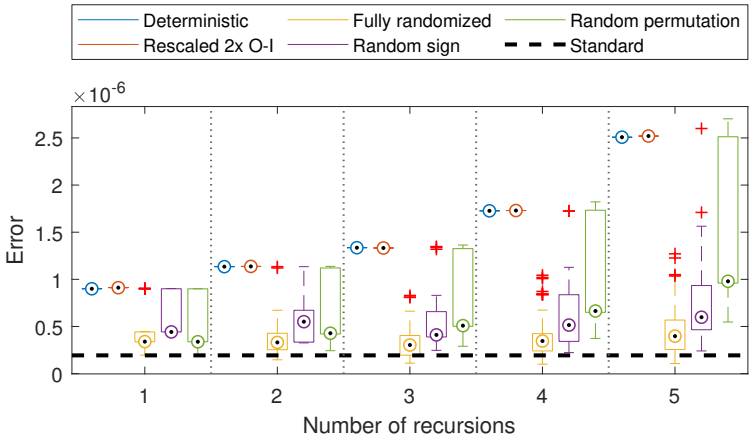

Figure 5.11: Same as Figure 5.7, but with type 3 adversarial matrices.

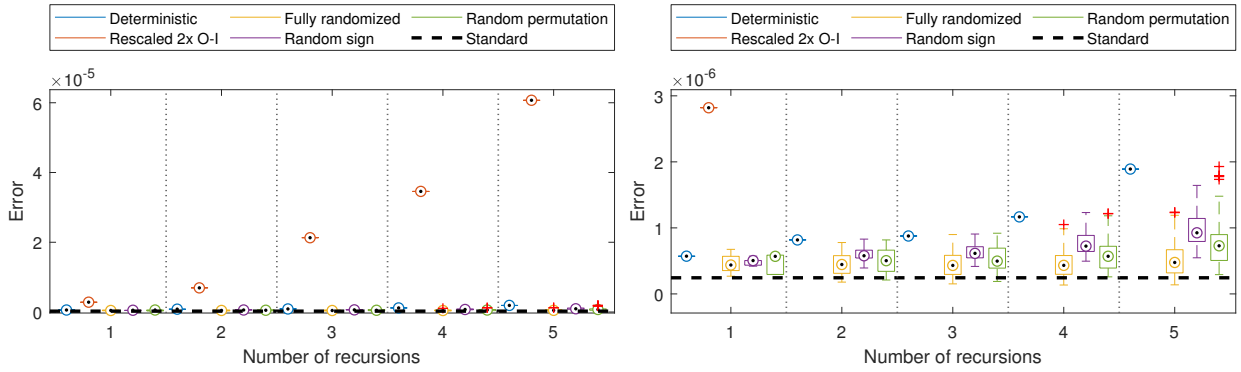

Figure 5.12: Same as Figure 5.7, but with Hilbert matrices. The left and right plots show the same results, with the right plot zoomed in closer to the interesting portion. The left plot is included to give a sense of the size of the errors for the rescaling method compared to the other methods.

well with the Hilbert matrix, which leads to large errors for the rescaling method. One benefit of our randomized method is that it can be done exactly even in low precision arithmetic, since it only involves permutation of rows/columns and the flipping of signs. The rescaling method, on the other hand, involves diagonal matrices with floating point numbers along the diagonals, which adds another potential source for numerical error in the algorithm. These experiments also indicate that both random signs and random permutations may separately help to reduce the error, and that combining the two seems to lower the error further.

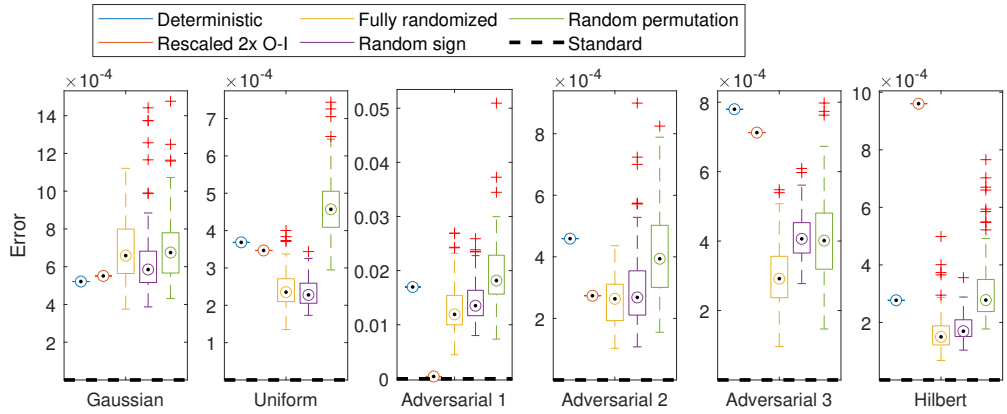

Figure 5.13: Error for different variants of the Bini [24] EBC in single precision floating point arithmetic, over 100 realizations of the randomized algorithms. $\boldsymbol{A}, \boldsymbol{B} \in \mathbb{R}^{12\times 12}$ are drawn from six different random distributions, one in each subplot, and remain fixed over all realizations.

5.3.3 Randomization of ABC derived from APA algorithm

In Section 5.1.1 we discussed APA algorithms and how to derive EBCs from them by taking a linear combination of a few instances of the APA algorithm following an idea of Bini [24]. Although the EBCs derived in this fashion are exact mathematically, they may suffer from greater numerical error than the standard $O(n^3)$ algorithm due to cancellation. In this subsection, we compare an instance of the EBC in [24], which is derived from the APA algorithm in [26], to a randomized version of the ABC we get by fixing the error parameter in the same APA algorithm. The goal with these experiments is to see if the expectation of our randomized ABC can perform better than the EBC. For the EBC, which is given by (5.7), we choose $\varepsilon_1, \varepsilon_2, \ldots, \varepsilon_7$ to be $0.1, 0.2, \ldots, 0.7$. To get an ABC, we choose $\varepsilon = 1.5\text{e}{-2}$ in the APA algorithm. This ABC is then randomized as in Definition 37. The experiments are similar to that in Figure 5.1, but with an added baseline which shows the performance of the EBC. The experiments are done for a single recursion on 12×12 matrices. All computations are done in single precision arithmetic. Figure 5.14 shows the results, which each subplot showing the outcome for a different kind of matrix. Based on these result, the expectation of the randomized ABC seems to perform better than the EBC.

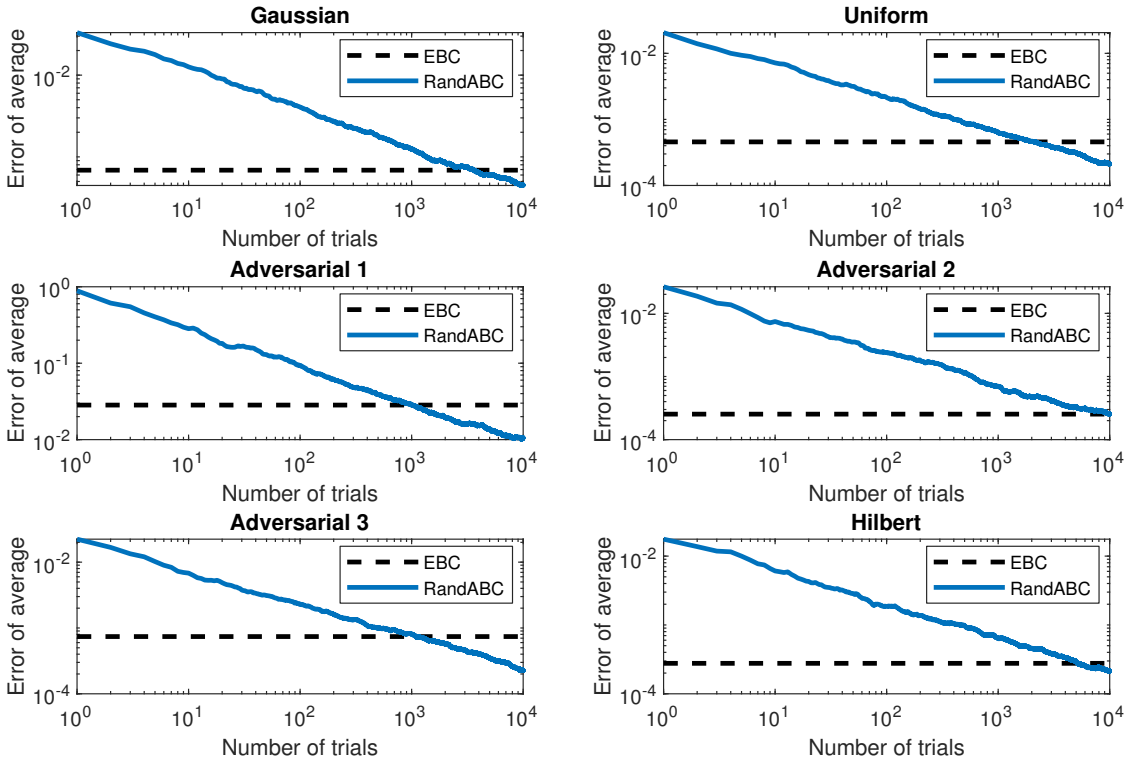

Figure 5.14: Comparison of EBC in [24] to randomized ABC derived from APA algorithm in [26].

5.4 Conclusion

In this paper we have suggested an approach for randomizing formulas for bilinear computation of matrix products which does not increase the asymptotic computational complexity. We have considered the implications of this approach when there are two sources of error: The first due to the algorithm itself being only approximately correct, and the second due to numerical error from using floating point arithmetic. We believe that our results are encouraging, and provide ideas for improving the properties of matrix multiplication when these error sources are present separately or together.

An interesting area for future research is to investigate other methods of randomization (e.g. combining the random sign changes in this paper with the fast Hadamard transform) and see if such a method can further improve the results. It would also be interesting to investigate how fast randomized approximate methods for matrix multiplication can be used in applications. One

potential application areas is for computations in neural networks, where some amount of error in the computation is acceptable, and where randomization may help improve robustness of the trained model. It would also be interesting to investigate how our randomization scheme can be used as a component in fast linear algebra algorithms, such as recursive dense matrix inversion; see [58] for details.

Chapter 6

Dynamic graph convolutional networks using the tensor M-product

6.1 Introduction

Graphs are popular data structures used to effectively represent interactions and structural relationships between entities in structured data domains. Inspired by the success of deep neural networks for learning representations in the image and language domains, recently, application of neural networks for graph representation learning has attracted much interest. A number of graph neural network (GNN) architectures have been explored in the contemporary literature for a variety of graph related tasks and applications [214, 228]. Methods based on graph convolution filters which extend convolutional neural networks (CNNs) to irregular graph domains are popular [38, 57, 114]. Most of these GNN models operate on a given, static graph.

In many real-world applications, the underlying graph changes over time, and learning representations of such dynamic graphs is essential. Examples include analyzing social networks [19], detecting fraud and crime in financial networks [173], traffic control [225], understanding neuronal activities in the brain [56], and analyzing contact tracing data [201]. In such dynamic settings, the temporal interdependence in the graph connections and features also play a substantial role. However, efficient GNN methods that handle time varying graphs and that capture the temporal correlations are lacking.

By **dynamic graph**, we refer to a sequence of graphs $\mathcal{G}^{(t)} = (V, \boldsymbol{A}^{(t)}, \boldsymbol{X}^{(t)})$, $t \in \{1, 2, \ldots, T\}$,

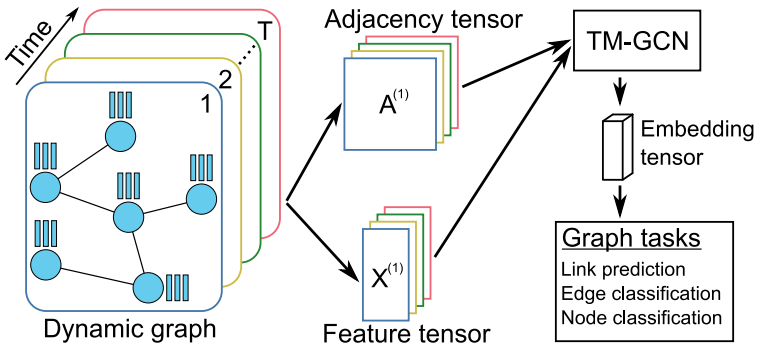

Figure 6.1: Our proposed TM-GCN approach.

with a fixed set V of N nodes, adjacency matrices $\boldsymbol{A}^{(t)} \in \mathbb{R}^{N\times N}$, and graph feature matrices $\boldsymbol{X}^{(t)} \in \mathbb{R}^{N\times F}$ where $\boldsymbol{X}^{(t)}_{n:} \in \mathbb{R}^{F}$ is the feature vector consisting of F features associated with node n at time t. The graphs can be weighted, and directed or undirected. They can also have additional properties like (time varying) node and edge classes, which would be stored in a separate structure. Suppose we only observe the first $T' < T$ graphs in the sequence. The goal of our method is to use these observations to predict some property of the remaining $T - T'$ graphs. In this paper, we consider edge classification, link prediction and node property prediction tasks.

In recent years, tensor constructs have been explored to effectively process high-dimensional data, in order to better leverage the multidimensional structure of such data [116]. Tensor based approaches have been shown to perform well in many applications. Recently, a new tensor framework called the **tensor M-product framework** [34, 110] was proposed that extends matrix based theory to high-dimensional architectures.

In this paper, we propose a novel tensor variant of the popular graph convolutional network (GCN) architecture [114], which we call TM-GCN. It captures correlation over time by leveraging the tensor M-product framework. The flexibility and matrix mimeticability of the framework, help us adapt the GCN architecture to tensor space. Figure 6.1 illustrates our method at a high level: First, the time varying adjacency matrices $\boldsymbol{A}^{(t)}$ and feature matrices $\boldsymbol{X}^{(t)}$ of the dynamic graph are aggregated into an adjacency tensor and a feature tensor, respectively. These tensors are then fed into our TM-GCN, which computes an embedding that can be used for a variety of tasks, such

as link prediction, and edge and node classification. GCN architectures are motivated by graph convolution filtering, i.e., applying filters/functions to the graph Laplacian [38], and we establish a similar connection between TM-GCN and spectral filtering of tensors. Such results suggest possible extensions of other convolution based GNNs such as [38, 57] for dynamic graphs using the tensor framework. The Message Passing Neural Network (MPNN) framework has been used to describe spatial convolution GNNs [81]. We show that TM-GCN is consistent with the MPNN framework, and accounts for spatial and temporal message passing. Experimental results on real datasets illustrate the performance of our method for the edge classification and link prediction tasks on dynamic graphs. We also demonstrate how TM-GCN can be used in an important application related to the COVID-19 pandemic. We show how GNNs can be used for early identification of individuals who are infected (potentially before they display symptoms) from contact tracing data and a dynamic graph based SEIR model [201].

6.2 Related work

Unsupervised embedding Unsupervised graph embedding techniques have been popular for link prediction on static graphs [41]. A number of dynamic graph embedding methods have been proposed recently, which extend the static ones. DANE [129] adapted the popular dimensionality reduction approaches such as Eigenmaps to time varying graphs by efficiently updating the eigenvectors from the prior ones. The popular random walk based methods have also been extended to obey the temporal order in recent works [160].

Numerous deep neural network based unsupervised learning methods have been developed for dynamic graph embedding. Examples include DynGEM [86], Know-Evolve [197], DyRep [198], Dynamic-Triad [229], and others. In most of these methods, a temporal smoothness regularization is used to obtain stable embedding across consecutive time-steps.

Supervised learning The idea of using graph convolution based on the spectral graph theory for GNNs was first introduced by [38]. [57] then proposed **Chebnet**, where the spectral filter was approximated by Chebyshev polynomials in order to make it faster and localized. [114]

presented the simplified GCN, a degree-one polynomial approximation of Chebnet, in order to speed up computation further and improve the performance. There are many other works that deal with GNNs when the graph and features are fixed/static; see the review papers [228] and [214] and references therein.

Recently, Li et. al [130] develop a diffusion convolutional RNN for traffic forecasting, where road networks are modeled assuming both the nodes and edges remain fixed over time, unlike in our setting. Seo et. al [184] devise the Graph Convolutional Recurrent Network for graphs with time varying features, while the edges are fixed over time. EdgeConv was proposed in [211], which is a neural network (NN) approach that applies convolution operations on static graphs in a dynamic fashion. [225] develop a temporal GCN method called T-GCN, which they apply for traffic prediction. Here too, the graph remains fixed over time, and only the features vary. [222] propose a method which they refer to as a tensor graph CNN. Here, the standard GCN [114] based on matrix algebra is considered, and a "cross graph convolution" layer is introduced to handle the time varying aspect of the dynamic graph. In particular, the cross graph convolution layer involves computing a parameterized Kronecker sum of the current adjacency matrix with the previously processed adjacency matrix, followed by a GCN layer. Recently, [135] described a tensor version of GCN for text classification, where the text semantics are represented as a three-dimensional graph tensor. This work neither considers time varying graphs, nor the tensor M-product framework.

The set of methods most relevant to our setting of learning embeddings of dynamic graphs use combinations of GNNs and recurrent architectures (RNN), to capture the graph structure and handle time dynamics, respectively. The approach in [149] uses Long Short-Term Memory (LSTM), a recurrent network, in order to handle time variations along with GNNs. They design architectures for semi-supervised node classification and for supervised graph classification. [173] presented a variant of GCN called **EvolveGCN**, where Gated Recurrent Units (GRUs) and LSTMs are coupled with a GCN to handle dynamic graphs. This paper is currently the state-of-the-art. [182] proposed the use of a temporal self-attention layer for dynamic graph representation learning. However, all these approaches are based on a heuristic RNN/GRU mechanism to evolve weights, and the models

are not time aware (time is not an explicit entity). [159] present a tensor NN which utilizes the tensor M-product framework. Their approach is applicable to image and other high-dimensional data that lie on regular grids.

6.3 Tensor M-product framework

Here, we cover the necessary preliminaries on tensors and the M-product framework. For a more general introduction to tensors, we refer the reader to the review paper [116]. In the present paper, a **tensor** is a three-dimensional array of real numbers denoted by boldface Euler script letters, e.g. $\boldsymbol{\mathcal{X}} \in \mathbb{R}^{I \times J \times T}$. Matrices are denoted by bold uppercase letters, e.g. $\boldsymbol{X}$; vectors are denoted by bold lowercase letter, e.g. $\boldsymbol{x}$; and scalars are denoted by lowercase letters, e.g. x. An element at position (i, j, t) in a tensor is denoted by subscripts, e.g. $\boldsymbol{\mathcal{X}}_{ijt}$, with similar notation for elements of matrices and vectors. A colon will denote all elements along that dimension; $\boldsymbol{X}_{i:}$ denotes the ith row of the matrix $\boldsymbol{X}$, and $\boldsymbol{\mathcal{X}}_{::k}$ denotes the kth frontal slice of $\boldsymbol{\mathcal{X}}$. The vectors $\boldsymbol{\mathcal{X}}_{ij:}$ are called the **tubes** of $\boldsymbol{\mathcal{X}}$.

The framework we consider relies on a new definition of the product of two tensors, called the M-product [34, 110, 113]. A distinguishing feature of this framework is that the M-product of two three-dimensional tensors is also three-dimensional, which is not the case for e.g. tensor contractions [116]. It allows one to elegantly generalize many classical numerical methods from linear algebra. The framework, originally developed for three-dimensional tensors, has been extended to handle tensors of dimension greater than three [113]. The following definitions 50–52 describe the M-product.

Definition 50 (M-transform)**.** Let $\boldsymbol{M} \in \mathbb{R}^{T \times T}$ be a mixing matrix. The **M-transform** of a tensor $\boldsymbol{\mathcal{X}} \in \mathbb{R}^{I \times J \times T}$ is denoted by $\boldsymbol{\mathcal{X}} \times_3 \boldsymbol{M} \in \mathbb{R}^{I \times J \times T}$ and defined elementwise as

$$(\boldsymbol{\mathcal{X}} \times_3 \boldsymbol{M})_{ijt} \stackrel{\text{def}}{=} \sum_{k=1}^{T} \boldsymbol{M}_{tk} \boldsymbol{\mathcal{X}}_{ijk}. \tag{6.1}$$

We say that $\boldsymbol{\mathcal{X}} \times_3 \boldsymbol{M}$ is in the **transformed space**. Note that if $\boldsymbol{M}$ is invertible, then $(\boldsymbol{\mathcal{X}} \times_3 \boldsymbol{M}) \times_3 \boldsymbol{M}^{-1} = \boldsymbol{\mathcal{X}}$. Consequently, $\boldsymbol{\mathcal{X}} \times_3 \boldsymbol{M}^{-1}$ is the **inverse M-transform** of $\boldsymbol{\mathcal{X}}$. The definition in (6.1)

may also be written in matrix form as $\mathcal{X} \times_3 \boldsymbol{M} \stackrel{\text{def}}{=} \text{fold}(\boldsymbol{M}\,\text{unfold}(\mathcal{X}))$, where the unfold operation takes the tubes of $\mathcal{X}$ and stack them as columns into a $T \times IJ$ matrix, and $\text{fold}(\text{unfold}(\mathcal{X})) = \mathcal{X}$.

Definition 51 (Facewise product)**.** Let $\mathcal{X} \in \mathbb{R}^{I\times J\times T}$ and $\mathcal{Y} \in \mathbb{R}^{J\times K\times T}$ be two tensors. The **facewise product**, denoted by $\mathcal{X} \triangle \mathcal{Y} \in \mathbb{R}^{I\times K\times T}$, is defined facewise as $(\mathcal{X} \triangle \mathcal{Y})_{::t} \stackrel{\text{def}}{=} \mathcal{X}_{::t}\mathcal{Y}_{::t}$.

Definition 52 (M-product)**.** Let $\mathcal{X} \in \mathbb{R}^{I\times J\times T}$ and $\mathcal{Y} \in \mathbb{R}^{J\times K\times T}$ be two tensors, and let $\boldsymbol{M} \in \mathbb{R}^{T\times T}$ be an invertible matrix. The **M-product**, denoted by $\mathcal{X} \star \mathcal{Y} \in \mathbb{R}^{I\times K\times T}$, is defined as

$$\mathcal{X} \star \mathcal{Y} \stackrel{\text{def}}{=} ((\mathcal{X} \times_3 \boldsymbol{M}) \triangle (\mathcal{Y} \times_3 \boldsymbol{M})) \times_3 \boldsymbol{M}^{-1}.$$

In the original formulation of the M-product, $\boldsymbol{M}$ was chosen to be the Discrete Fourier Transform (DFT) matrix, which allows efficient computation using the Fast Fourier Transform (FFT) [34, 113]. The framework was later extended for arbitrary invertible $\boldsymbol{M}$ (e.g. discrete cosine and wavelet transforms) [110]. Additional details are in the supplement.

6.4 Tensor dynamic graph embedding

Our approach is inspired by the first order GCN by [114] for static graphs, owed to its simplicity and effectiveness. For a graph with adjacency matrix $\boldsymbol{A}$ and feature matrix $\boldsymbol{X}$, a GCN layer takes the form $\boldsymbol{Y} = \sigma(\tilde{\boldsymbol{A}}\boldsymbol{X}\boldsymbol{W})$, where

$$\tilde{\boldsymbol{A}} \stackrel{\text{def}}{=} \tilde{\boldsymbol{D}}^{-1/2}(\boldsymbol{A} + \boldsymbol{I})\tilde{\boldsymbol{D}}^{-1/2},$$

$\tilde{\boldsymbol{D}}$ is diagonal with $\tilde{\boldsymbol{D}}_{ii} = 1 + \sum_j \boldsymbol{A}_{ij}$, $\boldsymbol{I}$ is the matrix identity, $\boldsymbol{W}$ is a matrix to be learned when training the NN, and σ is an activation function, e.g., ReLU. Our approach translates this to a tensor model by utilizing the M-product framework. We first introduce a tensor activation function $\hat{\sigma}$ which operates in the transformed space.

Definition 53. Let $\mathcal{A} \in \mathbb{R}^{I\times J\times T}$ be a tensor and σ an elementwise activation function. We define the activation function $\hat{\sigma}$ as $\hat{\sigma}(\mathcal{A}) \stackrel{\text{def}}{=} \sigma(\mathcal{A} \times_3 \boldsymbol{M}) \times_3 \boldsymbol{M}^{-1}$.

We can now define our proposed dynamic graph embedding. Let $\boldsymbol{\mathcal{A}} \in \mathbb{R}^{N \times N \times T}$ be a tensor with frontal slices $\boldsymbol{\mathcal{A}}_{::t} = \tilde{\boldsymbol{A}}^{(t)}$, where $\tilde{\boldsymbol{A}}^{(t)}$ is the normalization of $\boldsymbol{A}^{(t)}$. Moreover, let $\boldsymbol{\mathcal{X}} \in \mathbb{R}^{N \times F \times T}$ be a tensor with frontal slices $\boldsymbol{\mathcal{X}}_{::t} = \boldsymbol{X}^{(t)}$. Finally, let $\boldsymbol{\mathcal{W}} \in \mathbb{R}^{F \times F' \times T}$ be a weight tensor. We define our dynamic graph embedding as $\boldsymbol{\mathcal{Y}} = \boldsymbol{\mathcal{A}} \star \boldsymbol{\mathcal{X}} \star \boldsymbol{\mathcal{W}} \in \mathbb{R}^{N \times F' \times T}$. This computation can also be repeated in multiple layers. For example, a 2-layer formulation would be of the form

$$\boldsymbol{\mathcal{Y}} = \boldsymbol{\mathcal{A}} \star \hat{\sigma}(\boldsymbol{\mathcal{A}} \star \boldsymbol{\mathcal{X}} \star \boldsymbol{\mathcal{W}}^{(0)}) \star \boldsymbol{\mathcal{W}}^{(1)}.$$

One important consideration is how to choose the matrix $\boldsymbol{M}$ which defines the M-product. For time-varying graphs, we choose $\boldsymbol{M}$ to be lower triangular and banded so that each frontal slice $(\boldsymbol{\mathcal{A}} \times_3 \boldsymbol{M})_{::t}$ is a linear combination of the adjacency matrices $\boldsymbol{\mathcal{A}}_{::\max(1,t-b+1)}, \ldots, \boldsymbol{\mathcal{A}}_{::t}$, where we refer to b as the "bandwidth" of $\boldsymbol{M}$. This choice ensures that each frontal slice $(\boldsymbol{\mathcal{A}} \times_3 \boldsymbol{M})_{::t}$ only contains information from current and past graphs that are close temporally. We consider two variants of the lower banded triangular $\boldsymbol{M}$ matrix in the experiments; see the supplement for details. Another possibility is to treat $\boldsymbol{M}$ as a parameter matrix to be learned from the data.

In order to avoid over-parameterization and improve the performance, we choose the weight tensor $\boldsymbol{\mathcal{W}}$ (at each layer), such that each of the frontal slices of $\boldsymbol{\mathcal{W}}$ in the transformed domain remains the same, i.e., $(\boldsymbol{\mathcal{W}} \times_3 \boldsymbol{M})_{::t} = (\boldsymbol{\mathcal{W}} \times_3 \boldsymbol{M})_{::t'}\ \forall t, t'$. In other words, the parameters in each layer are shared and learned over all the training instances. This reduces the number of parameters to be learned significantly.

An embedding $\boldsymbol{\mathcal{Y}} \in \mathbb{R}^{N \times F' \times T}$ can now be used for various prediction tasks, like link prediction, and edge and node classification. In Section 6.5, we apply our method for edge classification and link prediction by using a model similar to that used by [173]: Given an edge between nodes m and n at time t, the predictive model is

$$p(m, n, t) \stackrel{\text{def}}{=} \text{softmax}(\boldsymbol{U}[(\boldsymbol{\mathcal{Y}} \times_3 \boldsymbol{M})_{m:t}, (\boldsymbol{\mathcal{Y}} \times_3 \boldsymbol{M})_{n:t}]^\top), \tag{6.2}$$

where $(\boldsymbol{\mathcal{Y}} \times_3 \boldsymbol{M})_{m:t} \in \mathbb{R}^{F'}$ and $(\boldsymbol{\mathcal{Y}} \times_3 \boldsymbol{M})_{n:t} \in \mathbb{R}^{F'}$ are row vectors, $\boldsymbol{U} \in \mathbb{R}^{C \times 2F'}$ is a weight matrix, and C the number of classes. Note that the embedding $\boldsymbol{\mathcal{Y}}$ is first M-transformed before the matrix $\boldsymbol{U}$

is applied to the appropriate feature vectors. This, combined with the fact that the tensor activation functions are applied elementwise in the transformed domain, allow us to avoid ever needing to apply the inverse M-transform. This approach reduces the computational cost, and has been found to improve performance in the edge classification task.

6.4.1 Theoretical motivation for TM-GCN

Here, we present the results that establish the connection between the proposed TM-GCN and spectral convolution of tensors, in particular spectral filtering and approximation on dynamic graphs. This is analogous to the graph convolution based on spectral graph theory in the GNNs by [38], [57], and [114]. All proofs and additional details are provided in Section B.4 of the supplement.

Let $\boldsymbol{\mathcal{L}} \in \mathbb{R}^{N\times N\times T}$ be a form of tensor Laplacian defined as $\boldsymbol{\mathcal{L}} \stackrel{\text{def}}{=} \boldsymbol{\mathcal{I}} - \boldsymbol{\mathcal{A}}$. Throughout the remainder of this subsection, we will assume that the adjacency matrices $\boldsymbol{A}^{(t)}$ are symmetric.

Proposition 54. *The tensor $\boldsymbol{\mathcal{L}}$ has an eigendecomposition $\boldsymbol{\mathcal{L}} = \boldsymbol{\mathcal{Q}} \star \boldsymbol{\mathcal{D}} \star \boldsymbol{\mathcal{Q}}^\top$.*

Definition 55 (Filtering)**.** Given a signal $\boldsymbol{\mathcal{X}} \in \mathbb{R}^{N\times 1\times T}$ and a function $g : \mathbb{R}^{1\times 1\times T} \to \mathbb{R}^{1\times 1\times T}$, we define the **tensor spectral graph filtering** of $\boldsymbol{\mathcal{X}}$ with respect to g as

$$\boldsymbol{\mathcal{X}}_{\text{filt}} \stackrel{\text{def}}{=} \boldsymbol{\mathcal{Q}} \star g(\boldsymbol{\mathcal{D}}) \star \boldsymbol{\mathcal{Q}}^\top \star \boldsymbol{\mathcal{X}}, \tag{6.3}$$

where

$$g(\boldsymbol{\mathcal{D}})_{mn:} \stackrel{\text{def}}{=} \begin{cases} g(\boldsymbol{\mathcal{D}}_{mn:}) & \text{if } m = n, \\ \boldsymbol{0} & \text{if } m \neq n. \end{cases}$$

In order to avoid the computation of an eigendecomposition, [57] uses a polynomial to approximate the filter function. We take a similar approach, and approximate $g(\boldsymbol{\mathcal{D}})$ with an M-product polynomial. For this approximation, we impose additional structure on g.

Assumption 56. Assume that $g : \mathbb{R}^{1\times 1\times T} \to \mathbb{R}^{1\times 1\times T}$ is defined as

$$g(\boldsymbol{\mathcal{V}}) \stackrel{\text{def}}{=} f(\boldsymbol{\mathcal{V}} \times_3 \boldsymbol{M}) \times_3 \boldsymbol{M}^{-1},$$

where f is defined elementwise as $f(\boldsymbol{\mathcal{V}} \times_3 \boldsymbol{M})_{11t} \stackrel{\text{def}}{=} f^{(t)}((\boldsymbol{\mathcal{V}} \times_3 \boldsymbol{M})_{11t})$ with each $f^{(t)} : \mathbb{R} \to \mathbb{R}$ continuous.

Proposition 57. *Suppose g satisfies Assumption 56. For any $\varepsilon > 0$, there exists an integer K and a set $\{\boldsymbol{\theta}^{(k)}\}_{k=1}^{K} \subset \mathbb{R}^{1\times 1\times T}$ such that*

$$\Big\| g(\boldsymbol{\mathcal{D}}) - \sum_{k=0}^{K} \boldsymbol{\mathcal{D}}^{\star k} \star \boldsymbol{\theta}^{(k)} \Big\| < \varepsilon,$$

where $\|\cdot\|$ is the tensor Frobenius norm, and where $\boldsymbol{\mathcal{D}}^{\star k} \stackrel{\text{def}}{=} \boldsymbol{\mathcal{D}} \star \cdots \star \boldsymbol{\mathcal{D}}$ is the M-product of k instances of $\boldsymbol{\mathcal{D}}$, with the convention that $\boldsymbol{\mathcal{D}}^{\star 0} = \boldsymbol{\mathcal{I}}$.

As in the work of [57], a tensor polynomial approximation allows us to approximate $\boldsymbol{\mathcal{X}}_{\text{filt}}$ in (6.3) without computing the eigendecomposition of $\boldsymbol{\mathcal{L}}$:

$$\begin{aligned} \boldsymbol{\mathcal{X}}_{\text{filt}} &= \boldsymbol{\mathcal{Q}} \star g(\boldsymbol{\mathcal{D}}) \star \boldsymbol{\mathcal{Q}}^{\top} \star \boldsymbol{\mathcal{X}} \\ &\approx \boldsymbol{\mathcal{Q}} \star \Big(\sum_{k=0}^{K} \boldsymbol{\mathcal{D}}^{\star k} \star \boldsymbol{\theta}^{(k)} \Big) \star \boldsymbol{\mathcal{Q}}^{\top} \star \boldsymbol{\mathcal{X}} \\ &= \Big(\sum_{k=0}^{K} \boldsymbol{\mathcal{L}}^{\star k} \star \boldsymbol{\theta}^{(k)} \Big) \star \boldsymbol{\mathcal{X}}. \end{aligned} \tag{6.4}$$

All that is necessary is to compute tensor powers of $\boldsymbol{\mathcal{L}}$. We can also define tensor polynomial analogs of the Chebyshev polynomials and do the approximation in (6.4) in terms of those instead of the tensor monomials $\boldsymbol{\mathcal{D}}^{\star k}$. We note that if a degree-one approximation is used, the computation in (6.4) becomes

$$\begin{aligned} \boldsymbol{\mathcal{X}}_{\text{filt}} &\approx (\boldsymbol{\mathcal{I}} \star \boldsymbol{\theta}^{(0)} + \boldsymbol{\mathcal{L}} \star \boldsymbol{\theta}^{(1)}) \star \boldsymbol{\mathcal{X}} \\ &= (\boldsymbol{\mathcal{I}} \star \boldsymbol{\theta}^{(0)} + (\boldsymbol{\mathcal{I}} - \boldsymbol{\mathcal{A}}) \star \boldsymbol{\theta}^{(1)}) \star \boldsymbol{\mathcal{X}}. \end{aligned}$$

Setting $\boldsymbol{\theta} \stackrel{\text{def}}{=} \boldsymbol{\theta}^{(0)} = -\boldsymbol{\theta}^{(1)}$, which is analogous to the parameter choice made in the degree-one approximation in [114], we get

$$\boldsymbol{\mathcal{X}}_{\text{filt}} \approx \boldsymbol{\mathcal{A}} \star \boldsymbol{\mathcal{X}} \star \boldsymbol{\theta}. \tag{6.5}$$

If we let $\boldsymbol{\mathcal{X}}$ contain F signals, i.e., $\boldsymbol{\mathcal{X}} \in \mathbb{R}^{N\times F\times T}$, and apply F' filters, (6.5) becomes

$$\boldsymbol{\mathcal{X}}_{\text{filt}} \approx \boldsymbol{\mathcal{A}} \star \boldsymbol{\mathcal{X}} \star \boldsymbol{\Theta} \in \mathbb{R}^{N\times F'\times T},$$

where $\mathbf{\Theta} \in \mathbb{R}^{F \times F' \times T}$. This is precisely our embedding model, with $\mathbf{\Theta}$ replaced by a learnable parameter tensor $\mathcal{W}$. These results show: (a) the connection between TM-GCN and spectral convolution of tensors, analogous to the GCN, and (b) that we can indeed develop higher order convolutional GNNs like [38, 57] for dynamic graphs using our framework.

6.4.2 Message passing framework

The Message Passing Neural Network (MPNN) framework is popularly used to describe spatial convolution GNNs [81]. The graph convolution operation is considered to be a message passing process, with information being passed from one node to another along the edges. The message passing phase of MPNN constitutes updating the hidden state $\boldsymbol{h}_{v,\ell}$ at node v in the ℓth layer with message $\boldsymbol{m}_{v,\ell+1}$ as

$$\boldsymbol{m}_{v,\ell+1} = \sum_{w \in N(v)} \Phi_\ell(\boldsymbol{h}_{v,\ell}, \boldsymbol{h}_{w,\ell}, e_{vw}),$$

$$\boldsymbol{h}_{v,\ell+1} = \Psi_\ell(\boldsymbol{h}_{v,\ell}, \boldsymbol{m}_{v,\ell+1}),$$

where $N(v)$ is the neighbors of v in the graph, e_{vw} is the edge between nodes v and w, Φ_ℓ is a message function, and Ψ_ℓ is an update function. A number of GNN models can be defined using this standard MPNN framework for static graphs. For the standard GCN model [114], we have $\Phi_\ell(\boldsymbol{h}_{v,\ell}, \boldsymbol{h}_{w,\ell}) = A_{v,w}\boldsymbol{h}_{w,\ell}$, where $A_{v,w}$ is the entry of adjacency matrix $\boldsymbol{A}$, and $\Psi_\ell(\boldsymbol{h}_{v,\ell}, \boldsymbol{m}_{v,\ell+1}) = \sigma(\boldsymbol{m}_{v,\ell+1})$ where σ is a pointwise non-linear function, e.g., ReLU.

In this paper, we consider dynamic graphs and the designed GNN has to do spatial and temporal message passing. That is, for a graph $\mathcal{G}^{(t)}$ at time t, the MPNN should be modeled such that the information/message is passed between neighboring nodes, as well as the corresponding nodes in the graphs $\{\mathcal{G}^{(t-1)}, \mathcal{G}^{(t-2)}, \ldots, \mathcal{G}^{(1)}\}$. Recently, a spatio-temporal message passing framework was defined for video processing in computer vision [151]. However, their framework does not account for time direction, and the graph is not considered to be evolving. We define the message passing framework for a dynamic graph as follows: The message passing phase will constitute updating the

Table 6.1: Dataset statistics.

				Window		Partitioning		
Dataset	Nodes	Edges	T	(days)	C	S_{train}	S_{val}	S_{test}
SBM	1,000	1,601,999	50	–	–	35	5	10
BitcoinOTC	6,005	35,569	135	14	2	95	20	20
BitcoinAlpha	7,604	24,173	135	14	2	95	20	20
Reddit	3,818	163,008	86	14	2	66	10	10
Chess	7,301	64,958	100	31	3	80	10	10

hidden state $\boldsymbol{h}_{v,\ell}^{(t)}$ at node v of graph $\mathcal{G}^{(t)}$ in the ℓth layer with message $\boldsymbol{m}_{v,\ell+1}^{(t)}$ as

$$\boldsymbol{m}_{v,\ell+1}^{(t)} = \sum_{w \in N(v)} \sum_{\tau=1}^{t} \Phi_{\ell}^{(\tau)} \left(\sum_{\tau=1}^{t} \Gamma_{\ell}^{(\tau)}(\boldsymbol{h}_{v,\ell}^{(\tau)}), \boldsymbol{h}_{w,\ell}^{(\tau)}, e_{vw}^{\tau} \right),$$

$$\boldsymbol{h}_{v,i+1}^{(t)} = \Psi_{\ell}(\boldsymbol{h}_{v,\ell}^{(t)}, \boldsymbol{m}_{v,\ell+1}^{(t)}).$$

Here, the function $\Gamma_{\ell}^{(\tau)}$ accounts for the message passing between hidden states over different time $\tau \leq t$, and function $\phi_{\ell}^{(\tau)}$ accounts for message passing between neighbors $N(v)$ over time $\tau \leq t$. The model accounts for extensive spatio-temporal message passing.

For the proposed TM-GCN model, we have a function $\Gamma_{\ell}^{(\tau)}(\boldsymbol{h}_{v,\ell}^{(\tau)}) = M_{t,\tau}\boldsymbol{h}_{v,\ell}^{(\tau)}$, where $M_{t,\tau}$ is the (t,τ) entry of the mixing matrix $\boldsymbol{M}$. The message function is $\Phi_{\ell}^{(\tau)}(\boldsymbol{z}_{v,\ell}^{(t)}, \boldsymbol{h}_{w,\ell}^{(\tau)}) = M_{t,\tau}A_{v,w,\tau}$, where $\boldsymbol{z}_{v,\ell}^{(t)} = \sum_{\tau=1}^{t} \Gamma^{(\tau)}(\boldsymbol{h}_{v,\ell}^{(\tau)})$, and $A_{v,w,\tau}$ is the entry of the adjacency tensor $\boldsymbol{\mathcal{A}}$. The update function is $\Psi_{\ell}(\boldsymbol{h}_{v,\ell}^{(t)}, \boldsymbol{m}_{v,\ell+1}^{(t)}) = \sigma(\boldsymbol{m}_{v,\ell+1}^{(t)})$ with an elementwise non-linear function σ.

Note that the above message passing model does not include the inverse transform $\boldsymbol{M}^{-1}$ as in the definition of the M-products. This is because, the M-transform is responsible for the temporal message passing and undoing it is not necessary. In our experiments too, we found that transforming back (applying the inverse transform $\boldsymbol{M}^{-1}$) did not yield improved results as suggested by the above MPNN model. This does not affect any of the theory presented in the previous section since the spectral filtering is performed in the transformed domain (see the supplement for details).

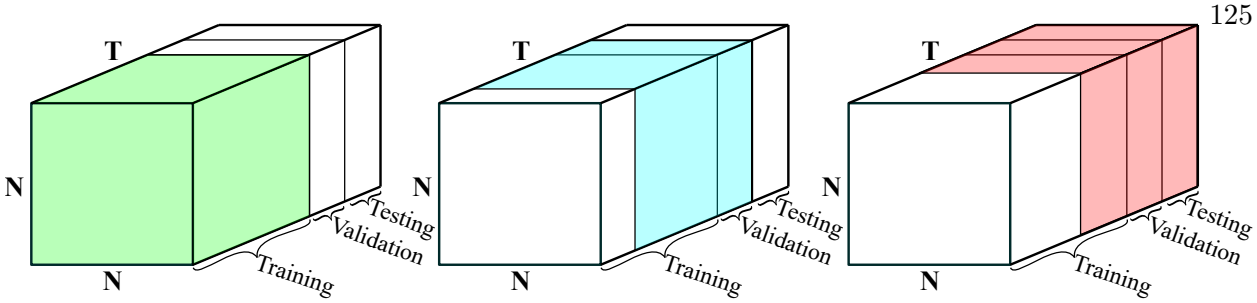

Figure 6.2: Partitioning of $\boldsymbol{\mathcal{A}}$ into training, validation and testing data.

Table 6.2: Results for edge classification. Performance measures are F1 score†or accuracy*. A higher value is better.

	Dataset			
Method	Bitcoin OTC†	Bitcoin Alpha†	Reddit†	Chess*
WD-GCN	0.3562	0.2533	**0.2337**	0.4311
EvolveGCN	0.3483	0.2273	0.2012	0.4351
GCN	0.3402	0.2381	0.1968	0.4342
TM-GCN - M1	0.3660	**0.3243**	0.2057	**0.4708**
TM-GCN - M2	**0.4361**	0.2466	0.1833	0.4513

Table 6.3: Results for link prediction. Performance measure is MAP. A higher value is better.

	Dataset				
Method	SBM	Bitcoin OTC	Bitcoin Alpha	Reddit	Chess
WD-GCN	0.9436	0.8071	0.8795	**0.3896**	0.1279
EvolveGCN	0.7620	0.6985	0.7722	0.2866	0.0915
GCN	0.9201	0.6847	0.7655	0.3099	0.0899
TM-GCN - M1	0.9684	0.8026	0.9318	0.2270	**0.1882**
TM-GCN - M2	**0.9799**	**0.8458**	**0.9631**	0.1405	0.1514

6.5 Numerical experiments

We first present results for edge classification and link prediction. We then show how we can use GNNs for predicting the state of individuals from COVID-19 contact tracing graphs.

6.5.1 Datasets and preprocessing

We consider five datasets (links to the datasets are in the supplement): The Bitcoin Alpha and OTC transaction datasets [173], the Reddit body hyperlink dataset [121], a chess results dataset [122], and SBM is the structure block matrix by [86]. The bitcoin datasets consist of transaction histories for users on two different platforms. Each node is a user, and each directed edge indicates a transaction and is labeled with an integer between -10 and 10 which indicates the senders trust for the receiver. We convert these labels to two classes: positive (trustworthy) and negative (untrustworthy). The Reddit dataset is built from hyperlinks from one subreddit to another. Each node represents a subreddit, and each directed edge is an interaction which is labeled with -1 for a hostile interaction or $+1$ for a friendly interaction. We only consider those subreddits which have a total of 20 interactions or more. In the chess dataset, each node is a player, and each directed edge represents a match with the source node being the white player and the target node being the black player. Each edge is labeled -1 for a black victory, 0 for a draw, and $+1$ for a white victory. Table 6.1 summarizes the statistics for the different datasets, where T is total # graphs and C is the # classes. The SBM dataset has no labels and hence we use it only for link prediction.

The data is temporally partitioned into T graphs, with each graph containing data from a particular time window. Both T and the time window length can vary between datasets. For each node-time pair (n, t) in these graphs, we compute the number of outgoing and incoming edges and use these two numbers as features. The adjacency tensor $\boldsymbol{\mathcal{A}}$ is then constructed as described in Section 6.4. The T frontal slices of $\boldsymbol{\mathcal{A}}$ are divided into S_{train} training slices, S_{val} validation slices, and S_{test} testing slices, which come sequentially after each other; see Figure 6.2 and Table 6.1.

Since the adjacency matrices corresponding to graphs are very sparse for these datasets, we apply the same technique as [173] and add the entries of each frontal slice $\boldsymbol{\mathcal{A}}_{::t}$ to the following $l-1$ frontal slices $\boldsymbol{\mathcal{A}}_{::t}, \ldots, \boldsymbol{\mathcal{A}}_{::(t+l-1)}$, where we refer to l as the "edge life." Note that this only affects $\boldsymbol{\mathcal{A}}$, and that the added edges are not treated as real edges in the classification and prediction problems.

The bitcoin and Reddit datasets are heavily skewed, with about 90% of edges labeled positively,

and the remaining labeled negatively. Since the negative instances are more interesting to identify (e.g. to prevent financial fraud or online hostility), we use the F1 score to evaluate the edge classification experiments on these datasets, treating the negative edges as the ones we want to identify. The classes are more well-balanced in the chess dataset, so we use accuracy to evaluate those edge classification experiments.

6.5.2 Graph tasks

For the link prediction experiments, we follow [173] and use negative sampling to construct non-existing edges, and use mean average precision (MAP) as a performance measure. The negative sampling is done so that 5% of edges are existing edges for each time slice, and all other edges are non-existing. Precise definitions of the different performance measures we use are given in Section B.2 of the supplement.

For edge classification, we use an embedding $\mathcal{Y}_{\text{train}} = \mathcal{A}_{::(1:S_{\text{train}})} \star \mathcal{X}_{::(1:S_{\text{train}})} \star \mathcal{W}$ for training. When computing the embeddings for the validation and testing data, we still need S_{train} frontal slices of $\mathcal{A}$, which we get by using a sliding window of slices. This is illustrated in Figure 6.2, where the green, blue and red blocks show the frontal slices used when computing the embeddings for the training, validation and testing data, respectively. The embeddings for the validation and testing data are $\mathcal{Y}_{\text{val}} = \mathcal{A}_{::(S_{\text{val}}+1:S_{\text{train}}+S_{\text{val}})} \star \mathcal{X}_{::(S_{\text{val}}+1:S_{\text{train}}+S_{\text{val}})} \star \mathcal{W}$ and $\mathcal{Y}_{\text{test}} = \mathcal{A}_{::(S_{\text{val}}+S_{\text{test}}+1:T)} \star \mathcal{X}_{::(S_{\text{val}}+S_{\text{test}}+1:T)} \star \mathcal{W}$, respectively. For link prediction, we use the same embeddings, with the only difference that the embedding blocks contain $S_{\text{train}} - 1$ slices. This is necessary since we want to use information up to time t to predict edge existence at time $t + 1$. Preliminary experiments with 2-layer architectures did not show convincing improvements in performance. We believe this is due to the fact that the datasets only have two features, and that a 1-layer architecture therefore is sufficient for extracting relevant information in the data.

For training, we use the cross entropy loss function:

$$\text{loss} = -\sum_{t} \sum_{(m,n)\in E_t} \sum_{c=1}^{C} \alpha_c f(m,n,t)_c \log(p(m,n,t)_c), \tag{6.6}$$

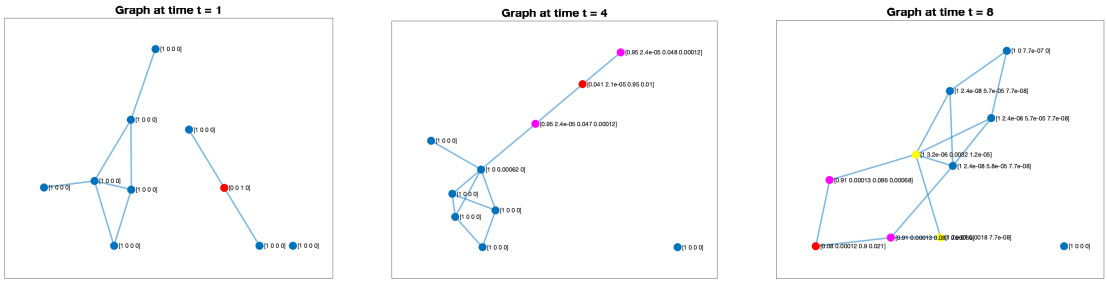

Figure 6.3: Graphical SEIR model disease transmission visualization.

where $\alpha \in \mathbb{R}^C$ is a vector summing to 1 which contains the weight of each class. For edge classification, $f(m,n,t) \in \mathbb{R}^C$ is a one-hot vector encoding the true class of the edge (m,n) at time t. For link prediction, $f(m,n,t) \in \mathbb{R}^2$ is also a one-hot vector, but now encoding if the edge is an existing or non-existing edge. As appropriate, we weigh the minority class more heavily in the loss function for skewed datasets, and treat α as a hyperparameter. See Section B.2 of the supplement for further details on the experiment setup, including the training setup and how hyperparameter tuning is done.

The experiments are implemented in PyTorch with some preprocessing done in Matlab. Our code is available at `https://github.com/IBM/TM-GCN`. In the experiments, we use an edge life of $l = 10$, a bandwidth $b = 20$, and $F' = 6$ output features. For TM-GCN, we consider two variants of the $\boldsymbol{M}$ matrix (M1 and M2); see the supplement for details.

We compare our method with three other methods. The first one is a variant of the WD-GCN by [149], which they specify in Equation (8a) of their paper. For the LSTM layer in their description, we use 6 output features instead of N. This is to avoid overfitting and make the method more comparable to ours which uses 6 output features. The second method is a 1-layer variant of EvolveGCN-H by [173]. The third method is a simple baseline which uses a 1-layer version of the GCN by [114]. It uses the same weight matrix $\boldsymbol{W}$ for all temporal graphs. Both EvolveGCN-H and the baseline GCN use 6 output features as well. We use the prediction model (6.2) as the final layer in all models we compare.

Tables 6.2 and 6.3 show the results for edge classification and link prediction, respectively.

Table 6.4: COVID-19 Data: Mean absolute error and error ratio for infection state I prediction.

Methods	COVID-19 Dataset	
	Error	Ratio
WD-GCN	1.667	0.337
EvolveGCN	4.969	0.912
TM-GCN	**1.466**	**0.278**

For edge classification, our method outperforms the other methods on the two bitcoin datasets and the chess dataset, with WD-GCN performing best on the Reddit dataset. For link prediction, our method outperforms the other methods on the SBM, bitcoin and chess datasets. For Reddit, our method performs worse than the other methods. Results from some additional experiments are provided in Section B.3 of the supplement.

6.5.3 COVID-19 application

One of the primary challenges related to the COVID-19 pandemic has been the issue of identifying early the individuals who are infected (ideally before they display symptoms) and prescribe testing. Here, we demonstrate how we can potentially use GNNs on contact tracing data to achieve this.

Contact tracing, a process where interactions between individuals (infected and others) are carefully tracked, has been shown the be an effective method for managing the spread of COVID-19. A variety of contact tracing methodologies have been used today around the world, see [4, 201] for lists. Recently, Ubaru et. al [201] presented a probabilistic graphical SEIR epidemiological model (Susceptible, Exposed, Infected, Recovered) to describe the dynamics of the disease transmission. Their model considers a dynamic graph (with individuals as nodes) that accounts for the interactions between individuals obtained from contact tracing, and uses a stochastic diffusion-reaction model to describe the disease transmission over the graph.

The novel SEIR model in [201] considers the graph Laplacian $\boldsymbol{L}_t$ (from contact tracing data) at each time t and describes the evolution of the state $\{S, E, I, R\}$ for each node/individual. Figure 6.3 illustrates the state $\{S, E, I, R\}$ evolution as defined by the model on a sample dynamic graph with

10 individuals (for easy visualization). We see how the infection (one individual as red node in first graph) transmits, we have magenta nodes with $I > 0.04$, and the yellow nodes with $I > 0.002$, and we note the interactions and the state change over time. Here, we show how we can use dynamic GNNs to predict the infection state I at time $T + 1$, using only the dynamic graphs up to time T, when the true SEIR model is unknown.

We consider a simulation with $N = 1000$ individuals and total time $T = 100$. We simulate the contact tracing dynamic graph as in [201], and assume at each time t a small number of individuals are tested at random for both IgM (if positive state I is set to 1) and IgG (state R is set to 1) antigen tests. The state of the remaining individuals are determined by the SEIR model. We train the dynamic GNNs on the first $T = 80$ time instances and test the GNNs on the remaining 20 time instances. Our goal is to train a GNN that learns the relation between the contact tracing graphs and the infection state (some exact values for those who were tested and others from the SEIR model), in order to **better predict** the individuals' state I at time $t + 1$ than just using the SEIR model. Table 6.4 gives the mean absolute error and error ratio obtained by the three dynamic GNNs for infection state I prediction on the test time instances. We note that, the proposed TM-GCN yields best results among the three methods, since it has a better time awareness (explicitly considers b previous time instances via the M-product) than others. Using such predictions, we can issue early warnings to individuals who are infected and prescribe testing.

6.6 Conclusion

We have presented a novel approach for dynamic graph embedding which leverages the tensor M-product framework. We used it for edge classification and link prediction in experiments on five datasets, where it performed competitively compared to state-of-the-art methods. We also demonstrated the method's effectiveness in an important application related to the COVID-19 pandemic. Future research directions include further developing the theoretical guarantees for the method, investigating optimal structure and learning of the transform matrix $\boldsymbol{M}$, using the method for other prediction tasks, and investigating how to utilize deeper architectures for dynamic graph

learning.

Chapter 7

A sampling-based method for tensor ring decomposition

7.1 Introduction

Tensor decomposition has recently found many applications in machine learning and related areas. Examples include parameter reduction in neural networks [80, 161, 216, 217, 219], understanding the expressivity of deep neural networks [52, 112], supervised learning [162, 190], filter learning [93, 181], image factor analysis and recognition [134, 204], scaling up Gaussian processes [103], multimodal feature fusion [98], natural language processing [126], data mining [171], feature extraction [17], and tensor completion [209]. The CP and Tucker decompositions are the two most popular tensor decompositions that each have strengths and weaknesses [116]. The number of parameters in the CP decomposition scales linearly with the tensor dimension, but finding a good decomposition numerically can be challenging. The Tucker decomposition is usually easier to work with numerically, but the number of parameters grows exponentially with tensor dimension. To address these challenges, decompositions based on tensor networks have recently gained in popularity. The tensor train (TT) is one such decomposition [167]. The number of parameters in a TT scales linearly with the tensor dimension. The tensor ring (TR) decomposition is a generalization of the TT decomposition. It is more expressive than the TT decomposition (see Remark 58), but maintains the linear scaling of the number of parameters with increased tensor dimension. Although the TR decomposition is known to have certain numerical stability issues it usually works well in practice, achieving a better compression ratio and lower storage cost than the TT decomposition [153].

It is crucial to have efficient algorithms for computing the decomposition of a tensor. This

task is especially challenging for large and high-dimensional tensors. In this paper, we present a sampling-based method for TR decomposition which has complexity sublinear in the number of input tensor entries. The method fits the TR core tensors by iteratively updating them in an alternating fashion. Each update requires solving a least squares problem. We use leverage score sampling to speed up the solution of these problems and avoid having to consider all elements in the input tensor for each update. The system matrices in the least squares problems have a particular structure that we take advantage of to efficiently estimate the leverage scores. To summarize, we make the following contributions:

- Propose a sampling-based method for TR decomposition which has complexity sublinear in the number of input tensor entries.
- Provide relative-error guarantees for the sampled least squares problems we use in our algorithm.
- Compare our proposed algorithm to four other methods in experiments on both synthetic and real data.
- Provide an example of how our method can be used for rapid feature extraction.

7.2 Related work

Leverage score sampling has been used previously for tensor decomposition. Cheng et al. [46] use it to reduce the size of the least squares problems that come up in the alternating least squares (ALS) algorithm for CP decomposition. They efficiently estimate the leverage scores by exploiting the Khatri–Rao structure of the design matrices in these least squares problems and prove additive-error guarantees. Larsen and Kolda [124] further improve this method by combining repeatedly sampled rows and by using a hybrid deterministic–random sampling scheme. These techniques make the method faster without sacrificing accuracy. Our paper differs from these previous works since we consider a different kind of tensor decomposition. Since the TR decomposition has core tensors

instead of factor matrices, the leverage score estimation formulas used in the previous works cannot be used for TR decomposition.

There are previous works that develop randomized algorithms for TR decomposition. Yuan et al. [220] develop a method which first applies a randomized variant of the HOSVD Tucker decomposition, followed by a TR decomposition of the core tensor using either a standard SVD- or ALS-based algorithm. Finally, the TR cores are contracted appropriately with the Tucker factor matrices to get a TR decomposition of the original tensor. Ahmadi-Asl et al. [2] present several TR decomposition algorithms that incorporate randomized SVD. In Section 7.5, we compare our proposed algorithm to methods by Yuan et al. [220] and Ahmadi-Asl et al. [2] in experiments.

Papers that develop randomized algorithms for other tensor decompositions include the works by Wang et al. [210], Battaglino et al. [15], Yang et al. [215] and Aggour et al. [1] for the CP decomposition; Biagioni et al. [22] and Malik and Becker [144] for tensor interpolative decomposition; Drineas and Mahoney [68], Tsourakakis [199], da Costa et al. [53], Malik and Becker [142], Sun et al. [194] and Minster et al. [156] for the Tucker decomposition; Zhang et al. [221] and Tarzanagh and Michailidis [196] for t-product-based decompositions; and Huber et al. [100] and Che and Wei [45] for the TT decomposition.

Skeleton approximation and other sampling-based methods for TR decomposition include the works by Espig et al. [76] and Khoo et al. [111]. Espig et al. [76] propose a skeleton/cross approximation type method for the TR format with complexity linear in the dimensionality of the tensor. Khoo et al. [111] propose an ALS-based scheme for constructing a TR decomposition based on only a few samples of the input tensor. Papers that use skeleton approximation and sampling techniques for other types of tensor decomposition include those by Mahoney et al. [140], Oseledets et al. [168], Oseledets and Tyrtyshnikov [166], Caiafa and Cichocki [42], and Friedland et al. [78].

7.3 Preliminaries

By tensor, we mean a multidimensional array of real numbers. Boldface Euler script letters, e.g. $\boldsymbol{\mathcal{X}}$, denote tensors of dimension 3 or greater; bold uppercase letters, e.g. $\boldsymbol{X}$, denote matrices;

bold lowercase letters, e.g. $\boldsymbol{x}$, denote vectors; and regular lowercase letters, e.g. x, denote scalars. Elements of tensors, matrices and vectors are denoted in parentheses. For example, $\boldsymbol{\mathcal{X}}(i_1, i_2, i_3)$ is the element on position (i_1, i_2, i_3) in the 3-way tensor $\boldsymbol{\mathcal{X}}$, and $\boldsymbol{y}(i_1)$ is the element on position i_1 in the vector $\boldsymbol{y}$. A colon is used to denote all elements along a dimension. For example, $\boldsymbol{X}(i, :)$ denotes the ith row of the matrix $\boldsymbol{X}$. For a positive integer I, define $[I] \stackrel{\text{def}}{=} \{1, \dots, I\}$. For indices $i_1 \in [I_1], \dots, i_N \in [I_N]$, the notation $\overline{i_1 i_2 \cdots i_N} \stackrel{\text{def}}{=} \sum_{n=1}^{N} i_n \prod_{j=1}^{n-1} I_j$ will be helpful when working with reshaped tensors. $\|\cdot\|_{\mathrm{F}}$ denotes the Frobenius norm of matrices and tensors, and $\|\cdot\|_2$ denotes the Euclidean norm of vectors.

7.3.1 Tensor ring decomposition

Let $\boldsymbol{\mathcal{X}} \in \mathbb{R}^{I_1 \times \cdots \times I_N}$ be an N-way tensor. For $n \in [N]$, let $\mathcal{G}^{(n)} \in \mathbb{R}^{R_{n-1} \times I_n \times R_n}$ be 3-way tensors with $R_0 = R_N$. A rank-$(R_1, \dots, R_N)$ TR decomposition of $\boldsymbol{\mathcal{X}}$ takes the form

$$\boldsymbol{\mathcal{X}}(i_1, \dots, i_N) = \sum_{r_1, \dots, r_N} \prod_{n=1}^{N} \mathcal{G}^{(n)}(r_{n-1}, i_n, r_n), \tag{7.1}$$

where each r_n in the sum goes from 1 to R_n and $r_0 = r_N$. $\mathcal{G}^{(1)}, \dots, \mathcal{G}^{(N)}$ are called **core tensors**. We will use $\mathrm{TR}((\mathcal{G}^{(n)})_{n=1}^{N})$ to denote a tensor ring with cores $\mathcal{G}^{(1)}, \dots, \mathcal{G}^{(N)}$. The name of the decomposition comes from the fact that it looks like a ring in tensor network[1] notation; see Figure 7.1. The TT decomposition is a special case of the TR decomposition with $R_0 = R_N = 1$, which corresponds to severing the connection between $\mathcal{G}^{(1)}$ and $\mathcal{G}^{(5)}$ in Figure 7.1. The TR decomposition can therefore also be interpreted as a sum of tensor train decompositions with the cores $\mathcal{G}^{(2)}, \dots, \mathcal{G}^{(N-1)}$ in common.

Remark 58 (Expressiveness of TR vs TT)**.** Since the TT decomposition is a special case of the TR decomposition, any TT is also a TR with the same number of parameters. So the TR decomposition is **at least as expressive** as the TT decomposition. To see that the TR decomposition is **strictly more expressive** than the TT decomposition, consider the following example: Let $\boldsymbol{\mathcal{X}} = \mathrm{TR}(\mathcal{G}, \mathcal{G}, \mathcal{G}, \mathcal{G})$ where $\mathcal{G} \in \mathbb{R}^{2 \times 2 \times 2}$ has entries $1, 2, \dots, 8$ ($\mathcal{G} =$ `reshape(1:8,2,2,2)` in Matlab).

[1] See Cichocki et al. [48, 49] for an introduction to tensor networks.

This TR of $\mathcal{X}$ requires $4 \cdot (2 \cdot 2 \cdot 2) = 32$ parameters. The TT ranks of $\mathcal{X}$ are $(2, 4, 2)$ (computed via Theorem 8.8 by Ye and Lim [218]). An exact TT decomposition of $\mathcal{X}$ therefore requires $2 \cdot (2 \cdot 2) + 2 \cdot (2 \cdot 2 \cdot 4) = 40$ parameters.

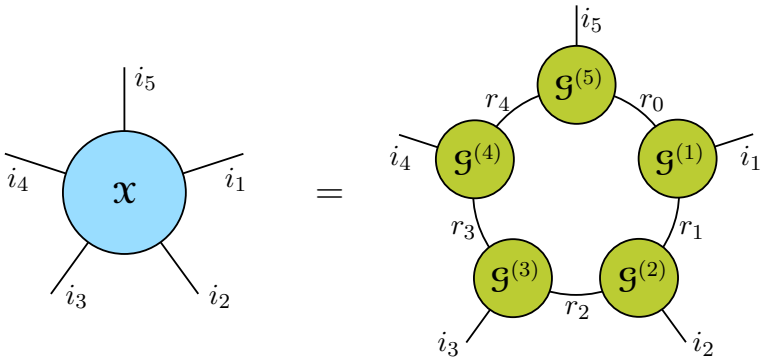

Figure 7.1: Tensor ring decomposition of 5-way tensor.

The problem of fitting $\operatorname{TR}((\mathcal{G}^{(n)})_{n=1}^N)$ to a data tensor $\mathcal{X}$ can be written as the minimization problem

$$\underset{\mathcal{G}^{(1)},\dots,\mathcal{G}^{(N)}}{\arg\min} \| \operatorname{TR}((\mathcal{G}^{(n)})_{n=1}^N) - \mathcal{X} \|_{\mathrm{F}}, \tag{7.2}$$

where the size of each core tensor is fixed. There are two common approaches to this fitting problem: one is SVD based and the other uses ALS. We describe the latter below since it is what we use in our work, and refer to Zhao et al. [226] and Mickelin and Karaman [153] for a description of the SVD-based approach.

Definition 59. The **classical mode-n unfolding** of $\mathcal{X}$ is the matrix $\boldsymbol{X}_{(n)} \in \mathbb{R}^{I_n \times \prod_{j \neq n} I_j}$ defined elementwise via

$$\boldsymbol{X}_{(n)}(i_n, \overline{i_1 \cdots i_{n-1} i_{n+1} \cdots i_N}) \stackrel{\text{def}}{=} \mathcal{X}(i_1, \dots, i_N).$$

The **mode-n unfolding** of $\mathcal{X}$ is the matrix $\boldsymbol{X}_{[n]} \in \mathbb{R}^{I_n \times \prod_{j \neq n} I_j}$ defined elementwise via

$$\boldsymbol{X}_{[n]}(i_n, \overline{i_{n+1} \cdots i_N i_1 \cdots i_{n-1}}) \stackrel{\text{def}}{=} \mathcal{X}(i_1, \dots, i_N).$$

Definition 60. By merging all cores except the nth, we get a **subchain tensor**

$$\mathcal{G}^{\neq n} \in \mathbb{R}^{R_n \times \prod_{j \neq n} I_j \times R_{n-1}}$$

defined elementwise via

$$\mathcal{G}^{\neq n}(r_n, \overline{i_{n+1} \dots i_N i_1 \dots i_{n-1}}, r_{n-1}) \stackrel{\text{def}}{=} \sum_{\substack{r_1,\dots,r_{n-2} \\ r_{n+1},\dots,r_N}} \prod_{\substack{j=1 \\ j \neq n}}^{N} \mathcal{G}^{(j)}(r_{j-1}, i_j, r_j).$$

It follows directly from Theorem 3.5 in Zhao et al. [226] that the objective in (7.2) can be rewritten as

$$\| \operatorname{TR}((\mathcal{G}^{(n)})_{n=1}^{N}) - \mathcal{X} \|_{\mathrm{F}} = \| \boldsymbol{G}_{[2]}^{\neq n} \boldsymbol{G}_{(2)}^{(n)\top} - \boldsymbol{X}_{[n]}^{\top} \|_{\mathrm{F}}.$$

The ALS approach to finding an approximate solution to (7.2) is to keep all cores except the nth fixed and solve the least squares problem above with respect to that core, and then repeat this multiple times for each $n \in [N]$ until some termination criterion is met. We summarize the approach in Algorithm 7. The initialization on line 1 can be done by, e.g., drawing each core entry independently from a standard normal distribution. Notice that the first core does not need to be initialized since it will be immediately updated. Normalization steps can also be added to the algorithm, but such details are omitted here. Throughout this paper we make the reasonable assumption that $\mathcal{G}^{\neq n} \neq \mathbf{0}$ for all $n \in [N]$ during the execution of the ALS-based methods. This is discussed further in Remark 86 in the supplement.

Algorithm 7: TR-ALS [226]

Input: $\mathcal{X} \in \mathbb{R}^{I_1 \times \cdots \times I_N}$, target ranks $(R_1, \dots, R_N)$
Output: TR cores $\mathcal{G}^{(1)}, \dots, \mathcal{G}^{(N)}$

1 Initialize cores $\mathcal{G}^{(2)}, \dots, \mathcal{G}^{(N)}$
2 **repeat**
3 **for** $n = 1, \dots, N$ **do**
4 Compute $\boldsymbol{G}_{[2]}^{\neq n}$ from cores
5 Update $\mathcal{G}^{(n)} = \arg\min_{\mathcal{Z}} \| \boldsymbol{G}_{[2]}^{\neq n} \boldsymbol{Z}_{(2)}^{\top} - \boldsymbol{X}_{[n]}^{\top} \|_{\mathrm{F}}$
6 **end**
7 **until termination criteria met**
8 **return** $\mathcal{G}^{(1)}, \dots, \mathcal{G}^{(N)}$

7.3.2 Leverage score sampling

Let $\boldsymbol{A} \in \mathbb{R}^{K \times L}$ and $\boldsymbol{y} \in \mathbb{R}^{K}$ where $K \gg L$. Solving the overdetermined least squares problem $\min_{\boldsymbol{x}} \| \boldsymbol{A}\boldsymbol{x} - \boldsymbol{y} \|_2$ using standard methods costs $O(KL^2)$. One approach to reducing this cost is

to randomly sample and rescale $J \ll K$ of the rows of $\boldsymbol{A}$ and $\boldsymbol{y}$ according to a carefully chosen probability distribution and then solve this smaller system instead. This will reduce the solution cost to $O(JL^2)$.

Definition 61. For a matrix $\boldsymbol{A} \in \mathbb{R}^{K \times L}$, let $\boldsymbol{U} \in \mathbb{R}^{K \times \operatorname{rank}(\boldsymbol{A})}$ contain the left singular vectors of $\boldsymbol{A}$. The ith **leverage score** of $\boldsymbol{A}$ is defined as $\ell_i(\boldsymbol{A}) \stackrel{\text{def}}{=} \|\boldsymbol{U}(i,:)\|_2^2$ for $i \in [K]$.

Definition 62. Let $\boldsymbol{q} \in \mathbb{R}^K$ be a probability distribution on $[K]$, and let $\boldsymbol{v} \in [K]^J$ be a random vector with independent elements satisfying $\mathbb{P}(\boldsymbol{v}(j) = i) = \boldsymbol{q}(i)$ for all $(i,j) \in [K] \times [J]$. Let $\boldsymbol{\Omega} \in \mathbb{R}^{J \times K}$ and $\boldsymbol{R} \in \mathbb{R}^{J \times J}$ be a random sampling matrix and a diagonal rescaling matrix, respectively, defined as $\boldsymbol{\Omega}(j,:) \stackrel{\text{def}}{=} \boldsymbol{e}_{\boldsymbol{v}(j)}^\top$ and $\boldsymbol{R}(j,j) \stackrel{\text{def}}{=} 1/\sqrt{J\boldsymbol{q}(\boldsymbol{v}(j))}$ for each $j \in [J]$, where $\boldsymbol{e}_i$ is the ith column of the $K \times K$ identity matrix. We say that $\boldsymbol{S} \stackrel{\text{def}}{=} \boldsymbol{R}\boldsymbol{\Omega} \in \mathbb{R}^{J \times K}$ is a **sampling matrix with parameters** $(J, \boldsymbol{q})$, or $\boldsymbol{S} \sim \mathcal{D}(J, \boldsymbol{q})$ for short. Let $\boldsymbol{A} \in \mathbb{R}^{K \times L}$ be a nonzero matrix, let $\boldsymbol{p} \in \mathbb{R}^K$ be a probability distribution on $[K]$ with entries $\boldsymbol{p}(i) \stackrel{\text{def}}{=} \ell_i(\boldsymbol{A})/\operatorname{rank}(\boldsymbol{A})$, and suppose $\beta \in (0,1]$. We say that $\boldsymbol{S} \sim \mathcal{D}(J, \boldsymbol{q})$ is a **leverage score sampling matrix for** $(\boldsymbol{A}, \beta)$ if $\boldsymbol{q}(i) \geq \beta \boldsymbol{p}(i)$ for all $i \in [K]$.

Notice that for any $\boldsymbol{A}$ we have $\sum_i \ell_i(\boldsymbol{A}) = \operatorname{rank}(\boldsymbol{A})$ and therefore $\sum_i \boldsymbol{p}(i) = 1$ as desired for a probability distribution. One can show that if $\boldsymbol{S} \sim \mathcal{D}(J, \boldsymbol{q})$ is a leverage score sampling matrix for $(\boldsymbol{A}, \beta)$ with J sufficiently large, then $\|\boldsymbol{A}\tilde{\boldsymbol{x}} - \boldsymbol{y}\|_2 \leq (1+\varepsilon) \min_{\boldsymbol{x}} \|\boldsymbol{A}\boldsymbol{x} - \boldsymbol{y}\|_2$ with high probability, where $\tilde{\boldsymbol{x}} \stackrel{\text{def}}{=} \arg\min_{\boldsymbol{x}} \|\boldsymbol{S}\boldsymbol{A}\boldsymbol{x} - \boldsymbol{S}\boldsymbol{y}\|_2$. How large to make J depends on the probability of success and the parameters ε and β. For further details on leverage score sampling, see Mahoney [138] and Woodruff [212].

7.4 Tensor ring decomposition via sampling

We propose using leverage score sampling to reduce the size of the least squares problem on line 5 in Algorithm 7. A challenge with using leverage score sampling is computing a sampling distribution $\boldsymbol{q}$ which satisfies the conditions in Definition 62. One option is to use $\boldsymbol{q} = \boldsymbol{p}$ and $\beta = 1$, with $\boldsymbol{p}$ defined as in Definition 62. However, computing $\boldsymbol{p}$ requires finding the left singular vectors of the design matrix which costs the same as solving the original least squares problem.

When the design matrix is $\boldsymbol{G}_{[2]}^{\neq n}$, a sampling distribution $\boldsymbol{q}$ can be computed much more efficiently directly from the cores $\mathcal{G}^{(1)}, \ldots, \mathcal{G}^{(n-1)}, \mathcal{G}^{(n+1)}, \ldots, \mathcal{G}^{(N)}$ without explicitly forming $\boldsymbol{G}_{[2]}^{\neq n}$. For each $n \in [N]$, let $\boldsymbol{p}^{(n)} \in \mathbb{R}^{I_n}$ be a probability distribution on $[I_n]$ defined elementwise via[2]

$$\boldsymbol{p}^{(n)}(i_n) \stackrel{\text{def}}{=} \frac{\ell_{i_n}(\boldsymbol{G}_{(2)}^{(n)})}{\operatorname{rank}(\boldsymbol{G}_{(2)}^{(n)})}. \tag{7.3}$$

Furthermore, let $\boldsymbol{q}^{\neq n} \in \mathbb{R}^{\prod_{j \neq n} I_j}$ be a vector defined elementwise via

$$\boldsymbol{q}^{\neq n}(\overline{i_{n+1} \cdots i_N i_1 \cdots i_{n-1}}) \stackrel{\text{def}}{=} \prod_{\substack{j=1 \\ j \neq n}}^{N} \boldsymbol{p}^{(j)}(i_j). \tag{7.4}$$

Lemma 63. *Let β_n be defined as*

$$\beta_n \stackrel{\text{def}}{=} \Big(R_{n-1} R_n \prod_{\substack{j=1 \\ j \notin \{n-1, n\}}}^{N} R_j^2 \Big)^{-1}.$$

For each $n \in [N]$, the vector $\boldsymbol{q}^{\neq n}$ is a probability distribution on $[\prod_{j \neq n} I_j]$, and $\boldsymbol{S} \sim \mathcal{D}(J, \boldsymbol{q}^{\neq n})$ is a leverage score sampling matrix for $(\boldsymbol{G}_{[2]}^{\neq n}, \beta_n)$.

This lemma can now be used to prove the following guarantees for a sampled variant of the least squares problem on line 5 of Algorithm 7.

Theorem 64. *Let $\boldsymbol{S} \sim \mathcal{D}(J, \boldsymbol{q}^{\neq n})$, $\varepsilon \in (0, 1)$, $\delta \in (0, 1)$ and $\tilde{\mathcal{Z}} \stackrel{\text{def}}{=} \arg\min_{\mathcal{Z}} \|\boldsymbol{S}\boldsymbol{G}_{[2]}^{\neq n} \boldsymbol{Z}_{(2)}^{\top} - \boldsymbol{S}\boldsymbol{X}_{[n]}^{\top}\|_{\mathrm{F}}$. If*

$$J > \Big(\prod_{j=1}^{N} R_j^2 \Big) \max\Big(\frac{16}{3(\sqrt{2}-1)^2} \ln\Big(\frac{4 R_{n-1} R_n}{\delta} \Big), \frac{4}{\varepsilon \delta} \Big), \tag{7.5}$$

then the following holds with probability at least $1 - \delta$:

$$\|\boldsymbol{G}_{[2]}^{\neq n} \tilde{\boldsymbol{Z}}_{(2)}^{\top} - \boldsymbol{X}_{[n]}^{\top}\|_{\mathrm{F}} \leq (1 + \varepsilon) \min_{\mathcal{Z}} \|\boldsymbol{G}_{[2]}^{\neq n} \boldsymbol{Z}_{(2)}^{\top} - \boldsymbol{X}_{[n]}^{\top}\|_{\mathrm{F}}.$$

Proofs of Lemma 63 and Theorem 64 are given in Section C.1 of the supplement. In Algorithm 8 we present our proposed TR decomposition algorithm.

We draw each core entry independently from a standard normal distribution during the initialization on line 1 in our experiments. The sample size on line 5 can be provided as a user

[2] $\operatorname{rank}(\boldsymbol{G}_{(2)}^{(n)}) \geq 1$ due to our assumption that $\mathcal{G}^{\neq n} \neq \mathbf{0}$ for all $n \in [N]$, so (7.3) is well-defined; see Remark 86 in the supplement.

Algorithm 8: TR-ALS-Sampled (proposal)

Input: $\mathcal{X} \in \mathbb{R}^{I_1 \times \cdots \times I_N}$, target ranks $(R_1, \ldots, R_N)$
Output: TR cores $\mathcal{G}^{(1)}, \ldots, \mathcal{G}^{(N)}$
1 Initialize cores $\mathcal{G}^{(2)}, \ldots, \mathcal{G}^{(N)}$
2 Compute distributions $\boldsymbol{p}^{(2)}, \ldots, \boldsymbol{p}^{(N)}$ via (7.3)
3 **repeat**
4 **for** $n = 1, \ldots, N$ **do**
5 Set sample size J
6 Draw sampling matrix $\boldsymbol{S} \sim \mathcal{D}(J, \boldsymbol{q}^{\neq n})$
7 Compute $\tilde{\boldsymbol{G}}_{[2]}^{\neq n} = \boldsymbol{S}\boldsymbol{G}_{[2]}^{\neq n}$ from cores
8 Compute $\tilde{\boldsymbol{X}}_{[n]}^{\top} = \boldsymbol{S}\boldsymbol{X}_{[n]}^{\top}$
9 Update $\mathcal{G}^{(n)} = \arg\min_{\mathcal{Z}} \|\tilde{\boldsymbol{G}}_{[2]}^{\neq n} \boldsymbol{Z}_{(2)}^{\top} - \tilde{\boldsymbol{X}}_{[n]}^{\top}\|_{\mathrm{F}}$
10 Update nth distribution $\boldsymbol{p}^{(n)}$ via (7.3)
11 **end**
12 **until termination criteria met**
13 **return** $\mathcal{G}^{(1)}, \ldots, \mathcal{G}^{(N)}$

input or be determined adaptively; this is discussed further in Section 7.4.2. The sampling matrix $\boldsymbol{S}$ is never explicitly constructed. Instead, on line 6, we draw and store a realization of the vector $\boldsymbol{v} \in [\prod_{j \neq n} I_j]^J$ in Definition 62. Entries of $\boldsymbol{v}$ can be drawn efficiently using Algorithm 9.

Algorithm 9: Sampling from $\boldsymbol{q}^{\neq n}$ (proposal)

Input: Distributions $\{\boldsymbol{p}^{(k)}\}_{k \neq n}$
Output: Sample $i \sim \boldsymbol{q}^{\neq n}$
1 **for** $j \in [N] \setminus n$ **do** draw realization $i_j \sim \boldsymbol{p}^{(j)}$
2 **return** $i = \overline{i_{n+1} \cdots i_N i_1 \cdots i_{n-1}}$

The sampling on line 7 in Algorithm 8 can be done efficiently directly from the cores without forming $\boldsymbol{G}_{[2]}^{\neq n}$. To see this, note that the matrix row $\boldsymbol{G}_{[2]}^{\neq n}(i, :)$ is the vectorization of the tensor slice $\mathcal{G}^{\neq n}(:, i, :)$ due to Definition 59. From Definition 60, this tensor slice in turn is given by the sequence of matrix multiplications

$$\mathcal{G}^{\neq n}(:, i, :) = \mathcal{G}^{(n+1)}(:, i_{n+1}, :) \cdots \mathcal{G}^{(N)}(:, i_N, :) \cdot \mathcal{G}^{(1)}(:, i_1, :) \cdots \mathcal{G}^{(n-1)}(:, i_{n-1}, :),$$

where $i = \overline{i_{n+1} \cdots i_N i_1 \cdots i_{n-1}}$ and each $\mathcal{G}^{(j)}(:, i_j, :)$ is treated as an $R_{j-1} \times R_j$ matrix. For a realization of $\boldsymbol{v} \in [\prod_{j \neq n} I_j]^J$, it is therefore sufficient to extract J lateral slices from each of the cores $\mathcal{G}^{(1)}, \ldots, \mathcal{G}^{(n-1)}, \mathcal{G}^{(n+1)}, \ldots, \mathcal{G}^{(N)}$ and then multiply together the jth slice from each core, for

each $j \in [J]$. This is illustrated in Figure 7.2. An algorithm for computing $\tilde{\boldsymbol{G}}_{[2]}^{\neq n}$ in this way is given in Algorithm 10.

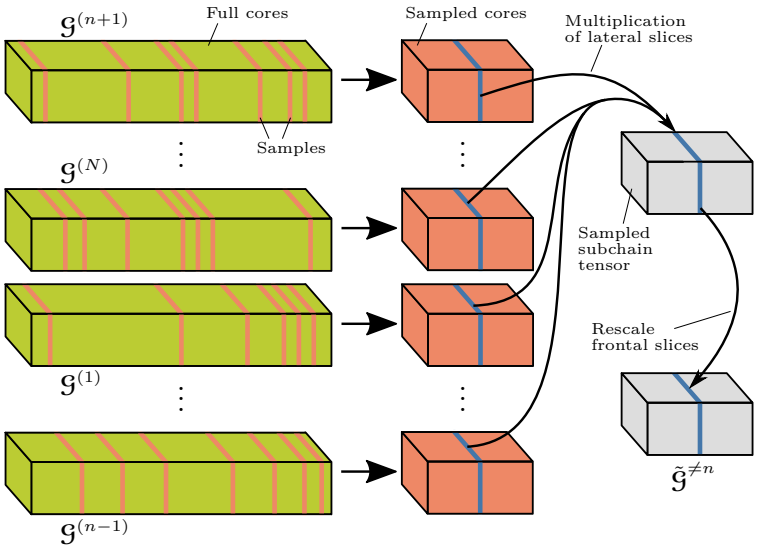

Figure 7.2: Illustration of how to efficiently construct $\tilde{\mathcal{G}}^{\neq n}$ by sampling the core tensors.

Algorithm 10: Computation of $\tilde{\boldsymbol{G}}_{[2]}^{\neq n}$ (proposal)

Input: Samples $\boldsymbol{v}$, distributions $\{\boldsymbol{p}^{(k)}\}_{k \neq n}$
Output: Sketch $\tilde{\boldsymbol{G}}_{[2]}^{\neq n} = \boldsymbol{S}\boldsymbol{G}_{[2]}^{\neq n}$

1 Initialize $\tilde{\mathcal{G}}^{\neq n} = \mathbf{0} \in \mathbb{R}^{R_n \times J \times R_{n-1}}$
2 **for** $j \in [J]$ **do**
3 $\quad i = \boldsymbol{v}(j)$ // Where $i = \overline{i_{n+1} \cdots i_N i_1 \cdots i_{n-1}}$
4 $\quad c = \sqrt{J \prod_{k \neq n} \boldsymbol{p}(i_k)}$
5 $\quad \tilde{\mathcal{G}}^{\neq n}(:, j, :) = \mathcal{G}^{(n+1)}(:, i_{n+1}, :) \cdots \mathcal{G}^{(N)}(:, i_N, :)$
6 $\qquad \cdot \mathcal{G}^{(1)}(:, i_1, :) \cdots \mathcal{G}^{(n-1)}(:, i_{n-1}, :)/c$
7 **end**
8 **return** $\tilde{\boldsymbol{G}}_{[2]}^{\neq n}$

The sampling of the data tensor on line 8 in Algorithm 8 only needs to access JI_n entries of $\mathcal{X}$, or less if $\mathcal{X}$ is sparse. This is why our proposed algorithm has a cost which is sublinear in the number of entries in $\mathcal{X}$. Due to Theorem 64, each sampled least squares solve on line 9 has high-probability relative-error guarantees, provided J is large enough.

7.4.1 Termination criteria

Both Algorithms 7 and 8 rely on using some termination criterion. One such criterion is to terminate the outer loop when the relative error $\|\operatorname{TR}((\mathcal{G}^{(n)})_{n=1}^{N}) - \mathcal{X}\|_{\mathrm{F}}/\|\mathcal{X}\|_{\mathrm{F}}$, or its change, is below some threshold. Computing the relative error exactly costs $O(I^N R^2)$ which may be too costly for large tensors. In particular, it would defeat the purpose of the sublinear complexity of our method. An alternative is to terminate when the change in $\|\operatorname{TR}((\mathcal{G}^{(n)})_{n=1}^{N})\|_{\mathrm{F}}$ is below some threshold. Such an approach is used for large scale Tucker decomposition by Kolda and Sun [117] and Malik and Becker [142]. The norm $\|\operatorname{TR}((\mathcal{G}^{(n)})_{n=1}^{N})\|_{\mathrm{F}}$ can be computed efficiently using an algorithm by Mickelin and Karaman [153]. When $I_n = I$ and $R_n = R$ for all $n \in [N]$ it has complexity $O(NIR^4)$; see Section 3.3 of Mickelin and Karaman [153]. Another approach is to estimate the relative error via sampling, which is used by Battaglino et al. [15] for their randomized CP decomposition method.

7.4.2 Adaptive sample size

Algorithm 8 can produce good results even if the sample size J is smaller than what Theorem 64 suggests. Choosing an appropriate J for a particular problem can therefore be challenging. Aggour et al. [1] develop an ALS-based algorithm for CP decomposition with an adaptive sketching rate to speed up convergence. Similar ideas can easily be incorporated in our Algorithm 8. Indeed, one of the benefits of our method is that the sample size J can be changed at any point with no overhead cost. By contrast, for rTR-ALS, another randomized method for TR decomposition which we describe in Section 7.5, the entire compression phase would need to be redone if the sketch rate is updated.

7.4.3 Complexity analysis

Table 7.1 compares the leading order complexity of our proposed method to the four other methods we consider in the experiments. The other methods are described in Section 7.5. For simplicity, we assume that $I_n = I$ and $R_n = R$ for all $n \in [N]$, and that $N < I$, $R^2 < I$ and

Table 7.1: Comparison of leading order computational complexity. "#iter" denotes the number of outer loop iterations in the ALS-based methods.

Method	Complexity
TR-ALS	$NIR^2 + \#\text{iter} \cdot NI^N R^2$
rTR-ALS	$NI^N K + \#\text{iter} \cdot NK^N R^2$
TR-SVD	$I^{N+1} + I^N R^3$
TR-SVD-Rand	$I^N R^2$
TR-ALS-S. (proposal)	$NIR^4 + \#\text{iter} \cdot \frac{NIR^{2(N+1)}}{\varepsilon\delta}$

$NR < I$. For rTR-ALS, the intermediate N-way Tucker core is assumed to be of size $K \times \cdots \times K$; see Yuan et al. [220] for details. The complexity for our method in the table assumes J satisfies (7.5), and that ε and δ are small enough so that (7.5) simplifies to $J > 4R^{2N}/(\varepsilon\delta)$. Any costs for checking convergence criteria are ignored. This is justified since e.g. the termination criterion based on computing $\| \operatorname{TR}((\mathcal{G}^{(n)})_{n=1}^N) \|_{\mathrm{F}}$ described in Section 7.4.1 would not impact the complexity of TR-ALS or our TR-ALS-Sampled, and would only impact that of rTR-ALS if $K^N < IR^2$. The complexities in Table 7.1 are derived in Section C.2 of the supplement.

The complexity of our method is sublinear in the number of tensor elements, avoiding the dependence on I^N that the other methods have. While there is still an exponential dependence on N, this is still much better than I^N for low-rank decomposition of large tensors in which case $R \ll I$. This is particularly true when the input tensor is too large to store in RAM and accessing its elements dominate the cost of the decomposition. We note that our method typically works well in practice even if (7.5) is not satisfied.

7.5 Experiments

7.5.1 Decomposition of real and synthetic datasets

We compare our proposed TR-ALS-Sampled method (Alg. 8) to four other methods: TR-ALS (Alg. 7), rTR-ALS (Alg. 1 in Yuan et al. [220]), TR-SVD (Alg. 1 in Mickelin and Karaman [153]) and a randomized variant of TR-SVD (Alg. 7 in Ahmadi-Asl et al. [2]) which we refer to as TR-SVD-Rand. TR-SVD takes an error tolerance as input and makes the ranks large enough to achieve

this tolerance. To facilitate comparison to the ALS-based methods, we modify TR-SVD so that it takes target ranks as inputs instead. TR-SVD-Rand takes target ranks as inputs by default. For TR-ALS, rTR-ALS and TR-SVD-Rand, we wrote our own implementations since codes were not publicly available. For TR-SVD, we modified the function `TRdecomp.m` provided by Mickelin and Karaman [153][3] . As suggested by Ahmadi-Asl et al. [2], we use an oversampling parameter of 10 in TR-SVD-Rand. The relative error of a decomposition is computed as $\|\operatorname{TR}((\mathcal{G}^{(n)})_{n=1}^{N}) - \mathcal{X}\|_{\mathrm{F}}/\|\mathcal{X}\|_{\mathrm{F}}$.

To get a fair comparison between the different methods we first decompose the input tensor with TR-ALS by running it until a convergence criterion has been met or a maximum number of iterations reached. All ALS-based algorithms (including TR-ALS) are then applied to the input tensor, running for twice as many iterations as it took the initial TR-ALS run to terminate, without checking any convergence criteria. This is done to strip away any run time influence that checking convergence has and only compare those aspects that differ between the methods. TR-ALS-Sampled and rTR-ALS apply sketching in fundamentally different ways. To best understand the trade-off between run time and accuracy, we start out these two algorithm with small sketch rates (J and K, respectively) and increase them until the resulting relative error is less than $(1+\varepsilon)E$, where E is the relative error achieved by TR-ALS during the second run and $\varepsilon \in (0, 1)$. The relative error and run time for the smallest sketch rate that achieve this relative-error bound are then reported. The exact details on how this is done for each individual experiment is described in Section C.4 of the supplement.

All experiments are run in Matlab R2017a with Tensor Toolbox 2.6 [11] on a computer with an Intel Xeon E5-2630 v3 @ 32x 3.2GHz CPU and 32 GB of RAM. All our code used in the experiments is available online[4] .

[3] Available at `https://github.com/oscarmickelin/tensor-ring-decomposition`.
[4] Available at `https://github.com/OsmanMalik/tr-als-sampled`.

7.5.1.1 Randomly generated data

Synthetic 3-way tensors are generated by creating 3 cores of size $R' \times I \times R'$ with entries drawn independently from a standard normal distribution. Here, R' is used to denote true rank, as opposed to the rank R used in the decomposition. For each core, one entry is chosen uniformly at random and set to 20. The goal of this is to increase the coherence in the least squares problems, which makes them more challenging for sampling-based methods like ours. The cores are then combined into a full tensor via (7.1). Finally, Gaussian noise with standard deviation 0.1 is added to all entries in the tensor independently.

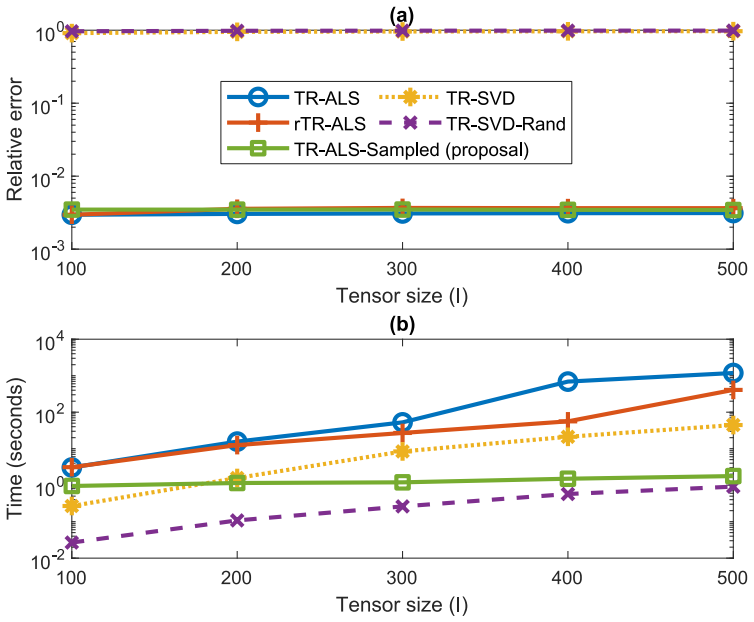

Figure 7.3: Synthetic experiment with true and target ranks $R' = R = 10$. (a) Relative error and (b) run time for each of the five methods. Average number of ALS iterations used is 21.

In the first synthetic experiment $R' = 10$ and $I \in \{100, 200, \ldots, 500\}$. The target rank is also set to $R = 10$. Ideally, decompositions should therefore have low error. Figure 7.3 shows the relative error and run time for the five different algorithms. All plotted quantities are the median over 10 runs. The ALS-based algorithms reach very low errors. On average, only 21 iterations are needed, and the number of iterations is fairly stable when the tensor size is increased. The SVD-based algorithms, by contrast do very poorly, having relative errors close to 1. Our proposed

TR-ALS-Sampled is the fastest method, aside from TR-SVD-Rand which returns very poor results. We achieve a substantial speedup of 668× over TR-ALS and 230× over rTR-ALS for $I = 500$.

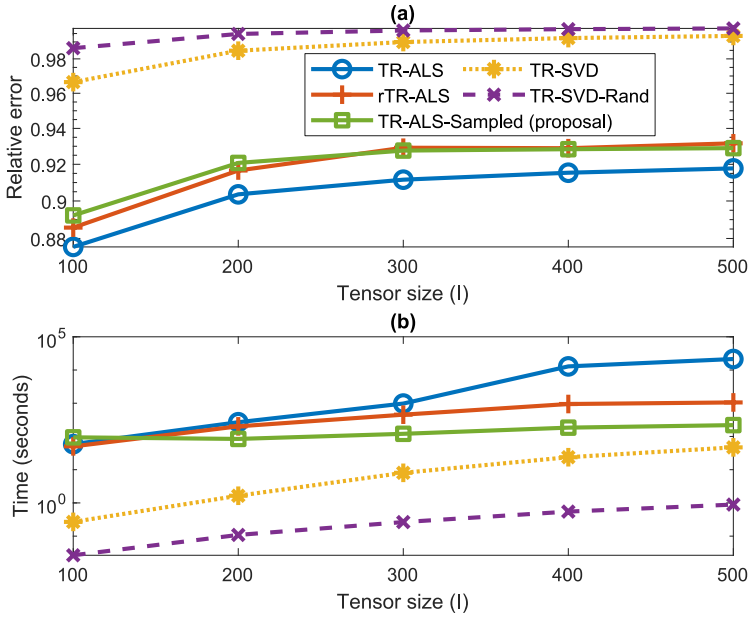

Figure 7.4: Synthetic experiment with true rank $R' = 20$ and target rank $R = 10$. (a) Relative error and (b) run time for each of the five methods. Average number of ALS iterations used is 417.

In the second synthetic experiment, we set $R' = 20$ but keep the target rank at $R = 10$. We therefore expect the decompositions to have a larger relative error. Figure 7.4 shows the results for these experiments. All plotted quantities are the median over 10 runs. The ALS-based methods still outperform the SVD-based ones. The ALS-based methods now require more iterations, on average running for 417 iterations, with greater variability between runs. Our proposed TR-ALS-Sampled remains faster than the other ALS-based methods, although with a smaller difference, achieving a speedup of 98× over TR-ALS and 5× over rTR-ALS for $I = 500$. Although the SVD-based methods are faster, especially the randomized variant, they are not useful since they give such a high error. See Remarks 89 and 90 in the supplement for further discussion of the results in Figures 7.3 and 7.4.

The signal-to-noise ratio (SNR) is about 50 dB and 59 dB, respectively, in the two experiments above. To validate the robustness of those results to higher levels of noise we run additional experiments on synthetic $300 \times 300 \times 300$ tensors with added Gaussian noise with standard deviation

Table 7.2: Decomposition results for highly oscillatory functions with target rank $R = 10$.

	Linear Growth		Airy		Chirp	
Method	Error	Time (s)	Error	Time (s)	Error	Time (s)
TR-ALS	0.010	370.3	0.020	531.7	0.020	613.1
rTR-ALS	0.011	382.8	0.019	521.6	0.019	645.9
TR-ALS-Sampled (proposal)	0.011	3.4	0.021	2.7	0.021	36.2

1 and 10 which corresponds to an SNR of 30 dB and 10 dB, respectively. All other settings are kept the same as in the first synthetic experiment. The results, shown in Figure 7.5, indicate that the earlier experiment results are robust to higher noise levels.

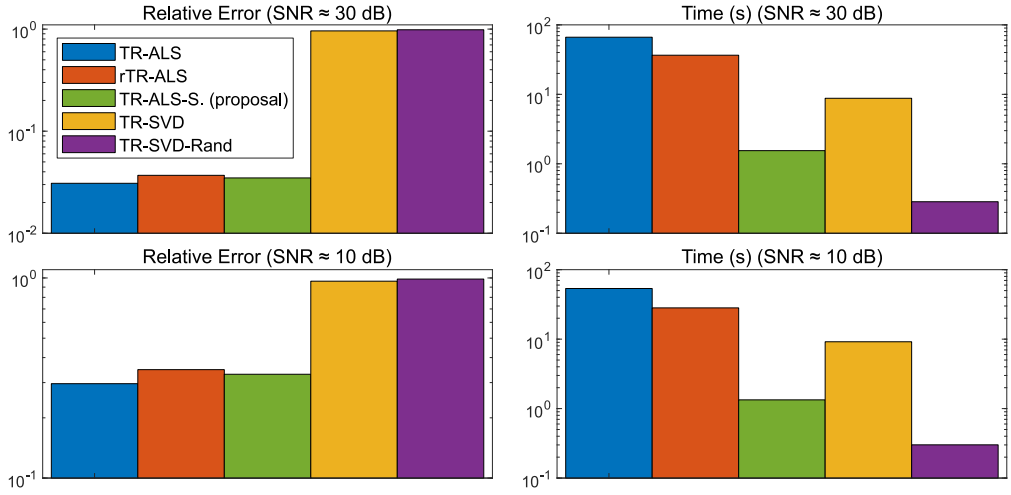

Figure 7.5: Comparison of methods for higher levels of noise.

7.5.1.2 Highly oscillatory functions

Next, we decompose the highly oscillatory functions considered by Zhao et al. [226]. The functions are illustrate in Figure 7.6. Each function is evaluated at 4^{10} points. These values are then reshaped into 10-way tensors of size $4 \times \cdots \times 4$ which are then decomposed. Table 7.2 shows the results for when all target ranks are $R = 10$. The SVD-based methods can not be applied in this experiment since they require $R_0 R_1 \leq I_1$ which is not satisfied.

All algorithms achieve a relative error of only 1%–2%. This is impressive, especially considering

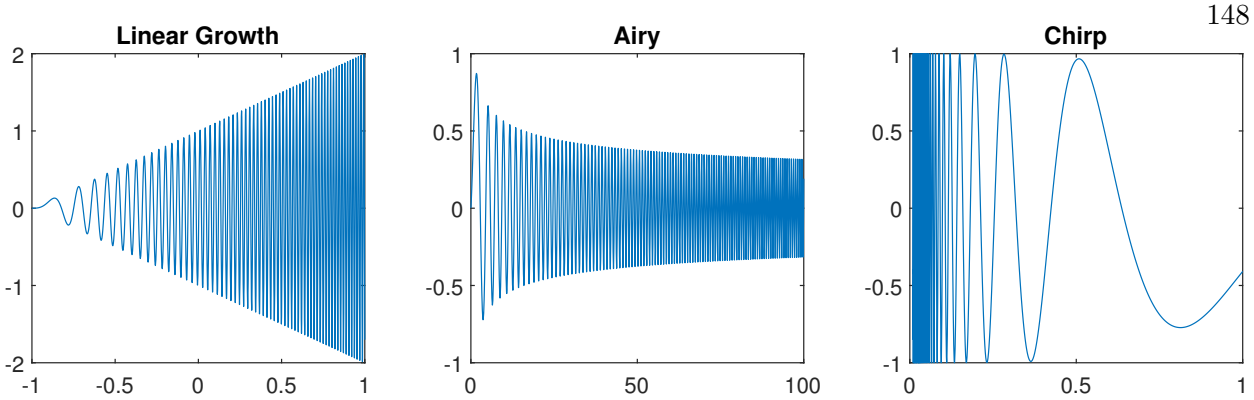

Figure 7.6: The functions are defined as follows. Linear growth: $(x+1)\sin(100(x+1)^2)$. Airy: $x^{-1/4}\sin(2x^{3/2}/3)$. Chirp: $\sin(4/x)\cos(x^2)$.

that the choice of $R_n = 10$ correspond to a 262× compression rate. With an error very similar to that of TR-ALS, our proposal TR-ALS-Sampled achieves speedups of 108×, 193× and 17× on the three tensors over TR-ALS. Our method achieves speedups of 112×, 190× and 18× over rTR-ALS. Notice that rTR-ALS yields no speedup over TR-ALS in Table 7.2. The reason for this is that each $I_n = 4$ is so small. For rTR-ALS to have acceptable accuracy, K must be chosen to be $K = 4$, meaning that rTR-ALS is just as slow as TR-ALS.

7.5.1.3 Image and video data

We consider five real image and video datasets[5] (sizes in parentheses): Pavia University ($610 \times 340 \times 103$) and DC Mall ($1280 \times 307 \times 191$) are 3-way tensors containing hyperspectral images. The first two dimensions are the image height and width, and the third dimension is the number of spectral bands. Park Bench ($1080 \times 1920 \times 364$) and Tabby Cat ($720 \times 1280 \times 286$) are 3-way tensors representing grayscale videos of a man sitting on a park bench and a cat, respectively. The first two dimension are frame height and width, and the third dimension is the number of frames. Red Truck ($128 \times 128 \times 3 \times 72$) is a 4-way tensor consisting of 72 color images of size 128 by 128 pixels depicting a red truck from different angles. It is a subset of the COIL-100 dataset [158].

Tables 7.3 and 7.4 show results for when all target ranks are $R = 10$ and $R = 20$, respectively.

[5] Links to the datasets are provided in Section C.3 of the supplement.

Table 7.3: Decomposition results for real datasets with target rank $R = 10$. Time is in seconds.

	Pavia Uni.		DC Mall		Park Bench		Tabby Cat		Red Truck	
Method	Error	Time	Error	Time	Error	Time	Error	Time	Error	Time
TR-ALS	0.16	62.1	0.16	801.4	0.07	5384.9	0.14	1769.4	0.18	181.2
rTR-ALS	0.18	16.3	0.17	25.4	0.08	64.8	0.14	9.0	0.19	22.0
TR-ALS-S. (proposal)	0.18	1.7	0.18	3.1	0.08	3.5	0.15	2.4	0.19	1.9
TR-SVD	0.21	2.1	0.20	12.8	0.08	136.5	0.15	37.7	0.26	1.3
TR-SVD-Rand	0.33	0.2	0.30	0.3	0.11	2.8	0.20	1.6	0.29	0.1

Table 7.4: Decomposition results for real datasets with target rank $R = 20$. The ✗ signifies that the SVD-based methods cannot handle this case since they require $R_0 R_1 \leq I_1$. Time is in seconds.

	Pavia Uni.		DC Mall		Park Bench		Tabby Cat		Red Truck	
Method	Error	Time	Error	Time	Error	Time	Error	Time	Error	Time
TR-ALS	0.06	249.9	0.06	622.8	0.04	2385.0	0.11	1397.2	0.11	1279.1
rTR-ALS	0.07	194.6	0.06	349.6	0.05	531.5	0.12	92.6	0.12	470.6
TR-ALS-S. (proposal)	0.07	13.7	0.06	26.5	0.05	25.0	0.12	19.0	0.12	22.6
TR-SVD	0.10	4.6	0.10	25.4	0.05	272.0	0.13	110.0	✗	✗
TR-SVD-Rand	0.28	0.5	0.25	1.0	0.10	8.5	0.17	4.1	✗	✗

The SVD-based algorithms require $R_0 R_1 \leq I_1$, so they cannot handle the Red Truck tensor when $R = 20$ since $I_1 = 128$ for that dataset. All results are based on a single trial. Our proposed TR-ALS-Sampled runs faster than the other ALS-based methods on all image and video datasets, achieving up to 1518× speedup over TR-ALS (for the Park Bench tensor with $R = 10$) and a 21× speedup over rTR-ALS (for the Red Truck tensor with $R = 20$). The SVD-based methods perform better than they did on the randomly generated data, but remain worse than the ALS-based methods. TR-SVD-Rand is always the fastest method, but also has a substantially higher error.

A popular preprocessing step used in tensor decomposition is to first reshape a tensor into a tensor with more modes. Next, we therefore try reshaping the five tensors above so that they each have twice as many modes. Some of the dimensions are truncated slightly to allow for this reshaping. Our method yields speedups in the range 94×–2880× over TR-ALS and 73×–1484× over rTR-ALS. The SVD-based methods cannot be applied since they require $R_0 R_1 \leq I_1$ which is not satisfied for these reshaped tensors. The speed benefit of our method appears to be **even greater**

when increasing the number of modes. Please see Section C.4.3 of the supplement for further details on this experiment, including detailed results.

An alternative to sampling according to the distribution defined by the leverage scores in (7.3) and (7.4) is to simply sample according to the uniform distribution, i.e., $\boldsymbol{q}^{\neq n} = 1/\prod_{j\neq n} I_j$. Although such a sampling approach does not come with any guarantees, it is faster than leverage score sampling since it avoids the cost of computing the sampling distribution. In order to investigate the performance of uniform sampling, we repeat the experiments whose results are show in Tables 7.3 and 7.4 for TR-ALS-Sampled, but using uniform sampling instead of leverage score sampling. Everything else is kept the same. Table 7.5 shows the results.

Table 7.5: Decomposition results for TR-ALS-Sampled with **uniform sampling**.

	$R = 10$		$R = 20$	
Dataset	Error	Time (s)	Error	Time (s)
Pavia Uni.	0.18	1.3	0.07	9.0
DC Mall	0.18	3.7	0.06	22.2
Park Bench	0.08	5.8	0.05	28.0
Tabby Cat	0.15	2.2	0.12	23.1
Red Truck	0.20	1.5	0.12	18.7

Using uniform instead of leverage score sampling does not appear to impact the error and overall run time much. However, uniform sampling requires 4.6–8.8 times as many samples to reach the target accuracy. Although more samples increase the cost of the least squares solve, the cost of updating the sampling distribution is avoided. It may be possible to combine uniform and leverage score sampling to speed up our method while still retaining some guarantees. Uniform sampling has also yielded promising empirical results for the CP decomposition; see e.g. Battaglino et al. [15] and Aggour et al. [1].

7.5.2 Rapid feature extraction for classification

Inspired by experiments in Zhao et al. [226], we now give a practical example of how our method can be used for rapid feature extraction for classification. We consider the full COIL-100

dataset which consists of 7200 color images of size 128×128 depicting 100 different objects from 72 different angles each. The images are downsampled to 32×32 and stacked into a tensor $\mathcal{X}$ of size $32 \times 32 \times 3 \times 7200$. We then use the various TR decomposition algorithms to compute a TR decomposition of $\mathcal{X}$ with each rank $R_n = 5$. Additional details on algorithm settings are given in Section C.4.4 of the supplement. The TR core $\mathcal{G}^{(4)}$ is of size $5 \times 7200 \times 5$ and contains latent features for the 4th mode of $\mathcal{X}$. We reshape it into a 7200×25 feature matrix and apply the k-nearest neighbor algorithm with $k = 1$. Table 7.6 reports the decomposition error as well as the accuracy achieved when classifying the 7200 images into the 100 different classes. The classification is done using 10-fold cross validation. The ALS-based methods yield a lower error than the SVD-based methods. All methods result in a similar classification accuracy. The higher decomposition errors for the SVD-based methods do not seem to compromise classification accuracy.

Table 7.6: Decomposition error and classification accuracy for rapid feature extraction experiment.

Method	Time (s)	Error	Acc. (%)
TR-ALS	248.7	0.27	99.07
rTR-ALS	3.1	0.29	98.96
TR-ALS-S. (proposal)	12.7	0.28	99.32
TR-SVD	7.3	0.37	99.68
TR-SVD-Rand	0.7	0.33	99.85

7.6 Conclusion

We have proposed a method for TR decomposition which uses leverage score sampled ALS and has complexity sublinear in the number of input tensor entries. We also proved high-probability relative-error guarantees for the sampled least squares problems. Our method achieved substantial speedup over competing methods in experiments on both synthetic and real data, in some cases by as much as two or three orders of magnitude, while maintaining good accuracy. We also demonstrated how our method can be used for rapid feature extraction.

The datasets in our experiments are dense. Based on limited testing, rTR-ALS seems to do particularly well on sparse datasets, outperforming our proposed method. Our method still yielded

a substantial speedup over TR-ALS. Further investigating this, and optimizing the various methods for sparse tensors, is an interesting direction for future research.

Chapter 8

Binary matrix factorization on special purpose hardware

8.1 Introduction

Many fundamental problems in data mining consist of discrete decision making and are combinatorial in nature. Examples include feature selection, data categorization, class assignment, identification of outlier instances, k-means clustering, combinatorial extensions of support vector machines, and consistent biclustering, to mention a few [172]. In many cases, these underlying problems are NP-hard, and approaches to solving them therefore dependent on heuristics. Recently, researchers have been exploring different computing paradigms to tackle these NP-hard problems, including quantum computing and the development of dedicated special purpose hardware. The Ising and QUBO models are now becoming unifying frameworks for the development of these novel types of hardware for combinatorial optimization problems.

Binary matrix factorization is a combinatorial problem that has been used for a wide range of applications in data mining and other areas, including clustering [154, 223, 224], pattern discovery [136, 186], dictionary learning [176], collaborative filtering [177], association rule mining [119], dimensionality reduction [13], and image rendering [120]. In this paper we show how the aforementioned hardware technologies, via the QUBO framework, can be used for binary matrix factorization. We make the following contributions:

- Provide two QUBO formulations for one variant of the binary matrix factorization problem.
- Show how constraints that are useful in clustering tasks can easily be incorporated into the

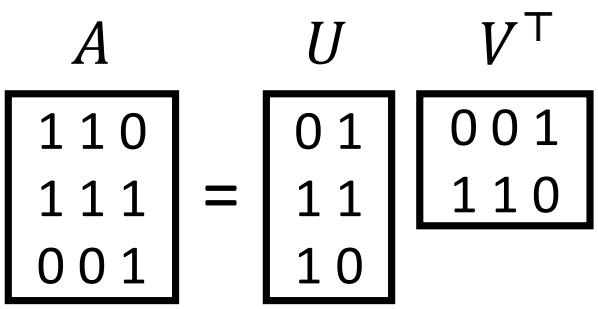

Figure 8.1: Example of an exact BMF.

QUBO formulations.

- Present a sampling heuristic for factorizing large rectangular matrices.
- Conduct experiments on both synthetic and real data on the Fujitsu Digital Annealer.

8.1.1 Binary matrix factorization

Let $\boldsymbol{A} \in \{0,1\}^{m \times n}$ be a matrix with binary entries. For a positive integer $r \leq \min(m,n)$, the rank-r binary matrix factorization (BMF) problem is

$$\begin{aligned} \min_{\boldsymbol{U},\boldsymbol{V}} \quad & \|\boldsymbol{A} - \boldsymbol{U}\boldsymbol{V}^\top\|_{\mathrm{F}}^2 \\ \text{s.t.} \quad & \boldsymbol{U} \in \{0,1\}^{m \times r},\ \boldsymbol{V} \in \{0,1\}^{n \times r}. \end{aligned} \tag{8.1}$$

Figure 8.1 shows a simple example of an exact BMF. We discuss other definitions of BMF that appear in the literature in Section 8.2.

8.1.2 The QUBO framework

Let $\boldsymbol{Q} \in \mathbb{R}^{n \times n}$ be a matrix. A Quadratic Unconstrained Binary Optimization (QUBO) problem takes the form

$$\min_{\boldsymbol{x}} \ \boldsymbol{x}^\top \boldsymbol{Q} \boldsymbol{x} \quad \text{s.t. } \boldsymbol{x} \in \{0,1\}^n. \tag{8.2}$$

The tutorial by Glover et al. [82] is a good introduction to this problem that also discusses some of its many applications.

8.1.3 The Digital Annealer

The Fujitsu Digital Annealer (DA) is a hardware accelerator for solving fully connected QUBO problems. Internally the hardware runs a modified version of the Metropolis–Hastings algorithm [92, 152] for simulated annealing. The hardware utilizes massive parallelization and a novel sampling technique. The novel sampling technique speeds up the traditional Markov Chain Monte Carlo (MCMC) method by almost always moving to a new state instead of being stuck in a local minimum. As explained in [5], in the DA, each Monte Carlo step takes the same amount of time, regardless of accepting a flip or not. In addition, when accepting the flip, the computational complexity of updating the effective fields is constant regardless of the connectivity of the graph. The DA also supports Parallel Tempering (replica exchange MCMC sampling) [195] which improves dynamic properties of the Monte Carlo method. In our experiments, we use the DA coupled with software techniques as our main QUBO solver.

8.1.4 Notation

Bold upper case letters (e.g. $\boldsymbol{A}$) denote matrices, bold lower case letters (e.g. $\boldsymbol{x}$) denote vectors, and lower case regular and Greek letters (e.g. x, λ) denote scalars. Subscripts are used to indicate entries in matrices and vectors. For example, a_{ij} is the entry on position (i, j) in $\boldsymbol{A}$. A $*$ in a subscript is used to denote all entries along a dimension. For example, $\boldsymbol{a}_{i*}$ and $\boldsymbol{a}_{*j}$ are the ith row and jth column of $\boldsymbol{A}$, respectively. We use $\mathbf{1}$, $\mathbf{0}$ and $\boldsymbol{I}$ to denote a matrix of ones, a matrix of zeros, and the identity matrix, respectively. Subscripts are also used to indicate the size of these matrices. For example, $\mathbf{1}_{m\times n}$ is an $m \times n$ matrix of all ones, $\mathbf{1}_n \stackrel{\text{def}}{=} \mathbf{1}_{n\times n}$ is an $n \times n$ matrix of all ones, and $\boldsymbol{I}_n$ is the $n \times n$ identity matrix. These subscripts are omitted when the size is obvious. Superscripts in parentheses will be used to number matrices and vector (e.g. $\boldsymbol{A}^{(1)}$, $\boldsymbol{A}^{(2)}$). The matrix Kronecker product is denoted by $\otimes$. The function vec$(\cdot)$ takes a matrix and turns it into a vector by stacking all its columns into one long column vector. The function diag$(\cdot)$ takes a vector input and returns a diagonal matrix with that vector along the diagonal. Semicolon is used as in Matlab to denote

vertical concatenation of vectors. For example, if $\boldsymbol{u} \in \mathbb{R}^m$ and $\boldsymbol{v} \in \mathbb{R}^n$ are column vectors, then $[\boldsymbol{u};\ \boldsymbol{v}]$ is a column vector of length $m+n$. The Frobenius norm of a matrix $\boldsymbol{A}$ is defined as

$$\|\boldsymbol{A}\|_{\mathrm{F}} \stackrel{\mathrm{def}}{=} \sqrt{\sum_{ij} a_{ij}^2}.$$

For positive integers n, we use the notation $[n] \stackrel{\mathrm{def}}{=} \{1, 2, \ldots, n\}$.

8.2 Related work

The most popular methods for BMF are the two by Zhang et al. [223, 224]. Their first approach alternates between updating $\boldsymbol{U}$ and $\boldsymbol{V}$ until some convergence criteria is met. It incorporates a penalty which encourages the entries of $\boldsymbol{U}$ and $\boldsymbol{V}$ to be near 0 or 1. At the end of the algorithm, the entries of $\boldsymbol{U}$ and $\boldsymbol{V}$ are rounded to ensure they are exactly 0 or 1. Their second approach initializes $\boldsymbol{U}$ and $\boldsymbol{V}$ using nonnegative matrix factorization. For each factor matrix, a threshold is then identified, and values in the matrix below and above the threshold are rounded to 0 and 1, respectively.

Koyutürk and Grama [119] develop a framework called PROXIMUS, which decomposes binary matrices by recursively using rank-1 approximations, which results in a hierarchical representation. Shen et al. [186] provide a linear program formulation for the rank-1 BMF problem and provide approximation guarantees. Ramírez [176] presents methods for BMF applied to binary dictionary learning. Kumar et al. [120] provide faster approximation algorithms for BMF as well as a variant of BMF for which inner products are computed over GF(2). Diop et al. [66] propose a variant of BMF for binary matrices which takes the form $\boldsymbol{A} \approx \Phi(\boldsymbol{U}\boldsymbol{V}^\top)$, where $\boldsymbol{U}$ and $\boldsymbol{V}$ are binary, and Φ is a nonlinear sigmoid function. They use a variant of the penalty approach by [223, 224] to compute the decomposition.

Boolean matrix decomposition, which is also referred to as binary matrix decomposition by some authors, is similar to the BMF in (8.1), but an element $(\boldsymbol{U}\boldsymbol{V}^\top)_{ij}$ is computed via

$$(\boldsymbol{U}\boldsymbol{V}^\top)_{ij} = \bigvee_{k=1}^{r} u_{ik} v_{jk}$$

instead of the standard inner product, where $\bigvee$ denotes disjuction. Some works that consider Boolean matrix decomposition include [13, 94, 118, 131, 136, 154, 155, 177]. For a theoretical comparison between BMF, Boolean matrix factorization and a variant of BMF computed over GF(2), we refer the reader to the recent paper by DeSantis et al. [60].

There are previous works that use special purpose hardware to solve linear algebra problems. O'Malley and Vesselinov [164] discuss how linear least squares can be solved via QUBO formulations on D-Wave quantum annealing machines. They consider both the case when the solution vector is restricted to being binary and when it is real valued. O'Malley et al. [165] consider a nonnegative/binary factorization of a real valued matrix of the form $\boldsymbol{A} \approx \boldsymbol{W}\boldsymbol{H}$, where $\boldsymbol{W}$ has nonnegative entries and $\boldsymbol{H}$ is binary. To compute this factorization, they use an alternating least squares approach by iteratively alternating between solving for $\boldsymbol{W}$ and $\boldsymbol{H}$. When solving for $\boldsymbol{H}$, they use a QUBO formulation for the corresponding binary least squares problem and do the computation on a D-Wave quantum annealer. Ottaviani and Amendola [169] propose a QUBO formulation for low-rank nonnegative matrix factorization and also implement it on a D-Wave machine. Borle et al. [30] show how the Quantum Approximate Optimization Algorithm (QAOA) framework can be used for solving binary linear least squares. Their paper includes experiments run on an IBM Q machine.

There has also been a large body of research on utilizing special purpose hardware for data clustering problems, for example [16, 51, 157, 185, 202] to name a few. A recent paper by Şeker et al. [74] performs a comprehensive computational study comparing the DA to multiple state-of-the-art solvers on multiple different combinatorial optimization problems. They find that the DA performs favorably compared to the other solvers, particularly on large problem instances.

8.3 QUBO formulations for BMF

Writing out the objective in (8.1), we get

$$\|\boldsymbol{A} - \boldsymbol{U}\boldsymbol{V}^\top\|_{\mathrm{F}}^2 = \|\boldsymbol{A}\|_{\mathrm{F}}^2 - 2\sum_{ijk} a_{ij}u_{ik}v_{jk} + \sum_{ijkk'} u_{ik}u_{ik'}v_{jk}v_{jk'}, \tag{8.3}$$

where the summations are over $i \in [m]$, $j \in [n]$, and $k, k' \in [r]$. Our goal is to reformulate this into the QUBO form in (8.2). The fourth order term in (8.3) stops us from directly writing (8.3) on the quadratic form in (8.2). We can get around this by introducing appropriate auxiliary variables and penalties.

8.3.1 Formulation 1

For our first formulation, we introduce auxiliary variables

$$w_{ij}^{(k)} \stackrel{\text{def}}{=} u_{ik} v_{jk} \quad \text{for } i \in [m],\ j \in [n],\ k \in [r]. \tag{8.4}$$

We arrange these variables into $m \times n$ matrices $\boldsymbol{W}^{(k)} = (w_{ij}^{(k)})$. An equivalent formulation to (8.1) is then

$$\begin{aligned} \min_{\boldsymbol{U},\boldsymbol{V},\{\boldsymbol{W}^{(k)}\}} \ & \|\boldsymbol{A}\|_{\mathrm{F}}^2 - 2\sum_{ijk} a_{ij} w_{ij}^{(k)} + \sum_{ijkk'} w_{ij}^{(k)} w_{ij}^{(k')} \\ \text{s.t. } & \boldsymbol{U} \in \{0,1\}^{m\times r},\ \boldsymbol{V} \in \{0,1\}^{n\times r},\ (8.4) \text{ satisfied.} \end{aligned} \tag{8.5}$$

To incorporate the constraints (8.4) in the QUBO model, we express them as a penalty instead. A standard technique for this [82] is to use a penalty function $f : \{0,1\}^3 \to \mathbb{R}$ defined via

$$f(a,b,c) \stackrel{\text{def}}{=} bc - 2ba - 2ca + 3a. \tag{8.6}$$

Notice that $f(a,b,c) = 0$ if $a = bc$ and $f(a,b,c) \geq 1$ otherwise. Letting λ be a positive constant, a penalty variant of (8.5) is

$$\begin{aligned} \min_{\boldsymbol{U},\boldsymbol{V},\{\boldsymbol{W}^{(k)}\}} \ & \|\boldsymbol{A}\|_{\mathrm{F}}^2 - 2\sum_{ijk} a_{ij} w_{ij}^{(k)} + \sum_{ijkk'} w_{ij}^{(k)} w_{ij}^{(k')} + \lambda \sum_{ijk} f(w_{ij}^{(k)}, u_{ik}, v_{jk}) \\ \text{s.t. } & \boldsymbol{U} \in \{0,1\}^{m\times r},\ \boldsymbol{V} \in \{0,1\}^{n\times r},\ \boldsymbol{W}^{(k)} \in \{0,1\}^{m\times n} \text{ for all } k \in [r]. \end{aligned} \tag{8.7}$$

Proposition 65. *Suppose $\lambda > 2r\|\boldsymbol{A}\|_{\mathrm{F}}^2$. A point $(\boldsymbol{U}, \boldsymbol{V}, \{\boldsymbol{W}^{(k)}\})$ minimizes (8.5) if and only if it minimizes (8.7).*

Proof. For brevity, we denote the objectives in (8.5) and (8.7) by OBJ_1 and OBJ_2, respectively. Setting all entries in the matrices $\boldsymbol{U}$, $\boldsymbol{V}$, $\boldsymbol{W}^{(1)}, \ldots, \boldsymbol{W}^{(r)}$ to zero would yield an objective value of

$\mathrm{OBJ}_2(\boldsymbol{U}, \boldsymbol{V}, \{\boldsymbol{W}^{(k)}\}) = \|\boldsymbol{A}\|_{\mathrm{F}}^2$. Moreover,

$$-2\sum_{ijk} a_{ij} w_{ij}^{(k)} + \sum_{ijkk'} w_{ij}^{(k)} w_{ij}^{(k')} \geq -2r\|\boldsymbol{A}\|_{\mathrm{F}}^2,$$

so any point $(\boldsymbol{U}, \boldsymbol{V}, \{\boldsymbol{W}^{(k)}\})$ that violates (8.4) would satisfy $\mathrm{OBJ}_2(\boldsymbol{U}, \boldsymbol{V}, \{\boldsymbol{W}^{(k)}\}) > \|\boldsymbol{A}\|_{\mathrm{F}}^2$ and therefore could not be a minimizer of (8.7). Any minimizer of (8.7) must therefore satisfy (8.4).

Suppose $p^* \stackrel{\text{def}}{=} (\boldsymbol{U}^*, \boldsymbol{V}^*, \{\boldsymbol{W}^{(k)*}\})$ minimizes (8.5). Since p^* satisfies (8.4), all penalty terms in (8.7) are zero for this point, and therefore $\mathrm{OBJ}_1(p^*) = \mathrm{OBJ}_2(p^*)$. p^* is also a minimizer of (8.7). If it was not, there would be a minimizer $p^\dagger$ of (8.7) for which $\mathrm{OBJ}_2(p^\dagger) < \mathrm{OBJ}_2(p^*)$ and which would satisfy (8.4). $p^\dagger$ would therefore be a feasible solution for (8.5) and it would satisfy $\mathrm{OBJ}_1(p^\dagger) = \mathrm{OBJ}_2(p^\dagger) < \mathrm{OBJ}_1(p^*)$, which contradicts the optimality of p^*.

Suppose $p^\dagger \stackrel{\text{def}}{=} (\boldsymbol{U}^\dagger, \boldsymbol{V}^\dagger, \{\boldsymbol{W}^{(k)\dagger}\})$ minimizes (8.7). Then $p^\dagger$ satisfies (8.4) and is therefore a feasible solution to (8.5) satisfying $\mathrm{OBJ}_1(p^\dagger) = \mathrm{OBJ}_2(p^\dagger)$. $p^\dagger$ is also a minimizer of (8.5). If it was not, there would be a minimizer p^* of (8.5) which would satisfy $\mathrm{OBJ}_2(p^*) = \mathrm{OBJ}_1(p^*) < \mathrm{OBJ}_1(p^\dagger) = \mathrm{OBJ}_2(p^\dagger)$, contradicting the optimality of $p^\dagger$. □

Since (8.1) and (8.5) are equivalent, Proposition 65 implies that the matrices $\boldsymbol{U}$ and $\boldsymbol{V}$ we get from minimizing (8.7) are minimizers of (8.1) when λ is sufficiently large.

We now state the QUBO formulation of (8.7). Define $\boldsymbol{u} \stackrel{\text{def}}{=} \mathrm{vec}(\boldsymbol{U})$, $\boldsymbol{v} \stackrel{\text{def}}{=} \mathrm{vec}(\boldsymbol{V})$, and $\boldsymbol{w}^{(k)} \stackrel{\text{def}}{=} \mathrm{vec}(\boldsymbol{W}^{(k)})$ for each $k \in [r]$, and let

$$\boldsymbol{x} \stackrel{\text{def}}{=} \left[\boldsymbol{u}\,;\quad \boldsymbol{v}\,;\quad \boldsymbol{w}^{(1)}\,;\quad \boldsymbol{w}^{(2)}\,;\quad \cdots\,;\quad \boldsymbol{w}^{(r)}\right], \tag{8.8}$$

where $\boldsymbol{x}$ is a column vector of length $(m+n+mn)r$. Furthermore, define the QUBO matrix as

$$\boldsymbol{Q} \stackrel{\text{def}}{=} \begin{bmatrix} \boldsymbol{Q}^{(1)} & \boldsymbol{Q}^{(2)} \\ \boldsymbol{0} & \boldsymbol{Q}^{(3)} \end{bmatrix} \in \mathbb{R}^{(m+n+mn)r \times (m+n+mn)r}, \tag{8.9}$$

where

$$\boldsymbol{Q}^{(1)} \stackrel{\text{def}}{=} \frac{\lambda}{2}\begin{bmatrix} \boldsymbol{0}_{mr} & \boldsymbol{I}_r \otimes \boldsymbol{1}_{m\times n} \\ \boldsymbol{I}_r \otimes \boldsymbol{1}_{n\times m} & \boldsymbol{0}_{nr} \end{bmatrix},$$

$$\boldsymbol{Q}^{(2)} \stackrel{\text{def}}{=} -2\lambda \begin{bmatrix} \boldsymbol{I}_r \otimes \boldsymbol{1}_{1\times n} \otimes \boldsymbol{I}_m \\ \boldsymbol{I}_{nr} \otimes \boldsymbol{1}_{1\times m} \end{bmatrix},$$

$$\boldsymbol{Q}^{(3)} \stackrel{\text{def}}{=} \boldsymbol{1}_r \otimes \boldsymbol{I}_{mn} - 2\,\mathrm{diag}\left((\boldsymbol{1}_{r\times 1} \otimes \boldsymbol{I}_{mn})\mathrm{vec}(\boldsymbol{A})\right) + 3\lambda \boldsymbol{I}_{mnr}.$$

Proposition 66. *With $\boldsymbol{x}$ and $\boldsymbol{Q}$ defined as in* (8.8) *and* (8.9)*, respectively, the problem* (8.7) *can be written as*

$$\min_{\boldsymbol{x}} \ \|\boldsymbol{A}\|_{\mathrm{F}}^2 + \boldsymbol{x}^\top \boldsymbol{Q}\boldsymbol{x} \quad \text{s.t. } \ \boldsymbol{x} \in \{0,1\}^{(m+n+mn)r}. \tag{8.10}$$

The proof is a straightforward but somewhat tedious calculation and is omitted. Although removing the constant $\|\boldsymbol{A}\|_{\mathrm{F}}^2$ does not affect the minimizer(s) of (8.10), it can serve as a useful target: If $\boldsymbol{x}^\top \boldsymbol{Q}\boldsymbol{x} = -\|\boldsymbol{A}\|_{\mathrm{F}}^2$, then we know that we have found a global minimum, provided that the condition on λ in Proposition 65 is satisfied. Such a target value can be supplied to QUBO solvers like D-Wave's QBSolv to allow for early termination when the target is reached.

8.3.2 Formulation 2

For our second formulation, we again consider (8.3) and introduce auxiliary variables

$$\begin{aligned} \widetilde{u}_{i(kk')} &\stackrel{\text{def}}{=} u_{ik}u_{ik'} \quad \text{for } i \in [m],\ k,k' \in [r], \\ \widetilde{v}_{j(kk')} &\stackrel{\text{def}}{=} v_{jk}v_{jk'} \quad \text{for } j \in [n],\ k,k' \in [r]. \end{aligned} \tag{8.11}$$

We treat $\widetilde{\boldsymbol{U}} = (\widetilde{u}_{i(kk')})$ and $\widetilde{\boldsymbol{V}} = (\widetilde{v}_{j(kk')})$ as matrices of size $m \times r^2$ and $n \times r^2$, respectively, with $\widetilde{\boldsymbol{u}}_{*(kk')}$ being the $(k + k'r)$th column of $\widetilde{\boldsymbol{U}}$, with similar column ordering for $\widetilde{\boldsymbol{V}}$. An equivalent formulation to (8.1) is then

$$\begin{aligned} &\min_{\boldsymbol{U},\boldsymbol{V},\widetilde{\boldsymbol{U}},\widetilde{\boldsymbol{V}}} \ \|\boldsymbol{A}\|_{\mathrm{F}}^2 - 2\sum_{ijk} a_{ij}u_{ik}v_{jk} + \sum_{ijkk'} \widetilde{u}_{i(kk')}\widetilde{v}_{j(kk')} \\ &\text{s.t. } \boldsymbol{U} \in \{0,1\}^{m\times r},\ \boldsymbol{V} \in \{0,1\}^{n\times r},\ (8.11) \text{ satisfied.} \end{aligned} \tag{8.12}$$

We use the function f defined in (8.6) to incorporate the constraints (8.11) in the objective. A penalty variant of (8.12) is

$$\begin{aligned}\min_{\boldsymbol{U},\boldsymbol{V},\widetilde{\boldsymbol{U}},\widetilde{\boldsymbol{V}}} \ & \|\boldsymbol{A}\|_{\mathrm{F}}^2 - 2\sum_{ijk} a_{ij}u_{ik}v_{jk} + \sum_{ijkk'} \widetilde{u}_{i(kk')}\widetilde{v}_{j(kk')} \\ & + \lambda \sum_{ikk'} f(\widetilde{u}_{i(kk')}, u_{ik}, u_{ik'}) + \lambda \sum_{jkk'} f(\widetilde{v}_{j(kk')}, v_{jk}, v_{jk'}) \\ \text{s.t. } & \boldsymbol{U} \in \{0,1\}^{m\times r},\ \boldsymbol{V} \in \{0,1\}^{n\times r},\ \widetilde{\boldsymbol{U}} \in \{0,1\}^{m\times r^2},\ \widetilde{\boldsymbol{V}} \in \{0,1\}^{n\times r^2}.\end{aligned} \tag{8.13}$$

Proposition 67. *Suppose $\lambda > 2r\|\boldsymbol{A}\|_{\mathrm{F}}^2$. A point $(\boldsymbol{U},\boldsymbol{V},\widetilde{\boldsymbol{U}},\widetilde{\boldsymbol{V}})$ minimizes* (8.12) *if and only if it minimizes* (8.13)*.*

The proof is similar to that for Proposition 65 and is omitted. Since (8.1) and (8.12) are equivalent, Proposition 67 implies that the matrices $\boldsymbol{U}$ and $\boldsymbol{V}$ we get from minimizing (8.13) are minimizers of (8.1) when λ is sufficiently large.

We now state the QUBO formulation of (8.13). Define $\boldsymbol{u}$ and $\boldsymbol{v}$ as before. Furthermore, define $\widetilde{\boldsymbol{u}} \stackrel{\text{def}}{=} \mathrm{vec}(\widetilde{\boldsymbol{U}})$, $\widetilde{\boldsymbol{v}} \stackrel{\text{def}}{=} \mathrm{vec}(\widetilde{\boldsymbol{V}})$ and

$$\boldsymbol{y} \stackrel{\text{def}}{=} \begin{bmatrix}\boldsymbol{u}\,; & \boldsymbol{v}\,; & \widetilde{\boldsymbol{u}}\,; & \widetilde{\boldsymbol{v}}\end{bmatrix}, \tag{8.14}$$

where $\boldsymbol{y}$ is a column vector of length $(m+n)(r+r^2)$. Furthermore, define the QUBO matrix as

$$\boldsymbol{P} \stackrel{\text{def}}{=} \begin{bmatrix}\boldsymbol{P}^{(1)} & \boldsymbol{P}^{(2)} \\ \boldsymbol{0} & \boldsymbol{P}^{(3)}\end{bmatrix} \in \mathbb{R}^{(m+n)(r+r^2)\times(m+n)(r+r^2)}, \tag{8.15}$$

where

$$\begin{aligned}\boldsymbol{P}^{(1)} &\stackrel{\text{def}}{=} \begin{bmatrix}\lambda\boldsymbol{1}_r \otimes \boldsymbol{I}_m & -2\boldsymbol{I}_r \otimes \boldsymbol{A} \\ \boldsymbol{0}_{nr\times mr} & \lambda\boldsymbol{1}_r \otimes \boldsymbol{I}_n\end{bmatrix}, \\ \boldsymbol{P}^{(2)} &\stackrel{\text{def}}{=} -2\lambda \begin{bmatrix}\boldsymbol{1}_{1\times r} \otimes \boldsymbol{I}_{mr} & \boldsymbol{0}_{mr\times nr^2} \\ \boldsymbol{0}_{nr\times mr^2} & \boldsymbol{1}_{1\times r} \otimes \boldsymbol{I}_{nr}\end{bmatrix} - 2\lambda \begin{bmatrix}\boldsymbol{I}_r \otimes \boldsymbol{1}_{1\times r} \otimes \boldsymbol{I}_m & \boldsymbol{0}_{mr\times nr^2} \\ \boldsymbol{0}_{nr\times mr^2} & \boldsymbol{I}_r \otimes \boldsymbol{1}_{1\times r} \otimes \boldsymbol{I}_n\end{bmatrix}, \\ \boldsymbol{P}^{(3)} &\stackrel{\text{def}}{=} \begin{bmatrix}3\lambda\boldsymbol{I}_{mr^2} & \boldsymbol{I}_{r^2} \otimes \boldsymbol{1}_{m\times n} \\ \boldsymbol{0}_{nr^2\times mr^2} & 3\lambda\boldsymbol{I}_{nr^2}\end{bmatrix}.\end{aligned}$$

Proposition 68. *With* $\boldsymbol{y}$ *and* $\boldsymbol{P}$ *defined as in* (8.14) *and* (8.15)*, respectively, the problem* (8.13) *can be written as*

$$\min_{\boldsymbol{y}} \ \|\boldsymbol{A}\|_{\mathrm{F}}^2 + \boldsymbol{y}^\top \boldsymbol{P}\boldsymbol{y} \quad \text{s.t. } \ \boldsymbol{y} \in \{0,1\}^{(m+n)(r+r^2)}. \tag{8.16}$$

The proof is a straightforward and is omitted.

8.4 Useful constraints for data analysis

In this section we show how certain constraints that are helpful for data mining tasks easily can be incorporated into the QUBO formulations. One approach to clustering of the rows and/or columns of a binary matrix $\boldsymbol{A}$ is to compute a BMF $\boldsymbol{A} \approx \boldsymbol{U}\boldsymbol{V}^\top$ and then use the information in $\boldsymbol{U}$ and $\boldsymbol{V}$ to build the clusters. This idea is used e.g. by [223, 224] for gene expression sample clustering and document clustering. For gene expression data, the rows of $\boldsymbol{A}$ represent genes and the columns represent samples, e.g. from different people. An unsupervised data mining task on such a dataset could be to identify and cluster people based on if they have cancer or not. One way to do this is to compute a rank-2 BMF of $\boldsymbol{A}$ and assign sample j to cluster $k \in \{1,2\}$ if $v_{jk} = 1$. In many cases, it is reasonable to require that each column belongs to precisely one cluster. For example, when clustering people based on if they have cancer or not, we want to assign every person to precisely one of two clusters. Such a requirement can be incorporated by enforcing that

$$\sum_k v_{jk} = 1 \quad \text{for all } j. \tag{8.17}$$

A penalty variant of this constraint is

$$\lambda \sum_j \Big(1 - \sum_k v_{jk} + 2 \sum_{k<k'} v_{jk} v_{jk'}\Big), \tag{8.18}$$

where $\lambda > 0$. Since $\boldsymbol{V}$ is binary, the penalty is zero when (8.17) is satisfied, and at least λ otherwise. This penalty can simply be added to the objectives in (8.7) and (8.13). As before, we can ensure that the penalized and constrained formulations have the same minimizers by choosing $\lambda > 2r\|\boldsymbol{A}\|_{\mathrm{F}}^2$.

The penalty (8.18) is straightforward to incorporate into either QUBO formulation. Define

an $(m+n)r \times (m+n)r$ matrix

$$\boldsymbol{C} \stackrel{\text{def}}{=} \begin{bmatrix} \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \lambda(\mathbf{1}_r \otimes \boldsymbol{I}_n - 2\boldsymbol{I}_{nr}) \end{bmatrix}.$$

The QUBO formulations in (8.10) and (8.16) are easily modified to incorporate (8.18) by defining modified QUBO matrices

$$\boldsymbol{Q}' \stackrel{\text{def}}{=} \begin{bmatrix} \boldsymbol{Q}^{(1)} + \boldsymbol{C} & \boldsymbol{Q}^{(2)} \\ \mathbf{0} & \boldsymbol{Q}^{(3)} \end{bmatrix}, \quad \boldsymbol{P}' \stackrel{\text{def}}{=} \begin{bmatrix} \boldsymbol{P}^{(1)} + \boldsymbol{C} & \boldsymbol{P}^{(2)} \\ \mathbf{0} & \boldsymbol{P}^{(3)} \end{bmatrix},$$

where the submatrices $\boldsymbol{Q}^{(i)}$ and $\boldsymbol{P}^{(i)}$ are defined as before.

8.5 Handling large rectangular matrices

In this section, we present a strategy for handling large rectangular matrices. We consider the case when $\boldsymbol{A}$ is $m \times n$ with $m \gg n$ and n is of moderate size. These ideas also apply when $n \gg m$ and m is of moderate size. Random sampling of rows and columns is a popular technique in numerical linear algebra for compressing large matrices. These compressed matrices are then used instead of the full matrices in computations. For an introduction to this topic we recommend the survey by Mahoney [138].

A popular sampling approach is to sample according to the **leverage scores** of the matrix. Suppose $\boldsymbol{A} \in \mathbb{R}^{m \times n}$ is a nonzero matrix, and let $\boldsymbol{B} \in \mathbb{R}^{m \times \operatorname{rank}(\boldsymbol{A})}$ be an orthonormal matrix whose columns form a basis for range($\boldsymbol{A}$). The leverage scores of $\boldsymbol{A}$ are defined as $\ell_i(\boldsymbol{A}) \stackrel{\text{def}}{=} \|\boldsymbol{B}_{i*}\|_2^2$ for $i \in [m]$. When sampling rows of $\boldsymbol{A}$ according to the leverage scores, we sample the ith row of $\boldsymbol{A}$ with probability $p_i \stackrel{\text{def}}{=} \ell_i(\boldsymbol{A})/\operatorname{rank}(\boldsymbol{A})$ for $i \in [m]$. This definition guarantees that $\sum_i p_i = 1$. We use leverage score sampling as a heuristic for compressing $\boldsymbol{A}$ by sampling $s \ll m$ of the rows of $\boldsymbol{A}$ with replacement according to the distribution (p_i) and putting these in a new matrix $\boldsymbol{A}^{(s)} \in \mathbb{R}^{s \times n}$. We then compute a rank-r BMF $\boldsymbol{A}^{(s)} \approx \boldsymbol{U}^{(s)}\boldsymbol{V}^\top$. To get a BMF for the original matrix $\boldsymbol{A}$, we then solve the binary least squares (BLS) problem

$$\boldsymbol{U} \stackrel{\text{def}}{=} \operatorname*{arg\,min}_{\boldsymbol{U}' \in \{0,1\}^{m \times r}} \|\boldsymbol{A} - \boldsymbol{U}'\boldsymbol{V}^\top\|_{\text{F}}^2, \tag{8.19}$$

where $\boldsymbol{V}$ comes from the factorization of $\boldsymbol{A}^{(s)}$. By expanding the objective, the problem in (8.19) can be written as m independent BLS problems involving r binary variables. These BLS problems can be solved via a QUBO formulation, as discussed in [164, 165]. Alternatively, when r is small, the optimal solution to each BLS problem can be found by simply testing all 2^r possible solutions. As an optional step after computing $\boldsymbol{U}$, a few additional alternating BLS steps can be added. This is done by minimizing the objective in (8.19) in an alternating fashion, first solving for $\boldsymbol{V}$ and treating $\boldsymbol{U}$ as fixed, and then solving for $\boldsymbol{U}$ and treating $\boldsymbol{V}$ as fixed.

8.6 Experiments

We found that Formulation 1 performs better than Formulation 2 on the Fujitsu DA. We therefore use the former in our experiments and refer to it as "DA BMF" or just "DA" in the tables. Additionally, we try adding a few extra alternating BLS steps (as discussed in Section 8.5) to the solutions we get from the DA. We do at most 20 alternating BLS solves, and whenever no improvement occurs after two consecutive BLS solves, we terminate. Since $r \leq 5$ in our experiments, we solve the BLS problems exactly by checking all possible solutions. We refer to this method as "DA+ALS BMF" or just "DA+ALS" in the tables. For some of the real datasets, we incorporate the constraint in Section 8.4. For cases when $\boldsymbol{A}$ is large and rectangular, we use the sampling technique in Section 8.5. We will point out when the sampling and/or additional constraints are used.

We run our proposed method on the Fujitsu DA for a fixed number of 1e+9 iterations[1] . We do not try to find an optimal number of iterations. We take this approach to avoid cherry picking a number of iterations that works great for each individual problem. By choosing a relative large number of iterations, we are also hoping to push the hardware to see how good solutions it can find.

As discussed by Glover et al. [82], although the penalty λ needs to be sufficiently large to ensure that the constrained and penalized versions of our optimization problems have the same minimizers, setting λ to a smaller value may improve the solution produced by a QUBO solver in practice. An intuitive explanation for this phenomenon is that a large λ value gives a steeper

[1] An iteration refers to one iteration of the for loop on line 5 in Algorithm 2 of [5].

optimization landscape which can make it difficult for a solver to escape local minima. We find this to be true when running our methods on the Fujitsu DA as well. We use $\lambda = 1$ in all our experiments since we found this to improve the solution quality, while at the same time avoiding constraint violations.

As mentioned in Section 8.2, the two methods by Zhang et al. [223, 224] are the most popular for the variant of BMF we consider. We therefore use these methods for comparison in our experiments. We refer to them as "Penalized" and "Thresholded," respectively. For the penalized version, we use the implementation available in the Nimfa[2] Python library. We leave all parameters to their default values, except the maximum number of iterations (`max_iter`) and the frequency of the convergence test (`test_conv`) which we both set to 1000 since we find that this improves performance substantially over the defaults in our experiments. We wrote our own implementation of the thresholded method since we could not find an existing implementation; see Section D.1 of the supplement for details.

We also include a simple baseline method. The idea behind it is simple: If we seek a rank-r BMF of $\boldsymbol{A}$, we can find one by simply choosing the densest r rows/columns in $\boldsymbol{A}$. Alternatively, when $\boldsymbol{A}$ has high density, we can approximate it by a rank-1 BMF with all entries equal to 1. This is clearly a very crude method, but it serves as a useful baseline and sanity check for the more sophisticated methods. See Section D.2 of the supplement for details on the baseline method.

All experiment results are evaluated in terms of the following relative error measure: $\|\boldsymbol{A} - \boldsymbol{U}\boldsymbol{V}^\top\|_{\mathrm{F}}^2 / \|\boldsymbol{A}\|_{\mathrm{F}}^2$. The norm is squared since this is more natural for binary data: When $\boldsymbol{U}$ and $\boldsymbol{V}$ are binary, $\|\boldsymbol{A} - \boldsymbol{U}\boldsymbol{V}^\top\|_{\mathrm{F}}^2$ is the number of entries that are incorrect in the decomposition and $\|\boldsymbol{A}\|_{\mathrm{F}}^2$ is the number of nonzero entries in $\boldsymbol{A}$.

8.6.1 Synthetic data

For the first set of synthetic experiments, $\boldsymbol{A}$ is generated in such a way that it has an exact rank-r decomposition, for $r \in [5]$. Our algorithm for generating these matrices is described in

[2] We use the Bmf method. See `http://nimfa.biolab.si` for details.

Table 8.1: Mean relative error for synthetic $\boldsymbol{A}$ with an exact decomposition. The * symbol indicates methods we propose. Best results are underlined.

	Target ranks r ($m = 30$)					Target ranks r ($m = 2000$)				
Method	1	2	3	4	5	1	2	3	4	5
*DA	0	0	0	0	0	0	0	0	0	0
*DA+ALS	0	0	0	0	0	0	0	0	0	0
Penalized	0	0	0	0.0170	0.0148	0	0	0	0	0.0361
Thresholded	0	0	0	0.0052	0.0273	0	0	0	0.0330	0.0594
*Baseline	0.8265	0.8706	0.8157	0.7434	0.6977	0.9181	0.9075	0.8730	0.8101	0.7585

	Target ranks r ($m = 50000$)				
Method	1	2	3	4	5
*DA	0	0	0	0	0
*DA+ALS	0	0	0	0	0
Penalized	0	0	0	0	0.0159
Thresholded	0	0	0.0240	0.0317	0.0632
*Baseline	0.8831	0.9088	0.8634	0.8127	0.7520

Section D.3 of the supplement. We use the true rank as the target rank for the decompositions. Ideally, the different methods should therefore be able to find an exact decomposition. We generate $\boldsymbol{A}$ to have $n = 30$ columns and $m \in \{30, 2000, 50000\}$ rows. When $m \geq 2000$, we use the sampling technique described in Section 8.5 for all our methods. We use a sample size of 30, so that $\boldsymbol{A}^{(s)} \in \{0, 1\}^{30\times 30}$. All experiments are repeated 10 times. Table 8.1 reports the mean relative error for these experiments.

For the second set of synthetic experiments, we draw each entry a_{ij} independently from a Bernoulli distribution with probability of success $p \in \{0.2, 0.5, 0.8\}$. We generate $\boldsymbol{A}$ with $n = 30$ columns and $m \in \{30, 2000, 50000\}$ rows, and use target ranks $r \in [5]$. When $m \geq 2000$, we use the sampling technique described in Section 8.5 for our methods with a sample size of $s = 30$. All experiments are repeated 10 times. Tables 8.2, 8.3 and 8.4 report mean relative errors for each of the three different values of p.

In all synthetic experiments, the QUBO problem has $(30 + 30 + 30^2)r = 960r$ binary variables. This is also true for the large rectangular matrices due to the choice of sample size $s = 30$.

Table 8.2: Mean relative error for synthetic $\boldsymbol{A}$ for which $a_{ij} \sim$ Bernoulli(0.2). The * symbol indicates methods we propose. Best results are underlined.

	Target ranks r ($m = 30$)					Target ranks r ($m = 2000$)				
Method	1	2	3	4	5	1	2	3	4	5
*DA	0.9209	0.8565	0.8018	0.7643	0.7161	0.9902	0.9743	0.9659	0.9452	0.9484
*DA-ALS	0.9209	0.8565	0.8018	0.7637	0.7161	0.9895	0.9727	0.9624	0.9403	0.9436
Penalized	0.9989	0.9230	0.8559	0.7875	0.7397	1.0000	1.0000	0.9998	0.9918	0.9751
Thresholded	0.9555	0.8904	0.8409	0.7954	0.7609	0.9990	0.9932	0.9777	0.9601	0.9368
*Baseline	0.9365	0.8787	0.8238	0.7734	0.7253	0.9639	0.9281	0.8928	0.8577	0.8228

	Target ranks r ($m = 50000$)				
Method	1	2	3	4	5
*DA	0.9914	0.9785	0.9624	0.9491	0.9421
*DA+ALS	0.9914	0.9785	0.9623	0.9489	0.9413
Penalized	1	1	1	0.9986	0.9839
Thresholded	0.9998	0.9951	0.9833	0.9661	0.9443
*Baseline	0.9660	0.9323	0.8985	0.8649	0.8313

Table 8.3: Mean relative error for synthetic $\boldsymbol{A}$ for which $a_{ij} \sim$ Bernoulli(0.5). The * symbol indicates methods we propose. Best results are underlined.

	Target ranks r ($m = 30$)					Target ranks r ($m = 2000$)				
Method	1	2	3	4	5	1	2	3	4	5
*DA	0.7844	0.6773	0.6056	0.5536	0.5165	0.8809	0.8234	0.7948	0.7540	0.7347
*DA+ALS	0.7844	0.6773	0.6054	0.5534	0.5146	0.8628	0.8100	0.7888	0.7507	0.7301
Penalized	0.8606	0.7306	0.6611	0.6147	0.5749	0.9802	0.9635	0.9371	0.8862	0.8478
Thresholded	0.7983	0.7170	0.6775	0.6875	0.6680	0.8568	0.8346	0.8299	0.8311	0.8038
*Baseline	0.9519	0.9103	0.8679	0.8265	0.7870	0.9651	0.9307	0.8964	0.8622	0.8282

	Target ranks r ($m = 50000$)				
Method	1	2	3	4	5
*DA	0.8820	0.8214	0.7846	0.7550	0.7325
*DA+ALS	0.8634	0.8099	0.7807	0.7521	0.7324
Penalized	0.9912	0.9819	0.9565	0.9033	0.8770
Thresholded	0.8555	0.8358	0.8324	0.8418	0.8323
*Baseline	0.9664	0.9328	0.8992	0.8657	0.8322

8.6.2 Real data

In the first experiment on real data we consider the MNIST handwritten digits dataset [125]. We consider ten instances of each digit 0–9. The digits are 28×28 grayscale images with pixel values in the range $[0, 255]$, where 0 represents white and 255 represents black. We make these

Table 8.4: Mean relative error for synthetic $\boldsymbol{A}$ for which $a_{ij} \sim$ Bernoulli(0.8). The * symbol indicates methods we propose. Best results are underlined.

	Target ranks r ($m = 30$)					Target ranks r ($m = 2000$)				
Method	1	2	3	4	5	1	2	3	4	5
*DA	0.2446	0.2288	0.2192	0.2097	0.2050	0.2503	0.2550	0.2539	0.2590	0.2480
*DA+ALS	0.2446	0.2257	0.2129	0.2012	0.1916	0.2503	0.2473	0.2459	0.2402	0.2384
Penalized	0.2446	0.2441	0.2539	0.2843	0.3346	0.2503	0.2500	0.3202	0.4757	0.5743
Thresholded	0.2446	0.2904	0.4390	0.5078	0.5332	0.2503	0.2625	0.6319	0.7702	0.7581
*Baseline	0.2446	0.2446	0.2446	0.2446	0.2446	0.2503	0.2503	0.2503	0.2503	0.2503

	Target ranks r ($m = 50000$)				
Method	1	2	3	4	5
*DA	0.2502	0.2546	0.2542	0.2632	0.2438
*DA+ALS	0.2502	0.2471	0.2447	0.2428	0.2363
Penalized	0.2502	0.2574	0.3220	0.4996	0.6021
Thresholded	0.2502	0.2527	0.6923	0.8000	0.8118
*Baseline	0.2502	0.2502	0.2502	0.2502	0.2502

binary by setting values less than 50 to 0, and values greater than or equal to 50 to 1. We apply BMF to the digits with target ranks $r \in [5]$. The QUBO problem in DA BMF and DA+ALS BMF has $(28 + 28 + 28^2)r = 840r$ binary variables. Table 8.5 presents the mean relative error for each method across all instances of all digits. Figure 8.2 shows an example of the binary digit 3 and the low-rank approximations given by our DA+ALS BMF method.

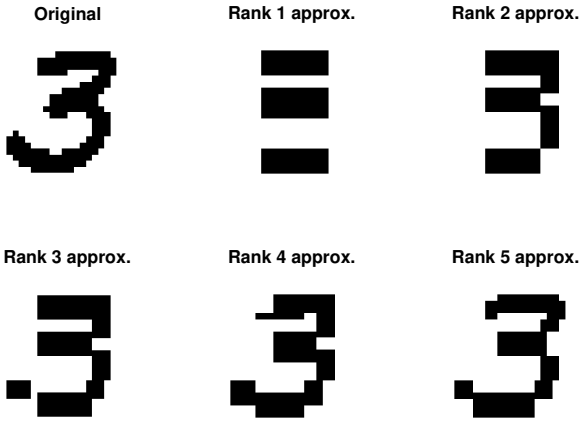

Figure 8.2: Binary low rank approximation to MNIST digit using DA+ALS BMF.

In the second experiment on real data, we consider two gene expression datasets for two types

Table 8.5: Mean relative error for MNIST experiments. The * symbol indicates methods we propose. Best results are underlined.

	Target ranks r				
Method	1	2	3	4	5
*DA	0.5796	0.3832	0.2673	0.1983	0.1522
*DA+ALS	0.5796	0.3832	0.2672	0.1982	0.1522
Penalized	0.6070	0.4072	0.2951	0.2238	0.1836
Thresholded	0.5872	0.4141	0.3171	0.2797	0.2738
*Baseline	0.8684	0.7484	0.6400	0.5476	0.4655

of cancer: leukemia and malignant melanoma. The first dataset[3] contains 38 gene samples for two kinds of leukemia, one of which can be further split into two subtypes [39]. The second dataset[4] contains 38 gene samples, 31 of which are melanomas and 7 of which are controls [28]. We make these datasets binary by using the same thresholding approach as [223] which we describe here briefly. Let $0 \leq c_1 < c_2$ be two real numbers. For a matrix $\boldsymbol{A} \in \mathbb{R}^{m \times n}$ with nonnegative entries, let $\kappa \stackrel{\text{def}}{=} \sum_{ij} a_{ij}/(mn)$. $\widetilde{\boldsymbol{A}}$, a discretized version of $\boldsymbol{A}$, is computed by setting its entries

$$\widetilde{a}_{ij} = \begin{cases} 1 & \text{if } a_{ij} \leq \kappa c_1 \text{ or } a_{ij} \geq \kappa c_2, \\ 0 & \text{otherwise.} \end{cases}$$

Optionally, the columns of $\boldsymbol{A}$ can be normalized so that they have unit Euclidean norm prior to discretization. As an additional step, we remove any rows from $\widetilde{\boldsymbol{A}}$ that contain only zeros. Following [223, 224], we use $c_1 = 1/7$ and $c_2 = 5$ without column normalization on the leukemia dataset. The melanoma dataset contains negative entries. We therefore first add a constant $-\min_{ij} a_{ij}$ to all entries in $\boldsymbol{A}$. Then, we normalize the columns of the resulting matrix and discretize using $c_1 = 0.96$ and $c_2 = 1.04$. Figure 8.3 shows what the thresholded leukemia data looks like.

After thresholding and removing any rows that are all zero, the two datasets are matrices of size 4806×38 and 2201×38, respectively. We use the sampling technique in Section 8.5 for both datasets with a sample size of $s = 30$. Furthermore, we include the constraints on the $\boldsymbol{V}$ matrix in

[3] Available at `https://www.pnas.org/content/101/12/4164`.
[4] Available at `https://schlieplab.org/Static/Supplements/CompCancer/CDNA/bittner-2000/`.

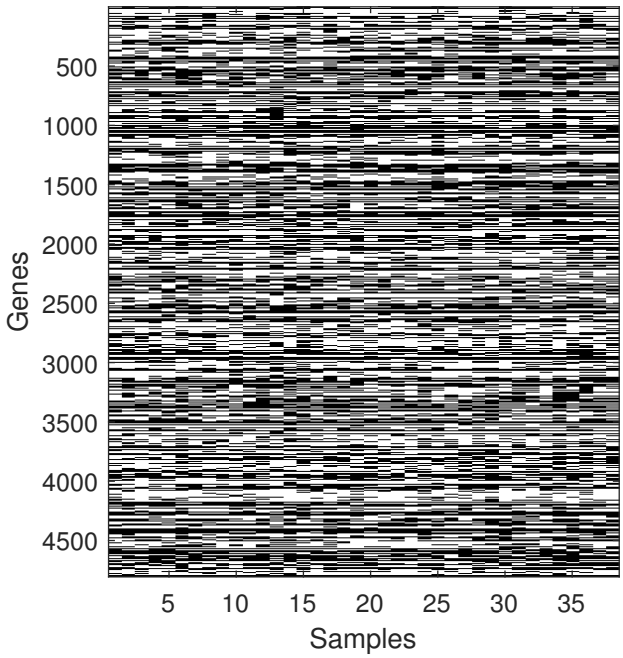

Figure 8.3: The thresholded leukemia data. Black entries are 1 and white entries are 0.

Table 8.6: Mean relative error for gene expression data. The * symbol indicates methods we propose. Best results are underlined.

	Target ranks r (leukemia dataset [39])					Target ranks r (melanoma dataset [28])				
Method	1	2	3	4	5	1	2	3	4	5
*DA	0.4292	0.4069	0.3853	0.4257	0.4025	0.8898	0.8392	0.7869	0.7850	0.7722
*DA+ALS	0.3939	0.3806	0.3716	0.3683	0.3600	0.8572	0.7757	0.7416	0.7252	0.7113
Penalized	0.3977	0.3952	0.4767	0.5399	0.5810	0.8662	0.8405	0.8110	0.7903	0.7633
Thresholded	0.3939	0.4219	0.6195	0.6625	0.6894	0.8662	0.8338	0.8226	0.8120	0.8116
*Baseline	0.9698	0.9408	0.9123	0.8842	0.8563	0.9504	0.9027	0.8569	0.8132	0.7695

the QUBO formulations as discussed in Section 8.4. Although we expect that this will increase the decomposition error somewhat, including such constraints can be helpful e.g. when clustering the samples. The QUBO problem in our methods has $(30 + 38 + 30 \cdot 38)r = 1208r$ binary variables. Table 8.6 reports the mean relative error across 10 trials for each dataset.

8.6.3 Discussion

The accuracy improvement of DA+ALS BMF over DA BMF is typically very small, but more substantial in a few cases. Our DA+ALS BMF has the same or better accuracy as either of the

methods by [223, 224] in 72 of the 75 experiments reported in the tables in this paper, and a strictly better accuracy in 57 of the experiments.

For a fixed rank, the number of binary variables for the QUBO problem in DA BMF is similar across all experiments. The anneal time, which is the time spent by the DA looking for a solution, is about 40 seconds for the largest problems. The additional ALS steps used for DA+ALS take on average much less than a second when $m = 30$, and less than about 3 seconds when $m = 2000$. For $m = 50000$, the additional time for ALS can be more substantial, adding on average as much as 66 seconds for rank 5 experiments.

Two advantages of the methods by [223, 224] are that they typically are quite fast and that they can run on a standard computer. For all experiments except the synthetic ones with $m = 50000$, their penalized and thresholded algorithms run in less than 2 and 20 seconds on average, respectively. The very large experiments with $m = 50000$ can take longer, up to 39 and 289 seconds on average for the penalized and thresholded algorithms, respectively, when $r = 5$.

Based on these observations, DA+ALS BMF seems like the superior method when accuracy is crucial. The methods by [223, 224] may be more suitable when speed and accessibility are more important. With that said, we believe that the DA could be run with many fewer iterations with little or no degradation in performance in most of our experiments. The trade-off between accuracy and speed for the DA, and how to choose the number of iterations to strike a good balance, are interesting directions for future research.

Certain matrices, like those of size 2000×30 and expected density 0.2 considered in Table 8.2, seem inherently difficult to handle for any of the sophisticated methods. Indeed, it is surprising that the simple baseline method substantially outperforms all other methods.

8.7 Conclusion

BMF has many applications in data mining. We have presented two ways to formulate BMF as QUBO problems. These formulations can be used to do BMF on special purpose hardware, such as the D-Wave quantum annealer and the Fujitsu DA. We also discussed how clustering constraints

can easily be incorporated into our QUBO formulations. Moreover, we showed how sampling and alternating binary least squares can be used to handle large rectangular matrices. Our experiments, which we run on the Fujitsu DA, are encouraging and show that our proposed methods typically give more accurate solutions than competing methods.

The special purpose hardware technologies discussed in this paper are still in an early phase of development. As these technologies mature, we believe that they will emerge as powerful tools for solving problems in data mining and other areas.

Bibliography

[1] Kareem S. Aggour, Alex Gittens, and Bülent Yener. Adaptive Sketching for Fast and Convergent Canonical Polyadic Decomposition. In International Conference on Machine Learning. PMLR, 2020.

[2] Salman Ahmadi-Asl, Andrzej Cichocki, Anh Huy Phan, Maame G. Asante-Mensah, Farid Mousavi, Ivan Oseledets, and Toshihisa Tanaka. Randomized algorithms for fast computation of low-rank tensor ring model. Machine Learning: Science and Technology, 2020.

[3] Nir Ailon and Bernard Chazelle. The fast Johnson–Lindenstrauss transform and approximate nearest neighbors. SIAM Journal on Computing, 39(1):302–322, 2009.

[4] Hannah Alsdurf, Yoshua Bengio, Tristan Deleu, Prateek Gupta, Daphne Ippolito, Richard Janda, Max Jarvie, Tyler Kolody, Sekoul Krastev, and Tegan Maharaj. Covi white paper. arXiv preprint arXiv:2005.08502, 2020.

[5] Maliheh Aramon, Gili Rosenberg, Elisabetta Valiante, Toshiyuki Miyazawa, Hirotaka Tamura, and Helmut G. Katzgraber. Physics-inspired optimization for quadratic unconstrained problems using a digital annealer. Frontiers in Physics, 7:48, 2019.

[6] Kendall Atkinson and Weimin Han. Theoretical Numerical Analysis: A Functional Analysis Framework. Number 39 in Texts in Applied Mathematics. Springer-Verlag, New York, 3rd edition, 2009. ISBN 978-1-4419-0457-7.

[7] Woody Austin, Grey Ballard, and Tamara G. Kolda. Parallel tensor compression for large-scale scientific data. In 2016 IEEE International Parallel and Distributed Processing Symposium (IPDPS), pages 912–922. IEEE, 2016.

[8] Haim Avron, Petar Maymounkov, and Sivan Toledo. Blendenpik: Supercharging LAPACK's least-squares solver. SIAM Journal on Scientific Computing, 32(3):1217–1236, 2010.

[9] Haim Avron, Huy L. Nguyen, and David P. Woodruff. Subspace Embeddings for the Polynomial Kernel. In Proceedings of the 27th International Conference on Neural Information Processing Systems - Volume 2, pages 2258–2266, Cambridge, MA, USA, 2014. MIT Press.

[10] Brett W. Bader and Tamara G. Kolda. Algorithm 862: MATLAB tensor classes for fast algorithm prototyping. ACM Transactions on Mathematical Software (TOMS), 32(4):635–653, 2006.

[11] Brett W. Bader, Tamara G. Kolda, and others. MATLAB Tensor Toolbox, Version 2.6. Available online at https://www.tensortoolbox.org, 2015.

[12] G. Ballard, A. Benson, A. Druinsky, B. Lipshitz, and O. Schwartz. Improving the Numerical Stability of Fast Matrix Multiplication. SIAM Journal on Matrix Analysis and Applications, 37(4):1382–1418, January 2016. ISSN 0895-4798. doi: 10.1137/15M1032168.

[13] Eduard Bartl, Radim Belohlavek, Petr Osicka, and Hana Řezanková. Dimensionality reduction in boolean data: Comparison of four bmf methods. In International Workshop on Clustering High-Dimensional Data, pages 118–133. Springer, 2012.

[14] Muthu Baskaran, Benoît Meister, Nicolas Vasilache, and Richard Lethin. Efficient and scalable computations with sparse tensors. In 2012 IEEE Conference on High Performance Extreme Computing, pages 1–6. IEEE, 2012.

[15] 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.

[16] Christian Bauckhage, E. Brito, K. Cvejoski, C. Ojeda, Rafet Sifa, and S. Wrobel. Ising Models for Binary Clustering via Adiabatic Quantum Computing. In Marcello Pelillo and Edwin Hancock, editors, Energy Minimization Methods in Computer Vision and Pattern Recognition, Lecture Notes in Computer Science, pages 3–17, Cham, 2018. Springer International Publishing. ISBN 978-3-319-78199-0. doi: 10.1007/978-3-319-78199-0_1.

[17] Johann A. Bengua, Ho N. Phien, and Hoang D. Tuan. Optimal feature extraction and classification of tensors via matrix product state decomposition. In 2015 IEEE International Congress on Big Data, pages 669–672. IEEE, 2015.

[18] Austin R. Benson and Grey Ballard. A framework for practical parallel fast matrix multiplication. In ACM SIGPLAN Notices, volume 50, pages 42–53. ACM, 2015.

[19] Tanya Y. Berger-Wolf and Jared Saia. A framework for analysis of dynamic social networks. In Proceedings of the 12th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pages 523–528. ACM, 2006.

[20] Gregory Beylkin and Martin J. Mohlenkamp. Numerical operator calculus in higher dimensions. Proceedings of the National Academy of Sciences, 99(16):10246–10251, 2002.

[21] Gregory Beylkin and Martin J. Mohlenkamp. Algorithms for Numerical Analysis in High Dimensions. SIAM Journal on Scientific Computing, 26(6):2133–2159, July 2006.

[22] David J. Biagioni, Daniel Beylkin, and Gregory Beylkin. Randomized interpolative decomposition of separated representations. Journal of Computational Physics, 281(C):116–134, January 2015. ISSN 0021-9991. doi: 10.1016/j.jcp.2014.10.009.

[23] Jacob Bien, Ya Xu, and Michael W. Mahoney. CUR from a sparse optimization viewpoint. In Advances in Neural Information Processing Systems, pages 217–225, 2010.

[24] Dario Bini. Relations between exact and approximate bilinear algorithms. Applications. Calcolo, 17(1):87–97, 1980.

[25] Dario Bini and Grazia Lotti. Stability of Fast Algorithms for Matrix Multiplication. Numerische Mathematik, 36(1):63–72, 1980.

[26] Dario Bini, Milvio Capovani, Francesco Romani, and Grazia Lotti. $O(n^{2.7799})$ complexity for $n \times n$ approximate matrix multiplication. Information Processing Letters, 8(5):234–235, June 1979. ISSN 0020-0190. doi: 10.1016/0020-0190(79)90113-3.

[27] Dario Bini, Grazia Lotti, and Francesco Romani. Approximate solutions for the bilinear form computational problem. SIAM Journal on Computing, 9(4):692–697, 1980.

[28] Meltzer Bittner, Paul Meltzer, Yidong Chen, Youfei Jiang, Elisabeth Seftor, M. Hendrix, M. Radmacher, Richard Simon, Zohar Yakhini, and Amir Ben-Dor. Molecular classification of cutaneous malignant melanoma by gene expression profiling. Nature, 406(6795):536–540, 2000.

[29] Markus Bläser. On the complexity of the multiplication of matrices of small formats. Journal of Complexity, 19(1):43–60, 2003.

[30] Ajinkya Borle, Vincent E. Elfving, and Samuel J. Lomonaco. Quantum Approximate Optimization for Hard Problems in Linear Algebra. arXiv preprint arXiv:2006.15438, 2020.

[31] Christos Boutsidis and David P. Woodruff. Optimal CUR matrix decompositions. SIAM Journal on Computing, 46(2):543–589, 2017.

[32] Christos Boutsidis, Michael W. Mahoney, and Petros Drineas. An improved approximation algorithm for the column subset selection problem. In Proceedings of the Twentieth Annual ACM-SIAM Symposium on Discrete Algorithms, pages 968–977. SIAM, 2009.

[33] Christos Boutsidis, Petros Drineas, and Malik Magdon-Ismail. Near-Optimal Column-Based Matrix Reconstruction. SIAM Journal on Computing, 43(2):687–717, April 2014.

[34] Karen Braman. Third-order tensors as linear operators on a space of matrices. Linear Algebra and its Applications, 433(7):1241–1253, 2010.

[35] Richard P. Brent. Algorithms for matrix multiplication. Technical Report STAN-CS-70-157, Stanford University, 1970.

[36] Richard P. Brent. Error Analysis of Algorithms for Matrix Multiplication and Triangular Decomposition using Winograd's Identity. Numerische Mathematik, 16(2):145–156, 1970.

[37] Peter Bürgisser, Michael Clausen, and Mohammad A. Shokrollahi. Algebraic Complexity Theory, volume 315. Springer Science & Business Media, 2013.

[38] Joan Bruna, Wojciech Zaremba, Arthur Szlam, and Yann LeCun. Spectral networks and locally connected networks on graphs. arXiv preprint arXiv:1312.6203, 2013.

[39] Jean-Philippe Brunet, Pablo Tamayo, Todd R. Golub, and Jill P. Mesirov. Metagenes and molecular pattern discovery using matrix factorization. Proceedings of the national academy of sciences, 101(12):4164–4169, 2004.

[40] Aydin Buluc, Tamara G. Kolda, Stefan M. Wild, Mihai Anitescu, Anthony DeGennaro, John Jakeman, Chandrika Kamath, Ramakrishnan, Kannan, Miles E. Lopes, Per-Gunnar Martinsson, Kary Myers, Jelani Nelson, Juan M. Restrepo, C. Seshadhri, Draguna Vrabie, Brendt Wohlberg, Stephen J. Wright, Chao Yang, and Peter Zwart. Randomized algorithms for scientific computing (RASC). 2021.

[41] Hongyun Cai, Vincent W. Zheng, and Kevin Chen-Chuan Chang. A comprehensive survey of graph embedding: Problems, techniques, and applications. IEEE Transactions on Knowledge and Data Engineering, 30(9):1616–1637, 2018.

[42] Cesar F. Caiafa and Andrzej Cichocki. Generalizing the column–row matrix decomposition to multi-way arrays. Linear Algebra and its Applications, 433(3):557–573, 2010.

[43] Rick R. Castrapel and John L. Gustafson. Precision improvement method for the Strassen/Winograd matrix multiplication method. U.S. Patent No., 7209939B2:1–11, April 2007.

[44] Moses Charikar, Kevin Chen, and Martin Farach-Colton. Finding Frequent Items in Data Streams. Theoretical Computer Science, 312(1):3–15, January 2004. ISSN 0304-3975. doi: 10.1016/S0304-3975(03)00400-6.

[45] 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.

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

[47] Hongwei Cheng, Zydrunas Gimbutas, Per-Gunnar Martinsson, and Vladimir Rokhlin. On the Compression of Low Rank Matrices. SIAM Journal on Scientific Computing, 26(4):1389–1404, March 2005. ISSN 1064-8275. doi: 10.1137/030602678.

[48] Andrzej Cichocki, Namgil Lee, Ivan Oseledets, Anh-Huy Phan, Qibin Zhao, and Danilo P. Mandic. Tensor networks for dimensionality reduction and large-scale optimization: Part 1 low-rank tensor decompositions. Foundations and Trends® in Machine Learning, 9(4-5): 249–429, 2016.

[49] Andrzej Cichocki, Anh-Huy Phan, Qibin Zhao, Namgil Lee, Ivan Oseledets, Masashi Sugiyama, and Danilo P. Mandic. Tensor networks for dimensionality reduction and large-scale optimization: Part 2 applications and future perspectives. Foundations and Trends® in Machine Learning, 9(6):431–673, 2017.

[50] Kenneth L. Clarkson and David P. Woodruff. Low-Rank Approximation and Regression in Input Sparsity Time. Journal of the ACM, 63(6):54:1–54:45, February 2017. ISSN 0004-5411. doi: 10.1145/3019134.

[51] Eldan Cohen, Avradip Mandal, Hayato Ushijima-Mwesigwa, and Arnab Roy. Ising-based consensus clustering on specialized hardware. In International Symposium on Intelligent Data Analysis, pages 106–118. Springer, 2020.

[52] Nadav Cohen, Or Sharir, and Amnon Shashua. On the expressive power of deep learning: A tensor analysis. In Conference on Learning Theory, pages 698–728, 2016.

[53] Michele N. da Costa, Renato R. Lopes, and Joao Marcos T. Romano. Randomized methods for higher-order subspace separation. In 2016 24th European Signal Processing Conference (EUSIPCO), pages 215–219. IEEE, 2016.

[54] Sanjoy Dasgupta and Anupam Gupta. An elementary proof of a theorem of Johnson and Lindenstrauss. Random Structures & Algorithms, 22(1):60–65, 2003.

[55] Himeshi De Silva, John L. Gustafson, and Weng-Fai Wong. Making Strassen Matrix Multiplication Safe. In 2018 IEEE 25th International Conference on High Performance Computing (HiPC), pages 173–182. IEEE, 2018.

[56] Fabrizio De Vico Fallani, Jonas Richiardi, Mario Chavez, and Sophie Achard. Graph analysis of functional brain networks: Practical issues in translational neuroscience. Philosophical Transactions of the Royal Society B: Biological Sciences, 369(1653):20130521, 2014.

[57] Michaël Defferrard, Xavier Bresson, and Pierre Vandergheynst. Convolutional neural networks on graphs with fast localized spectral filtering. In Advances in Neural Information Processing Systems, pages 3844–3852, 2016.

[58] James Demmel, Ioana Dumitriu, and Olga Holtz. Fast linear algebra is stable. Numerische Mathematik, 108(1):59–91, 2007.

[59] James Demmel, Ioana Dumitriu, Olga Holtz, and Robert Kleinberg. Fast matrix multiplication is stable. Numerische Mathematik, 106(2):199–224, 2007.

[60] Derek DeSantis, Erik Skau, and Boian Alexandrov. Factorizations of Binary Matrices–Rank Relations and the Uniqueness of Boolean Decompositions. arXiv preprint arXiv:2012.10496, 2020.

[61] Amit Deshpande and Luis Rademacher. Efficient volume sampling for row/column subset selection. In 2010 IEEE 51st Annual Symposium on Foundations of Computer Science, pages 329–338. IEEE, 2010.

[62] Amit Deshpande and Santosh Vempala. Adaptive sampling and fast low-rank matrix approximation. In Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques, pages 292–303. Springer, 2006.

[63] Amit Deshpande, Luis Rademacher, Santosh Vempala, and Grant Wang. Matrix approximation and projective clustering via volume sampling. In Proceedings of the Seventeenth Annual ACM-SIAM Symposium on Discrete Algorithm, pages 1117–1126. Society for Industrial and Applied Mathematics, 2006.

[64] Huaian Diao, Zhao Song, Wen Sun, and David Woodruff. Sketching for Kronecker Product Regression and P-splines. In Proceedings of the 21st International Conference on Artificial Intelligence and Statistics, pages 1299–1308, 2018.

[65] Huaian Diao, Rajesh Jayaram, Zhao Song, Wen Sun, and David P. Woodruff. Optimal Sketching for Kronecker Product Regression and Low Rank Approximation. arXiv preprint arXiv:1909.13384, 2019.

[66] Mamadou Diop, Anthony Larue, Sebastian Miron, and David Brie. A post-nonlinear mixture model approach to binary matrix factorization. In 2017 25th European Signal Processing Conference (EUSIPCO), pages 321–325. IEEE, 2017.

[67] Petros Drineas and Ravi Kannan. Pass efficient algorithms for approximating large matrices. In Proceedings of the Fourteenth Annual ACM-SIAM Symposium on Discrete Algorithms, pages 223–232, Baltimore, Maryland, January 2003. Society for Industrial and Applied Mathematics.

[68] Petros Drineas and Michael W. Mahoney. A randomized algorithm for a tensor-based generalization of the singular value decomposition. Linear algebra and its applications, 420 (2-3):553–571, 2007.

[69] Petros Drineas, Ravi Kannan, and Michael W. Mahoney. Fast Monte Carlo algorithms for matrices III: Computing a compressed approximate matrix decomposition. SIAM Journal on Computing, 36(1):184–206, 2006.

[70] 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.

[71] Petros Drineas, Michael W. Mahoney, and S. Muthukrishnan. Relative-error CUR matrix decompositions. SIAM Journal on Matrix Analysis and Applications, 30(2):844–881, 2008.

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

[73] Bogdan Dumitrescu. Improving and estimating the accuracy of Strassen's algorithm. Numerische Mathematik, 79(4):485–499, 1998.

[74] Oylum Şeker, Neda Tanoumand, and Merve Bodur. Digital Annealer for quadratic unconstrained binary optimization: A comparative performance analysis. arXiv preprint arXiv:2012.12264, 2020.

[75] Veit Elser. A network that learns strassen multiplication. The Journal of Machine Learning Research, 17(1):3964–3976, 2016.

[76] Mike Espig, Kishore Kumar Naraparaju, and Jan Schneider. A note on tensor chain approximation. Computing and Visualization in Science, 15(6):331–344, 2012.

[77] Hadi Fanaee-T and João Gama. Multi-aspect-streaming tensor analysis. Knowledge-Based Systems, 89:332–345, 2015.

[78] Shmuel Friedland, Volker Mehrmann, Agnieszka Miedlar, and M. Nkengla. Fast low rank approximations of matrices and tensors. Electronic Journal of Linear Algebra, 22:1031–1048, 2011. ISSN 1081-3810. doi: 10.13001/1081-3810.1489.

[79] Alan Frieze, Ravi Kannan, and Santosh Vempala. Fast Monte-Carlo algorithms for finding low-rank approximations. Journal of the ACM (JACM), 51(6):1025–1041, 2004.

[80] Timur Garipov, Dmitry Podoprikhin, Alexander Novikov, and Dmitry Vetrov. Ultimate tensorization: Compressing convolutional and fc layers alike. arXiv preprint arXiv:1611.03214, 2016.

[81] Justin Gilmer, Samuel S. Schoenholz, Patrick F. Riley, Oriol Vinyals, and George E. Dahl. Neural message passing for quantum chemistry. In International Conference on Machine Learning, pages 1263–1272. PMLR, 2017.

[82] Fred Glover, Gary Kochenberger, and Yu Du. A tutorial on formulating and using QUBO models. arXiv preprint arXiv:1811.11538, 2018.

[83] Gene H. Golub and Charles F. Van Loan. Matrix Computations. Johns Hopkins University Press, Baltimore, fourth edition, 2013. ISBN 978-1-4214-0794-4.

[84] Sergey A. Goreinov, Eugene E. Tyrtyshnikov, and Nickolai L. Zamarashkin. A theory of pseudoskeleton approximations. Linear Algebra and its Applications, 261(1-3):1–21, August 1997.

[85] Sergey A. Goreinov, Eugene E. Tyrtyshnikov, and Nickolai L. Zamarashkin. Pseudo-skeleton approximations by matrices of maximal volume. Mathematical Notes, 62(4):515–519, October 1997.

[86] Palash Goyal, Nitin Kamra, Xinran He, and Yan Liu. Dyngem: Deep embedding method for dynamic graphs. arXiv preprint arXiv:1805.11273, 2018.

[87] Ming Gu and Stanley C. Eisenstat. Efficient Algorithms for Computing a Strong Rank-Revealing QR Factorization. SIAM Journal on Scientific Computing, 17(4):848–869, July 1996. ISSN 1064-8275. doi: 10.1137/0917055.

[88] Ekta Gujral, Ravdeep Pasricha, and Evangelos E. Papalexakis. Sambaten: Sampling-based batch incremental tensor decomposition. In Proceedings of the 2018 SIAM International Conference on Data Mining, pages 387–395. SIAM, 2018.

[89] Suyog Gupta, Ankur Agrawal, Kailash Gopalakrishnan, and Pritish Narayanan. Deep learning with limited numerical precision. In International Conference on Machine Learning, pages 1737–1746, 2015.

[90] Venkatesan Guruswami and Ali Kemal Sinop. Optimal column-based low-rank matrix reconstruction. In Proceedings of the Twenty-Third Annual ACM-SIAM Symposium on Discrete Algorithms, pages 1207–1214. SIAM, 2012.

[91] Nathan Halko, Per-Gunnar Martinsson, and Joel A. Tropp. Finding Structure with Randomness: Probabilistic Algorithms for Constructing Approximate Matrix Decompositions. SIAM Review, 53(2):217–288, May 2011.

[92] W. Keith Hastings. Monte Carlo sampling methods using Markov chains and their applications. Biometrika, 57(1):97–109, April 1970. ISSN 0006-3444. doi: 10.1093/biomet/57.1.97.

[93] Tamir Hazan, Simon Polak, and Amnon Shashua. Sparse image coding using a 3D non-negative tensor factorization. In Tenth IEEE International Conference on Computer Vision (ICCV'05) Volume 1, volume 1, pages 50–57. IEEE, 2005.

[94] Sibylle Hess, Katharina Morik, and Nico Piatkowski. The PRIMPING routine—Tiling through proximal alternating linearized minimization. Data Mining and Knowledge Discovery, 31(4): 1090–1131, July 2017. ISSN 1573-756X. doi: 10.1007/s10618-017-0508-z.

[95] Michael Hopkins, Mantas Mikaitis, Dave R. Lester, and Steve Furber. Stochastic rounding and reduced-precision fixed-point arithmetic for solving neural ODEs. arXiv preprint arXiv:1904.11263, 2019.

[96] Roger A. Horn and Charles R. Johnson. Topics in Matrix Analysis. Cambridge University Press, 1994.

[97] Roger A. Horn and Charles R. Johnson. Matrix Analysis. Cambridge university press, 2012.

[98] Ming Hou, Jiajia Tang, Jianhai Zhang, Wanzeng Kong, and Qibin Zhao. Deep multimodal multilinear fusion with high-order polynomial pooling. In Advances in Neural Information Processing Systems, pages 12136–12145, 2019.

[99] Jianyu Huang, Chenhan D. Yu, and Robert A. van de Geijn. Implementing Strassen's Algorithm with CUTLASS on NVIDIA Volta GPUs. arXiv:1808.07984 [cs], August 2018.

[100] Benjamin Huber, Reinhold Schneider, and Sebastian Wolf. A randomized tensor train singular value decomposition. In Compressed Sensing and Its Applications, pages 261–290. Springer, 2017.

[101] Steven Huss-Lederman, Elaine M. Jacobson, Jeremy R. Johnson, Anna Tsao, and Thomas Turnbull. Implementation of Strassen's algorithm for matrix multiplication. In Supercomputing'96: Proceedings of the 1996 ACM/IEEE Conference on Supercomputing, pages 32–32. IEEE, 1996.

[102] M. A. Iwen, D. Needell, E. Rebrova, and A. Zare. Lower Memory Oblivious (Tensor) Subspace Embeddings with Fewer Random Bits: Modewise Methods for Least Squares. arXiv preprint arXiv:1912.08294, 2019.

[103] Pavel Izmailov, Alexander Novikov, and Dmitry Kropotov. Scalable gaussian processes with billions of inducing inputs via tensor train decomposition. In International Conference on Artificial Intelligence and Statistics, pages 726–735, 2018.

[104] Inah Jeon, Evangelos E. Papalexakis, Uksong Kang, and Christos Faloutsos. Haten2: Billion-scale tensor decompositions. In 2015 IEEE 31st International Conference on Data Engineering, pages 1047–1058. IEEE, 2015.

[105] Ruhui Jin, Tamara G Kolda, and Rachel Ward. Faster Johnson-Lindenstrauss transforms via kronecker products. Information and Inference: A Journal of the IMA, October 2020. ISSN 2049-8772. doi: 10.1093/imaiai/iaaa028.

[106] Rodney W. Johnson and Aileen M. McLoughlin. Noncommutative bilinear algorithms for 3×3 matrix multiplication. SIAM Journal on Computing, 15(2):595–603, 1986.

[107] William B. Johnson and Joram Lindenstrauss. Extensions of Lipschitz mappings into a Hilbert space. Contemporary Mathematics, 26(189-206):1, 1984.

[108] Igor Kaporin. The aggregation and cancellation techniques as a practical tool for faster matrix multiplication. Theoretical Computer Science, 315(2-3):469–510, 2004.

[109] Oguz Kaya and Bora Uçar. High performance parallel algorithms for the tucker decomposition of sparse tensors. In 2016 45th International Conference on Parallel Processing (ICPP), pages 103–112. IEEE, 2016.

[110] Eric Kernfeld, Misha Kilmer, and Shuchin Aeron. Tensor–tensor products with invertible linear transforms. Linear Algebra and its Applications, 485:545–570, 2015.

[111] Yuehaw Khoo, Jianfeng Lu, and Lexing Ying. Efficient construction of tensor ring representations from sampling. arXiv preprint arXiv:1711.00954, 2019.

[112] Valentin Khrulkov, Alexander Novikov, and Ivan Oseledets. Expressive power of recurrent neural networks. In International Conference on Learning Representations, 2018.

[113] Misha E. Kilmer, Karen Braman, Ning Hao, and Randy C. Hoover. Third-order tensors as operators on matrices: A theoretical and computational framework with applications in imaging. SIAM Journal on Matrix Analysis and Applications, 34(1):148–172, 2013.

[114] Thomas N. Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. arXiv preprint arXiv:1609.02907, 2016.

[115] Tamara G. Kolda. Multilinear operators for higher-order decompositions. Sandia Report SAND2006-2081, April 2006.

[116] Tamara G. Kolda and Brett W. Bader. Tensor Decompositions and Applications. SIAM Review, 51(3):455–500, August 2009. ISSN 0036-1445. doi: 10.1137/07070111X.

[117] Tamara G. Kolda and Jimeng Sun. Scalable tensor decompositions for multi-aspect data mining. In 2008 Eighth IEEE International Conference on Data Mining, pages 363–372. IEEE, 2008.

[118] Reka A. Kovacs, Oktay Gunluk, and Raphael A. Hauser. Binary Matrix Factorisation via Column Generation. arXiv preprint arXiv:2011.04457, 2020.

[119] Mehmet Koyutürk and Ananth Grama. PROXIMUS: A framework for analyzing very high dimensional discrete-attributed datasets. In Proceedings of the Ninth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pages 147–156, 2003.

[120] Ravi Kumar, Rina Panigrahy, Ali Rahimi, and David Woodruff. Faster algorithms for binary matrix factorization. In International Conference on Machine Learning, pages 3551–3559, 2019.

[121] Srijan Kumar, William L. Hamilton, Jure Leskovec, and Dan Jurafsky. Community interaction and conflict on the web. In Proceedings of the 2018 World Wide Web Conference, pages 933–943. International World Wide Web Conferences Steering Committee, 2018.

[122] Jérôme Kunegis. Konect: The koblenz network collection. In Proceedings of the 22nd International Conference on World Wide Web, pages 1343–1350. ACM, 2013.

[123] Julian D. Laderman. A noncommutative algorithm for multiplying 3x3 matrices using 23 multiplications. Bulletin of the American Mathematical Society, 82(1):126–128, 1976. ISSN 0002-9904, 1936-881X. doi: 10.1090/S0002-9904-1976-13988-2.

[124] Brett W. Larsen and Tamara G. Kolda. Practical Leverage-Based Sampling for Low-Rank Tensor Decomposition. arXiv preprint arXiv:2006.16438, 2020.

[125] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. Proceedings of the IEEE, 86(11):2278–2324, 1998.

[126] Tao Lei, Yu Xin, Yuan Zhang, Regina Barzilay, and Tommi Jaakkola. Low-rank tensors for scoring dependency structures. In Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), pages 1381–1391, 2014.

[127] Jiajia Li, Casey Battaglino, Ioakeim Perros, Jimeng Sun, and Richard Vuduc. An input-adaptive and in-place approach to dense tensor-times-matrix multiply. In SC'15: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis, pages 1–12. IEEE, 2015.

[128] Jiajia Li, Yuchen Ma, Chenggang Yan, and Richard Vuduc. Optimizing sparse tensor times matrix on multi-core and many-core architectures. In 2016 6th Workshop on Irregular Applications: Architecture and Algorithms (IA3), pages 26–33. IEEE, 2016.

[129] Jundong Li, Harsh Dani, Xia Hu, Jiliang Tang, Yi Chang, and Huan Liu. Attributed network embedding for learning in a dynamic environment. In Proceedings of the 2017 ACM on Conference on Information and Knowledge Management, pages 387–396, 2017.

[130] Yaguang Li, Rose Yu, Cyrus Shahabi, and Yan Liu. Diffusion convolutional recurrent neural network: Data-driven traffic forecasting. arXiv preprint arXiv:1707.01926, 2017.

[131] Lifan Liang, Kunju Zhu, and Songjian Lu. BEM: Mining Coregulation Patterns in Transcriptomics via Boolean Matrix Factorization. Bioinformatics, 36(13):4030–4037, 2020.

[132] Edo Liberty, Franco Woolfe, Per-Gunnar Martinsson, Vladimir Rokhlin, and Mark Tygert. Randomized algorithms for the low-rank approximation of matrices. Proceedings of the National Academy of Sciences, 104(51):20167–20172, December 2007. ISSN 0027-8424, 1091-6490. doi: 10.1073/pnas.0709640104.

[133] Bangtian Liu, Chengyao Wen, Anand D. Sarwate, and Maryam Mehri Dehnavi. A unified optimization approach for sparse tensor operations on GPUs. In 2017 IEEE International Conference on Cluster Computing (CLUSTER), pages 47–57. IEEE, 2017.

[134] Ding Liu, Shi-Ju Ran, Peter Wittek, Cheng Peng, Raul Blázquez García, Gang Su, and Maciej Lewenstein. Machine learning by unitary tensor network of hierarchical tree structure. New Journal of Physics, 21(7):073059, 2019.

[135] Xien Liu, Xinxin You, Xiao Zhang, Ji Wu, and Ping Lv. Tensor graph convolutional networks for text classification. In Proceedings of the AAAI Conference on Artificial Intelligence, volume 34, pages 8409–8416, 2020.

[136] Claudio Lucchese, Salvatore Orlando, and Raffaele Perego. Mining top-k patterns from binary datasets in presence of noise. In Proceedings of the 2010 SIAM International Conference on Data Mining, pages 165–176. SIAM, 2010.

[137] Malik Magdon-Ismail. Row sampling for matrix algorithms via a non-commutative Bernstein bound. arXiv preprint arXiv:1008.0587, 2010.

[138] Michael W. Mahoney. Randomized algorithms for matrices and data. Foundations and Trends in Machine Learning, 3(2):123–224, 2011.

[139] Michael W. Mahoney and Petros Drineas. CUR matrix decompositions for improved data analysis. Proceedings of the National Academy of Sciences, 106(3):697–702, 2009.

[140] Michael W. Mahoney, Mauro Maggioni, and Petros Drineas. Tensor-CUR decompositions for tensor-based data. SIAM Journal on Matrix Analysis and Applications, 30(3):957–987, 2008.

[141] Konstantin Makarychev, Yury Makarychev, and Ilya Razenshteyn. Performance of Johnson–Lindenstrauss transform for k-means and k-medians clustering. In Proceedings of the 51st Annual ACM SIGACT Symposium on Theory of Computing, pages 1027–1038. ACM, 2019.

[142] Osman Asif Malik and Stephen Becker. Low-Rank Tucker Decomposition of Large Tensors Using TensorSketch. In Advances in Neural Information Processing Systems, pages 10096–10106, 2018.

[143] Osman Asif Malik and Stephen Becker. Guarantees for the Kronecker fast Johnson–Lindenstrauss transform using a coherence and sampling argument. Linear Algebra and its Applications, 602:120–137, 2020. ISSN 0024-3795. doi: 10.1016/j.laa.2020.05.004.

[144] Osman Asif Malik and Stephen Becker. Fast randomized matrix and tensor interpolative decomposition using CountSketch. Advances in Computational Mathematics, 46(6):76, 2020. ISSN 1572-9044. doi: 10.1007/s10444-020-09816-9.

[145] Osman Asif Malik and Stephen Becker. A sampling-based method for tensor ring decomposition. arXiv preprint arXiv:2010.08581, 2021.

[146] Osman Asif Malik and Stephen Becker. Randomization of approximate bilinear computation for matrix multiplication. International Journal of Computer Mathematics: Computer Systems Theory, 6(1):54–93, 2021. doi: 10.1080/23799927.2020.1861104.

[147] Osman Asif Malik, Hayato Ushijima-Mwesigwa, Arnab Roy, Avradip Mandal, and Indradeep Ghosh. Binary Matrix Factorization on Special Purpose Hardware. arXiv preprint arXiv:2010.08693, 2020.

[148] Osman Asif Malik, Shashanka Ubaru, Lior Horesh, Misha E. Kilmer, and Haim Avron. Dynamic graph convolutional networks using the tensor M-product. In Proceedings of the 2021 SIAM International Conference on Data Mining (SDM), pages 729–737. 2021. doi: 10.1137/1.9781611976700.82.

[149] Franco Manessi, Alessandro Rozza, and Mario Manzo. Dynamic graph convolutional networks. Pattern Recognition, 97:107000, 2020. ISSN 0031-3203. doi: https://doi.org/10.1016/j.patcog.2019.107000.

[150] Per-Gunnar Martinsson, Vladimir Rokhlin, and Mark Tygert. A randomized algorithm for the decomposition of matrices. Applied and Computational Harmonic Analysis, 30(1):47–68, January 2011. ISSN 1063-5203. doi: 10.1016/j.acha.2010.02.003.

[151] Effrosyni Mavroudi, Benjamín Béjar Haro, and René Vidal. Representation Learning on Visual-Symbolic Graphs for Video Understanding. In European Conference on Computer Vision, pages 71–90. Springer, 2020.

[152] Nicholas Metropolis, Arianna W. Rosenbluth, Marshall N. Rosenbluth, Augusta H. Teller, and Edward Teller. Equation of state calculations by fast computing machines. The journal of chemical physics, 21(6):1087–1092, 1953.

[153] Oscar Mickelin and Sertac Karaman. On algorithms for and computing with the tensor ring decomposition. Numerical Linear Algebra with Applications, 27(3):e2289, 2020.

[154] Pauli Miettinen and Jilles Vreeken. Model order selection for boolean matrix factorization. In Proceedings of the 17th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pages 51–59, 2011.

[155] Pauli Miettinen, Taneli Mielikäinen, Aristides Gionis, Gautam Das, and Heikki Mannila. The discrete basis problem. IEEE transactions on knowledge and data engineering, 20(10): 1348–1362, 2008.

[156] Rachel Minster, Arvind K. Saibaba, and Misha E. Kilmer. Randomized algorithms for low-rank tensor decompositions in the Tucker format. SIAM Journal on Mathematics of Data Science, 2(1):189–215, 2020.

[157] Christian F. A. Negre, Hayato Ushijima-Mwesigwa, and Susan M. Mniszewski. Detecting multiple communities using quantum annealing on the D-Wave system. Plos one, 15(2): e0227538, 2020.

[158] Sameer A. Nene, Shree K. Nayar, and Hiroshi Murase. Columbia Object Image Library (COIL-100). Technical Report CUCS-006-96, Columbia University, 1996.

[159] Elizabeth Newman, Lior Horesh, Haim Avron, and Misha Kilmer. Stable Tensor Neural Networks for Rapid Deep Learning. arXiv preprint arXiv:1811.06569, 2018.

[160] Giang Hoang Nguyen, John Boaz Lee, Ryan A. Rossi, Nesreen K. Ahmed, Eunyee Koh, and Sungchul Kim. Continuous-time dynamic network embeddings. In Companion Proceedings of the The Web Conference 2018, pages 969–976, 2018.

[161] Alexander Novikov, Dmitrii Podoprikhin, Anton Osokin, and Dmitry P. Vetrov. Tensorizing neural networks. In Advances in Neural Information Processing Systems, pages 442–450, 2015.

[162] Alexander Novikov, Mikhail Trofimov, and Ivan Oseledets. Exponential machines. arXiv preprint arXiv:1605.03795, 2017.

[163] Jinoh Oh, Kijung Shin, Evangelos E. Papalexakis, Christos Faloutsos, and Hwanjo Yu. S-hot: Scalable high-order tucker decomposition. In Proceedings of the Tenth ACM International Conference on Web Search and Data Mining, pages 761–770, 2017.

[164] Daniel O'Malley and Velimir V. Vesselinov. ToQ. jl: A high-level programming language for D-Wave machines based on Julia. In 2016 IEEE High Performance Extreme Computing Conference (HPEC), pages 1–7. IEEE, 2016.

[165] Daniel O'Malley, Velimir V. Vesselinov, Boian S. Alexandrov, and Ludmil B. Alexandrov. Nonnegative/binary matrix factorization with a d-wave quantum annealer. PloS one, 13(12): e0206653, 2018.

[166] Ivan Oseledets and Eugene Tyrtyshnikov. TT-cross approximation for multidimensional arrays. Linear Algebra and its Applications, 432(1):70–88, 2010.

[167] Ivan V. Oseledets. Tensor-train decomposition. SIAM Journal on Scientific Computing, 33 (5):2295–2317, 2011.

[168] Ivan V. Oseledets, D. V. Savostianov, and Eugene E. Tyrtyshnikov. Tucker dimensionality reduction of three-dimensional arrays in linear time. SIAM Journal on Matrix Analysis and Applications, 30(3):939–956, 2008.

[169] Daniele Ottaviani and Alfonso Amendola. Low rank non-negative matrix factorization with d-wave 2000q. arXiv preprint arXiv:1808.08721, 2018.

[170] Rasmus Pagh. Compressed Matrix Multiplication. ACM Transactions on Computation Theory, 5(3):9:1–9:17, August 2013. ISSN 1942-3454. doi: 10.1145/2493252.2493254.

[171] Evangelos E. Papalexakis, Christos Faloutsos, and Nicholas D. Sidiropoulos. Tensors for data mining and data fusion: Models, applications, and scalable algorithms. ACM Transactions on Intelligent Systems and Technology (TIST), 8(2):1–44, 2016.

[172] Panos M. Pardalos, Ding-Zhu Du, and Ronald L. Graham. Handbook of Combinatorial Optimization. Springer, 2013.

[173] Aldo Pareja, Giacomo Domeniconi, Jie Chen, Tengfei Ma, Toyotaro Suzumura, Hiroki Kanezashi, Tim Kaler, and Charles E. Leisersen. Evolvegcn: Evolving graph convolutional networks for dynamic graphs. arXiv preprint arXiv:1902.10191, 2019.

[174] Ninh Pham and Rasmus Pagh. Fast and Scalable Polynomial Kernels via Explicit Feature Maps. In Proceedings of the 19th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD '13, pages 239–247, New York, NY, USA, 2013. ACM. ISBN 978-1-4503-2174-7. doi: 10.1145/2487575.2487591.

[175] Beheshteh T. Rakhshan and Guillaume Rabusseau. Tensorized Random Projections. arXiv preprint arXiv:2003.05101, 2020.

[176] Ignacio Ramírez. Binary matrix factorization via dictionary learning. IEEE journal of selected topics in signal processing, 12(6):1253–1262, 2018.

[177] Siamak Ravanbakhsh, Barnabás Póczos, and Russell Greiner. Boolean Matrix Factorization and Noisy Completion via Message Passing. In ICML, volume 69, pages 945–954, 2016.

[178] Sidney I. Resnick. A Probability Path. Modern Birkhäuser Classics. Birkhäuser Basel, 2014. ISBN 978-0-8176-8408-2.

[179] Matthew J. Reynolds, Alireza Doostan, and Gregory Beylkin. Randomized Alternating Least Squares for Canonical Tensor Decompositions: Application to A PDE With Random Data. SIAM Journal on Scientific Computing, 38(5):A2634–A2664, September 2016.

[180] Matthew J. Reynolds, Gregory Beylkin, and Alireza Doostan. Optimization via separated representations and the canonical tensor decomposition. Journal of Computational Physics, 348(C):220–230, November 2017.

[181] Roberto Rigamonti, Amos Sironi, Vincent Lepetit, and Pascal Fua. Learning separable filters. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pages 2754–2761, 2013.

[182] Aravind Sankar, Yanhong Wu, Liang Gou, Wei Zhang, and Hao Yang. Dynamic graph representation learning via self-attention networks. arXiv preprint arXiv:1812.09430, 2018.

[183] Martin Schaefer. Note on the k-dimensional Jensen inequality. The Annals of Probability, pages 502–504, 1976.

[184] Youngjoo Seo, Michaël Defferrard, Pierre Vandergheynst, and Xavier Bresson. Structured sequence modeling with graph convolutional recurrent networks. In International Conference on Neural Information Processing, pages 362–373. Springer, 2018.

[185] Ruslan Shaydulin, Hayato Ushijima-Mwesigwa, Ilya Safro, Susan Mniszewski, and Yuri Alexeev. Community detection across emerging quantum architectures. 3rd International Workshop on Post Moore's Era Supercomputing (PMES 2018), 2018.

[186] Bao-Hong Shen, Shuiwang Ji, and Jieping Ye. Mining discrete patterns via binary matrix factorization. In Proceedings of the 15th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pages 757–766, 2009.

[187] Yang Shi, Uma Naresh Niranjan, Animashree Anandkumar, and Cris Cecka. Tensor contractions with extended BLAS kernels on CPU and GPU. In 2016 IEEE 23rd International Conference on High Performance Computing (HiPC), pages 193–202. IEEE, 2016.

[188] A. V. Smirnov. The bilinear complexity and practical algorithms for matrix multiplication. Computational Mathematics and Mathematical Physics, 53(12):1781–1795, 2013. ISSN 1555-6662. doi: 10.1134/S0965542513120129.

[189] Shaden Smith, Kejun Huang, Nicholas D. Sidiropoulos, and George Karypis. Streaming tensor factorization for infinite data sources. In Proceedings of the 2018 SIAM International Conference on Data Mining, pages 81–89. SIAM, 2018.

[190] E. Miles Stoudenmire and David J. Schwab. Supervised Learning with Quantum-Inspired Tensor Networks. arXiv:1605.05775 [cond-mat, stat], May 2017.

[191] Volker Strassen. Gaussian Elimination is not Optimal. Numerische Mathematik, 13(4): 354–356, August 1969. ISSN 0029-599X. doi: 10.1007/BF02165411.

[192] Jimeng Sun, Dacheng Tao, Spiros Papadimitriou, Philip S. Yu, and Christos Faloutsos. Incremental tensor analysis: Theory and applications. ACM Transactions on Knowledge Discovery from Data (TKDD), 2(3):1–37, 2008.

[193] Yiming Sun, Yang Guo, Joel A. Tropp, and Madeleine Udell. Tensor random projection for low memory dimension reduction. In NeurIPS Workshop on Relational Representation Learning, 2018.

[194] 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.

[195] Robert H. Swendsen and Jian-Sheng Wang. Replica Monte Carlo simulation of spin-glasses. Physical review letters, 57(21):2607, 1986.

[196] Davoud Ataee Tarzanagh and George Michailidis. Fast Randomized Algorithms for t-Product Based Tensor Operations and Decompositions with Applications to Imaging Data. SIAM Journal on Imaging Sciences, 11(4):2629–2664, 2018.

[197] Rakshit Trivedi, Hanjun Dai, Yichen Wang, and Le Song. Know-evolve: Deep temporal reasoning for dynamic knowledge graphs. In International Conference on Machine Learning, pages 3462–3471. PMLR, 2017.

[198] Rakshit Trivedi, Mehrdad Farajtabar, Prasenjeet Biswal, and Hongyuan Zha. Dyrep: Learning representations over dynamic graphs. In International Conference on Learning Representations, 2019.

[199] Charalampos E. Tsourakakis. Mach: Fast randomized tensor decompositions. In Proceedings of the 2010 SIAM International Conference on Data Mining, pages 689–700. SIAM, 2010.

[200] Eugene E. Tyrtyshnikov. Incomplete Cross Approximation in the Mosaic-Skeleton Method. Computing, 64:367–380, 2000.

[201] Shashanka Ubaru, Lior Horesh, and Guy Cohen. Dynamic graph based epidemiological model for COVID-19 contact tracing data analysis and optimal testing prescription. arXiv preprint arXiv:2009.04971, 2020.

[202] Hayato Ushijima-Mwesigwa, Christian F. A. Negre, and Susan M. Mniszewski. Graph partitioning using quantum annealing on the D-Wave system. In Proceedings of the Second International Workshop on Post Moores Era Supercomputing, pages 22–29. ACM, 2017.

[203] Charles F. Van Loan. The ubiquitous Kronecker product. Journal of Computational and Applied Mathematics, 123(1):85–100, November 2000. ISSN 0377-0427. doi: 10.1016/S0377-0427(00)00393-9.

[204] M. Alex O. Vasilescu and Demetri Terzopoulos. Multilinear analysis of image ensembles: Tensorfaces. In European Conference on Computer Vision, pages 447–460. Springer, 2002.

[205] Roman Vershynin. High-Dimensional Probability: An Introduction with Applications in Data Science, volume 47. Cambridge University Press, 2018.

[206] Sergey Voronin and Per-Gunnar Martinsson. Efficient algorithms for CUR and interpolative matrix decompositions. Advances in Computational Mathematics, 43(3):495–516, June 2017.

[207] Naigang Wang, Jungwook Choi, Daniel Brand, Chia-Yu Chen, and Kailash Gopalakrishnan. Training deep neural networks with 8-bit floating point numbers. In Advances in Neural Information Processing Systems, pages 7675–7684, 2018.

[208] Shusen Wang and Zhihua Zhang. Improving CUR matrix decomposition and the Nyström approximation via adaptive sampling. The Journal of Machine Learning Research, 14(1): 2729–2769, 2013.

[209] Wenqi Wang, Vaneet Aggarwal, and Shuchin Aeron. Efficient low rank tensor ring completion. In Proceedings of the IEEE International Conference on Computer Vision, pages 5697–5705, 2017.

[210] Yining Wang, Hsiao-Yu Tung, Alexander J. Smola, and Anima Anandkumar. Fast and guaranteed tensor decomposition via sketching. In Advances in Neural Information Processing Systems, pages 991–999, 2015.

[211] Yue Wang, Yongbin Sun, Ziwei Liu, Sanjay E. Sarma, Michael M. Bronstein, and Justin M. Solomon. Dynamic graph cnn for learning on point clouds. arXiv preprint arXiv:1801.07829, 2018.

[212] David P. Woodruff. Sketching as a tool for numerical linear algebra. Foundations and Trends in Theoretical Computer Science, 10(1-2):1–157, 2014.

[213] Franco Woolfe, Edo Liberty, Vladimir Rokhlin, and Mark Tygert. A fast randomized algorithm for the approximation of matrices. Applied and Computational Harmonic Analysis, 25(3): 335–366, November 2008. ISSN 1063-5203. doi: 10.1016/j.acha.2007.12.002.

[214] Zonghan Wu, Shirui Pan, Fengwen Chen, Guodong Long, Chengqi Zhang, and Philip S. Yu. A comprehensive survey on graph neural networks. arXiv preprint arXiv:1901.00596, 2019.

[215] Bo Yang, Ahmed Zamzam, and Nicholas D. Sidiropoulos. ParaSketch: Parallel Tensor Factorization via Sketching. In Proceedings of the 2018 SIAM International Conference on Data Mining, pages 396–404. SIAM, 2018.

[216] Yinchong Yang, Denis Krompass, and Volker Tresp. Tensor-train recurrent neural networks for video classification. arXiv preprint arXiv:1707.01786, 2017.

[217] Jinmian Ye, Linnan Wang, Guangxi Li, Di Chen, Shandian Zhe, Xinqi Chu, and Zenglin Xu. Learning compact recurrent neural networks with block-term tensor decomposition. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pages 9378–9387, 2018.

[218] Ke Ye and Lek-Heng Lim. Tensor network ranks. arXiv preprint arXiv:1801.02662, 2018.

[219] Rose Yu, Stephan Zheng, Anima Anandkumar, and Yisong Yue. Long-term forecasting using higher order tensor RNNs. arXiv preprint arXiv:1711.00073, 2017.

[220] Longhao Yuan, Chao Li, Jianting Cao, and Qibin Zhao. Randomized tensor ring decomposition and its application to large-scale data reconstruction. In ICASSP 2019-2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), pages 2127–2131. IEEE, 2019.

[221] Jiani Zhang, Arvind K. Saibaba, Misha E. Kilmer, and Shuchin Aeron. A randomized tensor singular value decomposition based on the t-product. Numerical Linear Algebra with Applications, 25:e2179, 2018.

[222] Tong Zhang, Wenming Zheng, Zhen Cui, and Yang Li. Tensor graph convolutional neural network. arXiv preprint arXiv:1803.10071, 2018.

[223] Zhong-Yuan Zhang, Tao Li, Chris Ding, Xian-Wen Ren, and Xiang-Sun Zhang. Binary matrix factorization for analyzing gene expression data. Data Mining and Knowledge Discovery, 20 (1):28, 2010.

[224] Zhongyuan Zhang, Tao Li, Chris Ding, and Xiangsun Zhang. Binary matrix factorization with applications. In Seventh IEEE International Conference on Data Mining (ICDM 2007), pages 391–400. IEEE, 2007.

[225] Ling Zhao, Yujiao Song, Chao Zhang, Yu Liu, Pu Wang, Tao Lin, Min Deng, and Haifeng Li. T-GCN: A Temporal Graph Convolutional Network for Traffic Prediction. IEEE Transactions on Intelligent Transportation Systems, 2019.

[226] Qibin Zhao, Guoxu Zhou, Shengli Xie, Liqing Zhang, and Andrzej Cichocki. Tensor ring decomposition. arXiv preprint arXiv:1606.05535, 2016.

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

[228] Jie Zhou, Ganqu Cui, Zhengyan Zhang, Cheng Yang, Zhiyuan Liu, and Maosong Sun. Graph neural networks: A review of methods and applications. arXiv preprint arXiv:1812.08434, 2018.

[229] Lekui Zhou, Yang Yang, Xiang Ren, Fei Wu, and Yueting Zhuang. Dynamic network embedding by modeling triadic closure process. In Proceedings of the AAAI Conference on Artificial Intelligence, volume 32, 2018.

Appendix A

Supplementary material for Chapter 5

A.1 Additional experiments

First, we repeat the second experiment we did in Section 5.3.1 for the Strassen ABC with multiple Gaussian, uniform, and type 1–3 adversarial matrices. We use the same setup as in the main manuscript: For each experiment, we create an approximate algorithm by perturbing Strassen's algorithm, we draw random matrices $\boldsymbol{A}, \boldsymbol{B} \in \mathbb{R}^{320\times 320}$, and then we compute the quantity in (5.24) for $i \in [100]$ and $Q \in [5]$ and compare it to the relative error for the deterministic approximate algorithm. Figures A.1–A.5 show the results. In the case of Gaussian matrices (Figure A.1), the results look very similar to those in the main manuscript, with almost no difference in error between the deterministic and the randomized approximate algorithms. Figures A.2–A.5 (uniform and type 1–3 adversarial) show that randomization can both increase and decrease the error. Note that both the deterministic and randomized approximate algorithms do particularly poorly on type 1 adversarial matrices.

Next, we repeat the first experiment in Section 5.3.2 with multiple Gaussian, uniform and type 1–3 adversarial matrices. We use the same setup as in the main manuscript: For each experiment, we draw random matrices $\boldsymbol{A}, \boldsymbol{B} \in \mathbb{R}^{80\times 80}$ and compute the quantity in (5.23), but with each $\hat{F}_i^{(Q)}$ replaced by $\hat{G}_i^{(Q)}$, for $n \in [10^4]$ and $Q \in [3]$. Figures A.6–A.10 show the results. Although using more recursions seems to increase the error for a single realization of the algorithm, it also seems to reduce the error of the expectation of the computed matrix product.

Finally, we repeat the second experiment in Section 5.3.2 for the Strassen EBC with multiple

Gaussian, uniform and type 1–3 adversarial matrices. We use the same setup as in the main manuscript: For each experiment, we draw random matrices $\boldsymbol{A}, \boldsymbol{B} \in \mathbb{R}^{320\times 320}$ and compute the quantity in (5.24), but with each $\hat{F}_i^{(Q)}$ replaced by $\hat{G}_i^{(Q)}$, for $i \in [100]$ and $Q \in [5]$. We do the same for variants of the algorithm that only use random permutations or random sign functions. We compare these to the deterministic version $G^{(Q)}$, and also include the error of the standard algorithm computed in single precision as a reference. Figures A.11–A.15 show the results, which look very similar to those presented in the main manuscript.

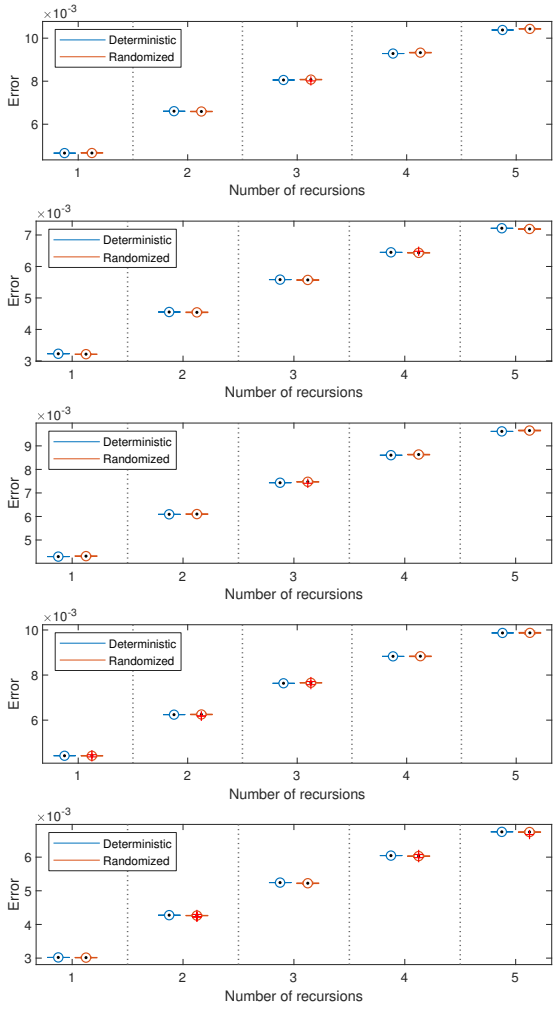

Figure A.1: (Five subplots above) Error for determinstic ABC compared to the error for the randomized counterpart, over 100 realizations of the randomized algorithm. $\boldsymbol{A}, \boldsymbol{B} \in \mathbb{R}^{320\times 320}$ are **Gaussian**, and each subplot corresponds to one realization of the pair $(\boldsymbol{A}, \boldsymbol{B})$.

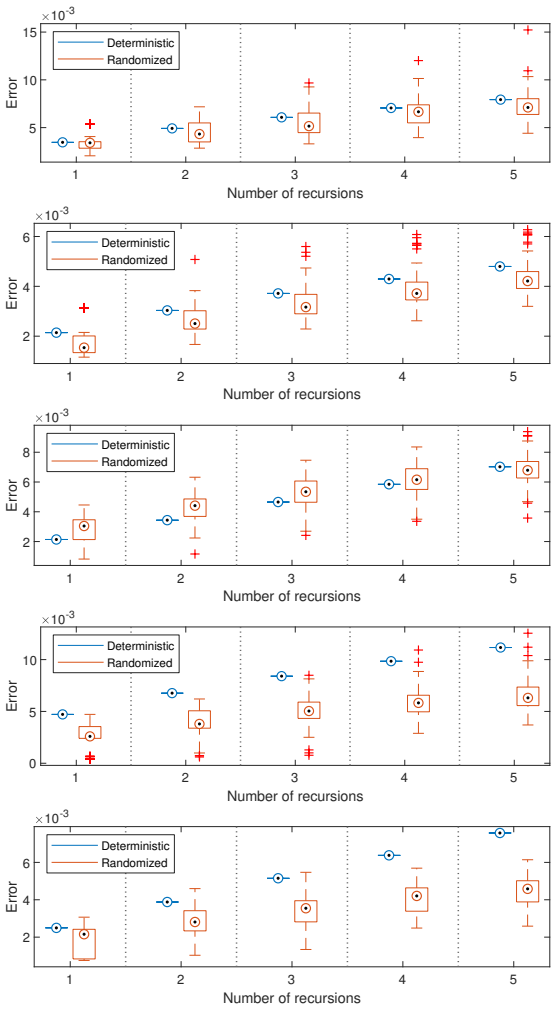

Figure A.2: (Five subplots above) Error for determinstic ABC compared to the error for the randomized counterpart, over 100 realizations of the randomized algorithm. $\boldsymbol{A}, \boldsymbol{B} \in \mathbb{R}^{320\times 320}$ are **uniform**, and each subplot corresponds to one realization of the pair $(\boldsymbol{A}, \boldsymbol{B})$.

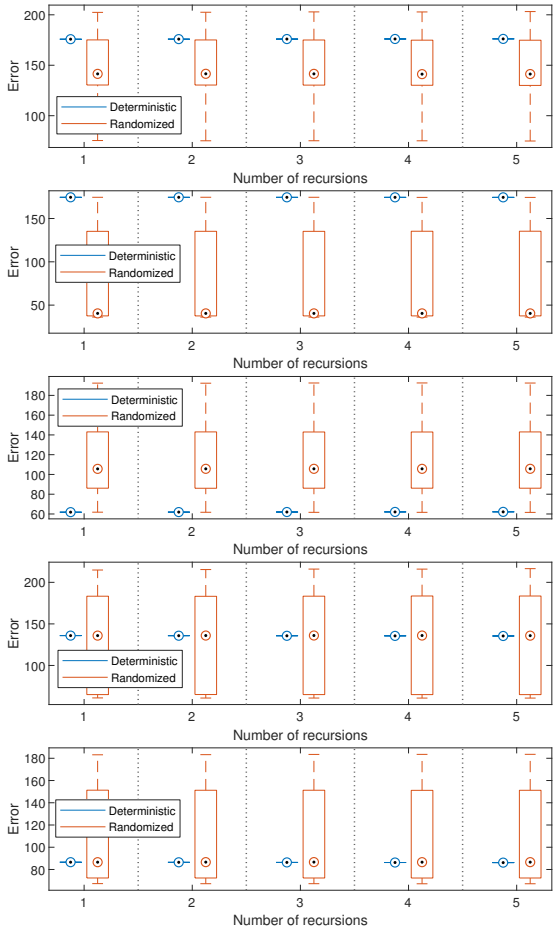

Figure A.3: (Five subplots above) Error for determinstic ABC compared to the error for the randomized counterpart, over 100 realizations of the randomized algorithm. $\boldsymbol{A}, \boldsymbol{B} \in \mathbb{R}^{320\times 320}$ are **type 1 adversarial**, and each subplot corresponds to one realization of the pair $(\boldsymbol{A}, \boldsymbol{B})$.

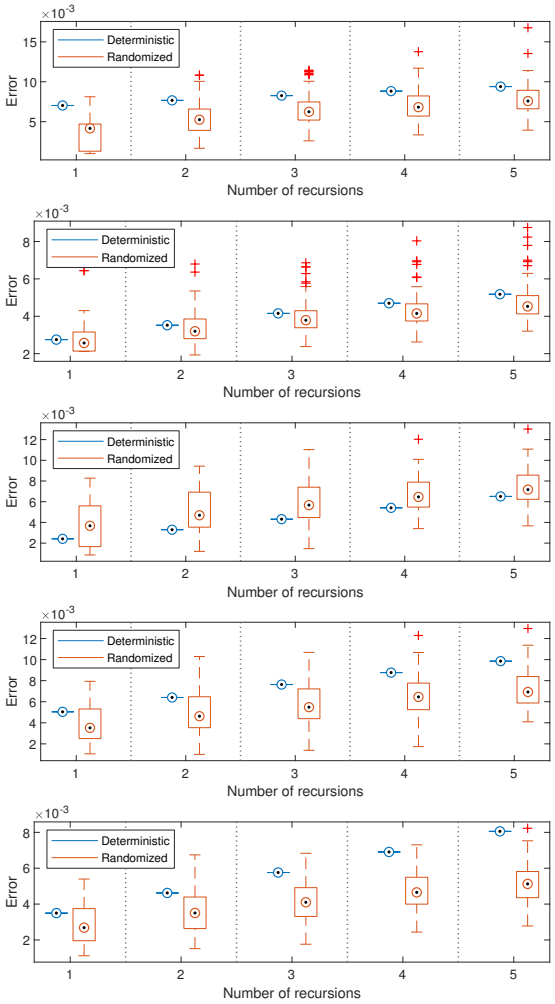

Figure A.4: (Five subplots above) Error for determinstic ABC compared to the error for the randomized counterpart, over 100 realizations of the randomized algorithm. $\boldsymbol{A}, \boldsymbol{B} \in \mathbb{R}^{320\times 320}$ are **type 2 adversarial**, and each subplot corresponds to one realization of the pair $(\boldsymbol{A}, \boldsymbol{B})$.

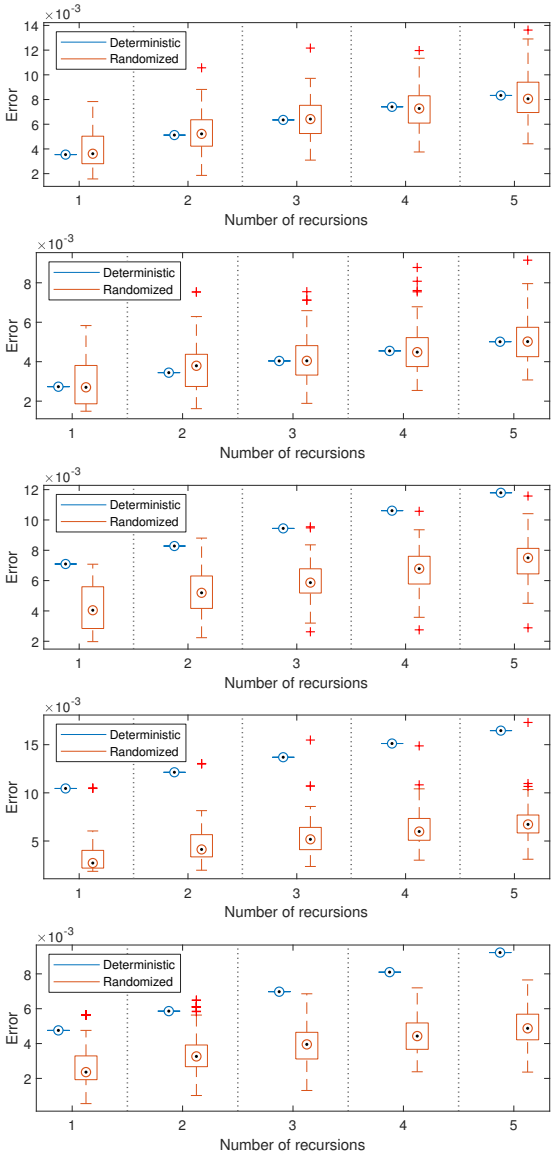

Figure A.5: (Five subplots above) Error for determinstic ABC compared to the error for the randomized counterpart, over 100 realizations of the randomized algorithm. $\boldsymbol{A}, \boldsymbol{B} \in \mathbb{R}^{320\times 320}$ are **type 3 adversarial**, and each subplot corresponds to one realization of the pair $(\boldsymbol{A}, \boldsymbol{B})$.

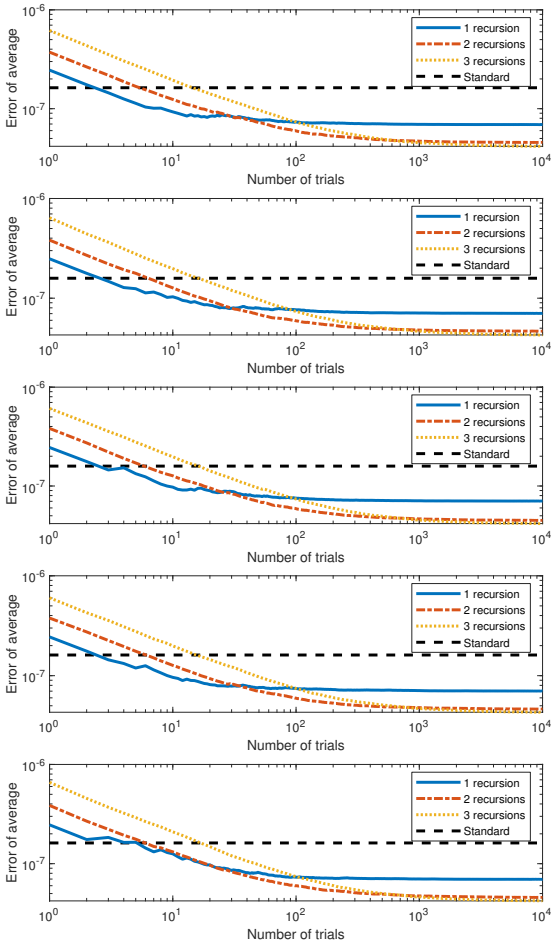

Figure A.6: (Five subplots above) Error for average of randomized EBC compared to the standard algorithm in single precision floating point arithmetic. $\boldsymbol{A}, \boldsymbol{B} \in \mathbb{R}^{80\times 80}$ are **Gaussian**, and each subplot corresponds to one realization of the pair $(\boldsymbol{A}, \boldsymbol{B})$.

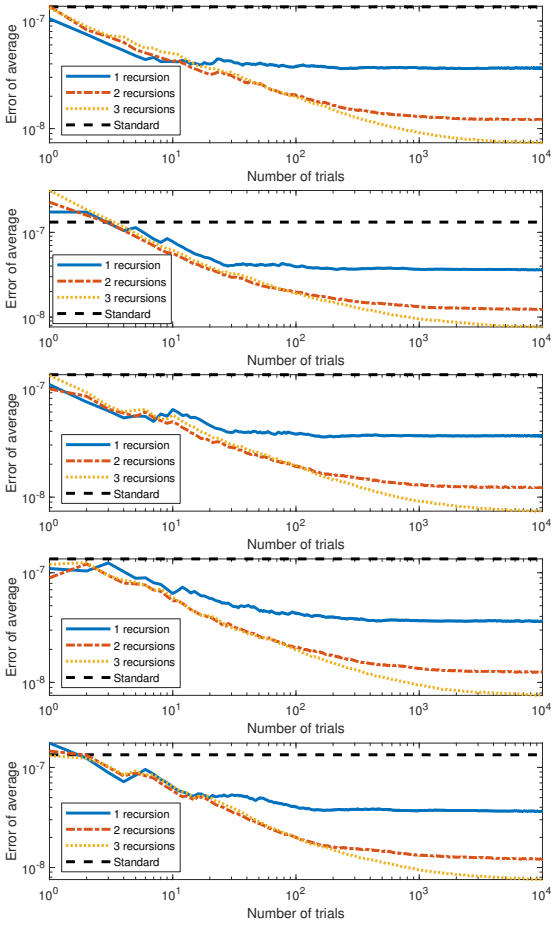

Figure A.7: (Five subplots above) Error for average of randomized EBC compared to the standard algorithm in single precision floating point arithmetic. $\boldsymbol{A}, \boldsymbol{B} \in \mathbb{R}^{80\times 80}$ are **uniform**, and each subplot corresponds to one realization of the pair $(\boldsymbol{A}, \boldsymbol{B})$.

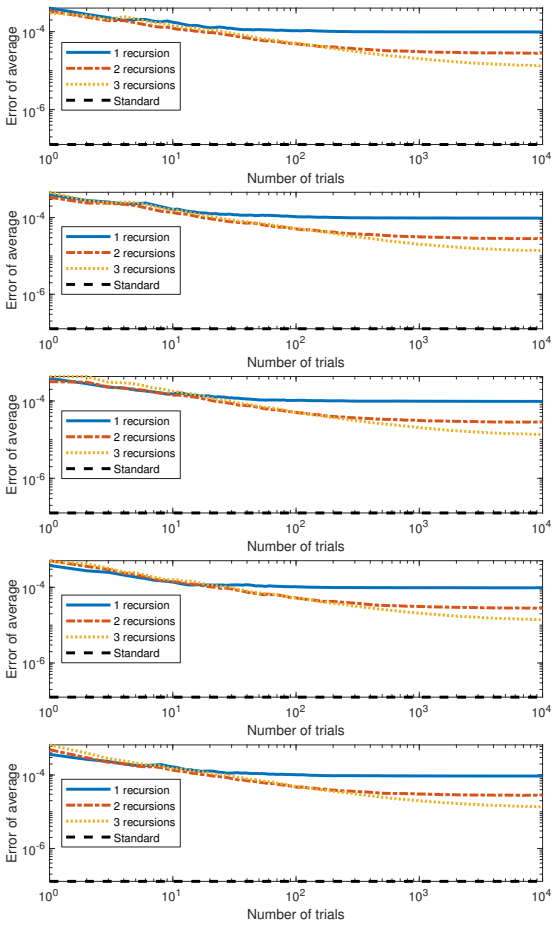

Figure A.8: (Five subplots above) Error for average of randomized EBC compared to the standard algorithm in single precision floating point arithmetic. $\boldsymbol{A}, \boldsymbol{B} \in \mathbb{R}^{80\times 80}$ are **type 1 adversarial**, and each subplot corresponds to one realization of the pair $(\boldsymbol{A}, \boldsymbol{B})$.

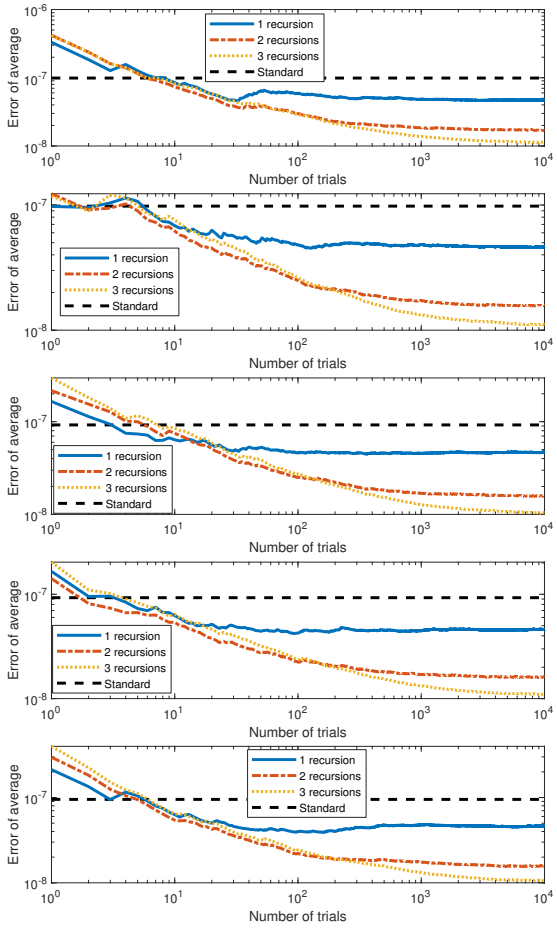

Figure A.9: (Five subplots above) Error for average of randomized EBC compared to the standard algorithm in single precision floating point arithmetic. $\boldsymbol{A}, \boldsymbol{B} \in \mathbb{R}^{80\times 80}$ are **type 2 adversarial**, and each subplot corresponds to one realization of the pair $(\boldsymbol{A}, \boldsymbol{B})$.

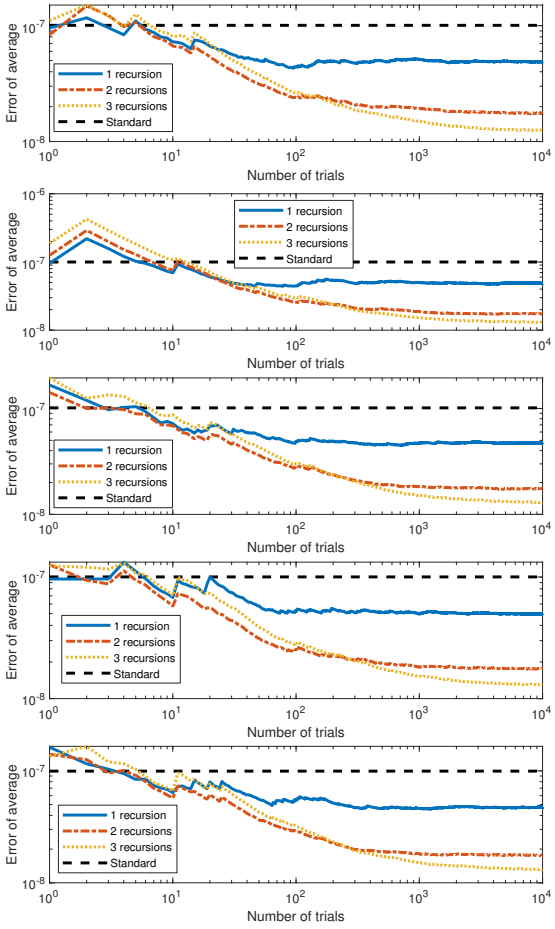

Figure A.10: (Five subplots above) Error for average of randomized EBC compared to the standard algorithm in single precision floating point arithmetic. $\boldsymbol{A}, \boldsymbol{B} \in \mathbb{R}^{80\times 80}$ are **type 3 adversarial**, and each subplot corresponds to one realization of the pair $(\boldsymbol{A}, \boldsymbol{B})$.

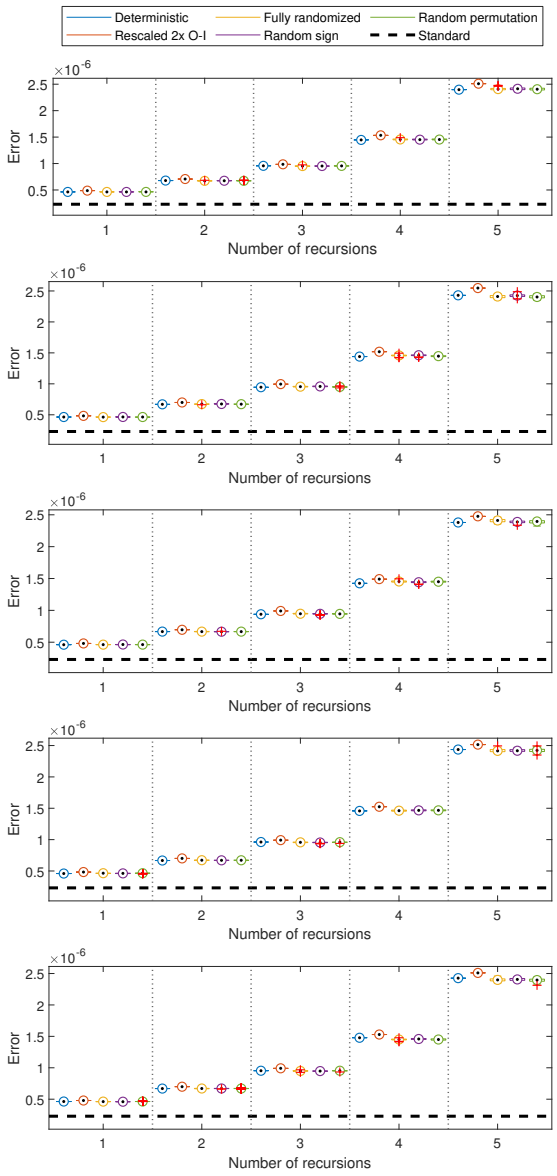

Figure A.11: (Five subplots above) Error for different variants of the EBC in single precision floating point arithmetic, over 100 realizations of the randomized algorithms. $\boldsymbol{A}, \boldsymbol{B} \in \mathbb{R}^{320\times 320}$ are **Gaussian**, and each subplot corresponds to one realization of the pair $(\boldsymbol{A}, \boldsymbol{B})$.

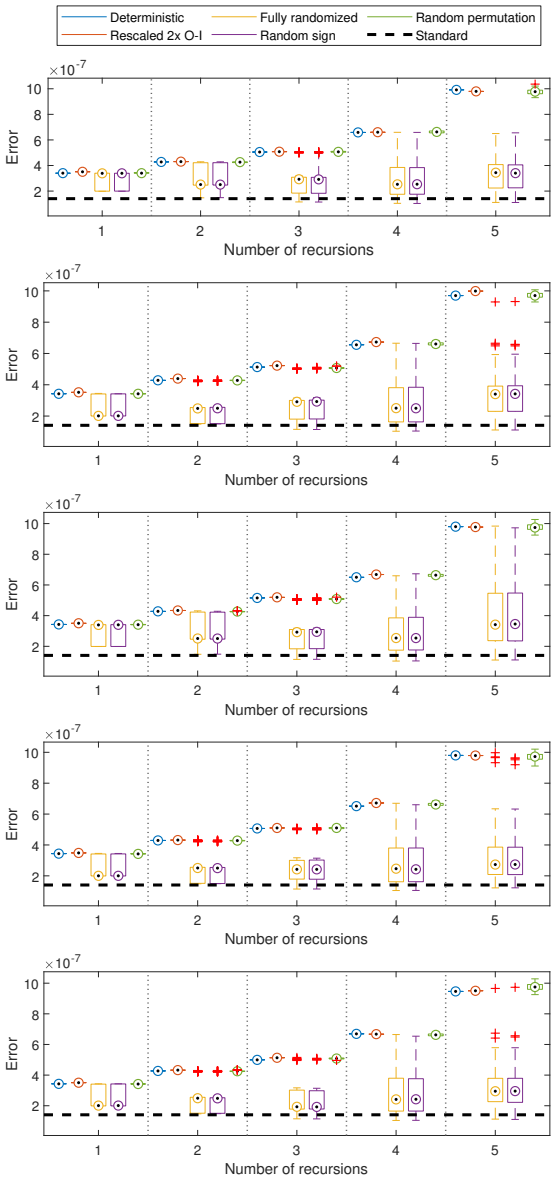

Figure A.12: (Five subplots above) Error for different variants of the EBC in single precision floating point arithmetic, over 100 realizations of the randomized algorithms. $\boldsymbol{A}, \boldsymbol{B} \in \mathbb{R}^{320\times 320}$ are **uniform**, and each subplot corresponds to one realization of the pair $(\boldsymbol{A}, \boldsymbol{B})$.

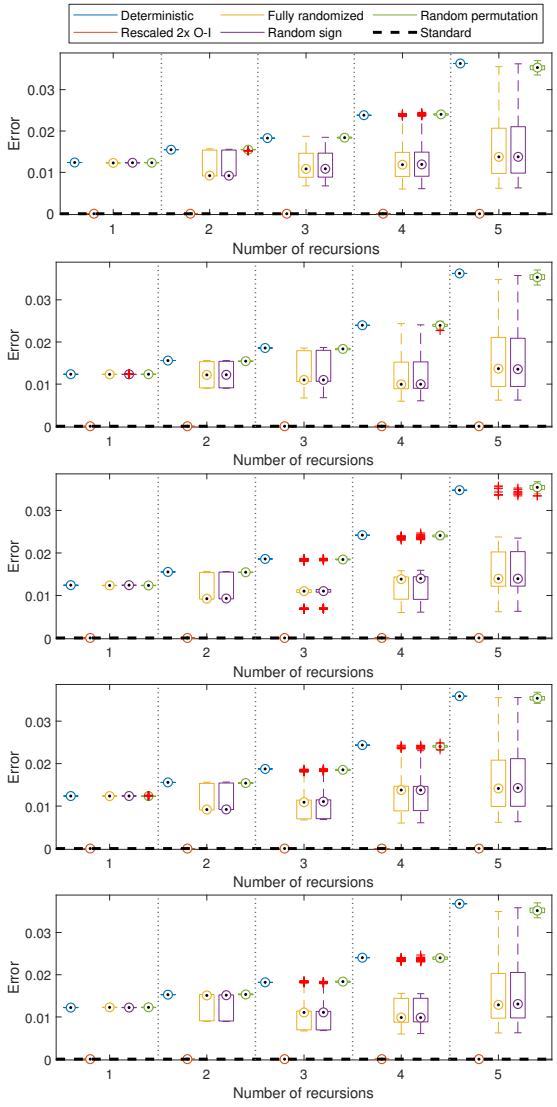

Figure A.13: (Five subplots above) Error for different variants of the EBC in single precision floating point arithmetic, over 100 realizations of the randomized algorithms. $\boldsymbol{A}, \boldsymbol{B} \in \mathbb{R}^{320\times 320}$ are **type 1 adversarial**, and each subplot corresponds to one realization of the pair $(\boldsymbol{A}, \boldsymbol{B})$.

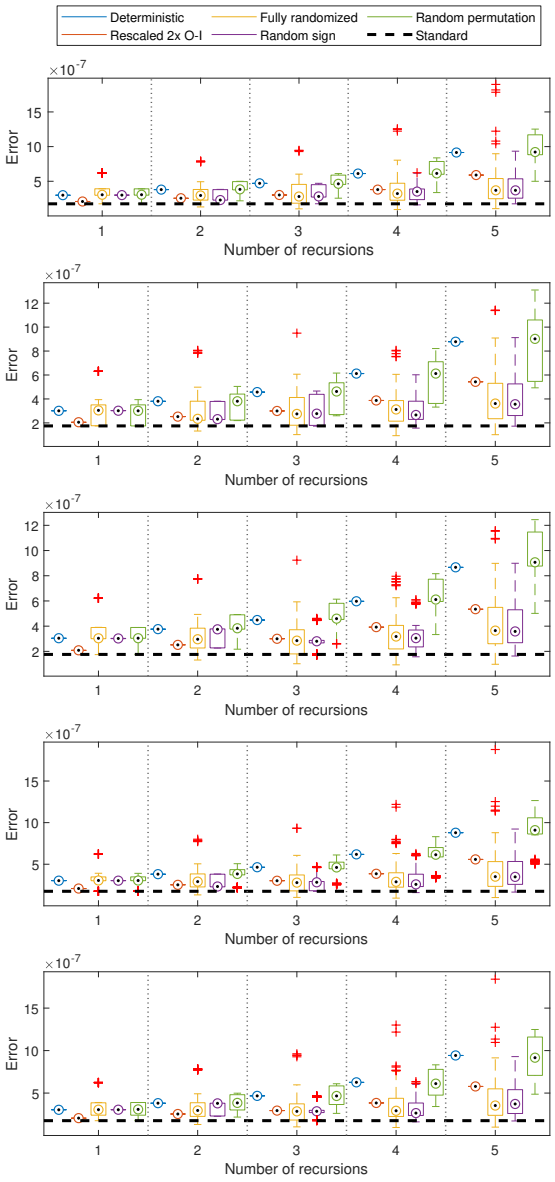

Figure A.14: (Five subplots above) Error for different variants of the EBC in single precision floating point arithmetic, over 100 realizations of the randomized algorithms. $\boldsymbol{A}, \boldsymbol{B} \in \mathbb{R}^{320\times 320}$ are **type 2 adversarial**, and each subplot corresponds to one realization of the pair $(\boldsymbol{A}, \boldsymbol{B})$.

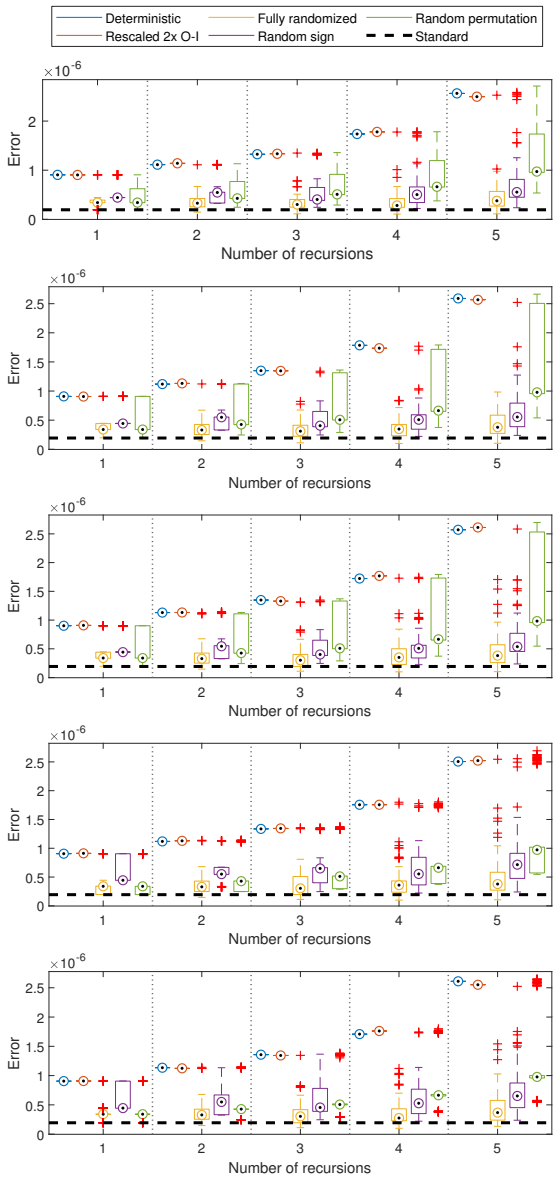

Figure A.15: (Five subplots above) Error for different variants of the EBC in single precision floating point arithmetic, over 100 realizations of the randomized algorithms. $\boldsymbol{A}, \boldsymbol{B} \in \mathbb{R}^{320\times 320}$ are **type 3 adversarial**, and each subplot corresponds to one realization of the pair $(\boldsymbol{A}, \boldsymbol{B})$.

Appendix B

Supplementary material for Chapter 6

B.1 Links to datasets

- The Bitcoin Alpha dataset is available at

  `https://snap.stanford.edu/data/soc-sign-bitcoin-alpha.html`.

- The Bitcoin OTC dataset is available at

  `https://snap.stanford.edu/data/soc-sign-bitcoin-otc.html`.

- The Reddit dataset is available at

  `https://snap.stanford.edu/data/soc-RedditHyperlinks.html`.

  Note that we use the dataset with hyperlinks in the body of the posts.

- The chess dataset was originally downloaded from

  `http://konect.uni-koblenz.de/networks/chess`.

  This link no longer works, and the Koblenz Network Collection appears to no longer be available online. We therefore added the chess dataset to the TM-GCN GitHub repository at `https://github.com/IBM/TM-GCN`.

- SBM (structured block matrix) was generated using the code

  `https://github.com/palash1992/DynamicGEM/`.

  We used 1000 nodes, 8 communities (blocks) and 50 time instances.

B.2 Further details on the experiment setup

When partitioning the data into T graphs, as described in Section 6.5, if there are multiple data points corresponding to an edge (m, n) for a given time step t, we only add that edge once to the corresponding graph and set the label equal to the sum of the labels of the different data points. E.g., if bitcoin user m makes three transactions to n during time step t with ratings 10, 2, -1, then we add a single edge (m, n) to graph t with label $10 + 2 - 1 = 11$.

B.2.1 Edge classification

For training, we run gradient descent with a learning rate of 0.01 and momentum of 0.9 for 10,000 iterations. For each 100 iterations, we compute and store the performance of the model on the validation data. The weight vector α in the loss function (6.6) is treated as a hyperparameter in the bitcoin and Reddit experiments. Since these datasets all have two edge classes, let α_0 and α_1 be the weights of the minority (negative) and majority (positive) classes, respectively. Since these parameters add to 1, we have $\alpha_1 = 1 - \alpha_0$. For all methods, we repeat the bitcoin and Reddit experiments once for each $\alpha_0 \in \{0.75, 0.76, \ldots, 0.95\}$. For each model and dataset, we then find the best stored performance of the model on the validation data across all α_0 values. We then treat the corresponding model as the trained model, and report its performance on the test data. The results for the chess experiment are computed in the same way, but only for a single vector $\alpha = [1/3,\ 1/3,\ 1/3]$.

B.2.2 Link prediction

For training, we run gradient descent with a learning rate of 0.01 and momentum of 0.9 for 1,000 iterations. For each 100 iterations, we compute and store the performance of the model on the validation data. We used $\alpha_0 = 0.90$ in our experiments, where α_0 is the weight of the class corresponding to existing edges.

B.2.3 Definition of performance measures

Suppose we are classifying N objects into C classes. Let $\boldsymbol{u} \in \{1, 2, \ldots, C\}^N$ be a vector containing our computed classification, and let $\boldsymbol{v} \in \{1, 2, \ldots, C\}^N$ be a vector which contains the true classes. Furthermore, let k denote the class we are interested in identifying (i.e., negative edges in bitcoin and Reddit edge classification problems). Then, the F1 score is defined as follows:

$$\text{F1 score} = 2 \cdot \frac{\text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}},$$

where

$$\begin{aligned}
\text{precision} &\stackrel{\text{def}}{=} \frac{\text{true positive}}{\text{true positive} + \text{false positive}}, \\
\text{recall} &\stackrel{\text{def}}{=} \frac{\text{true positive}}{\text{true positive} + \text{false negative}}, \\
\text{true positive} &\stackrel{\text{def}}{=} |\{n \in \{1, 2, \ldots, N\} : \boldsymbol{u}_n = \boldsymbol{v}_n = k\}|, \\
\text{false positive} &\stackrel{\text{def}}{=} |\{n \in \{1, 2, \ldots, N\} : \boldsymbol{u}_n = k, \boldsymbol{v}_n \neq k\}|, \\
\text{false negative} &\stackrel{\text{def}}{=} |\{n \in \{1, 2, \ldots, N\} : \boldsymbol{u}_n \neq k, \boldsymbol{v}_n = k\}|.
\end{aligned}$$

For the accuracy in the edge classification experiment on the chess dataset, we simply compute it as the proportion of correctly labeled edges.

For computing mean average precision, we use the `average_precision_score` function in the Scikit-learn library. As input, we use a vector containing the probabilities for the class of interest (i.e., the class corresponding to existing edges in link prediction) generated by the output model (6.2).

B.2.4 Choice of M matrix

We consider two variants of the lower banded triangular $\boldsymbol{M}$ matrices in our experiments. Specifically, in the first matrix $M1$, the entries of $\boldsymbol{M}$ are set to

$$\boldsymbol{M}_{tk} \stackrel{\text{def}}{=} \begin{cases} \frac{1}{\min(b,t)} & \text{if } \max(1, t-b+1) \leq k \leq t, \\ 0 & \text{otherwise,} \end{cases} \tag{B.1}$$

which implies that $\sum_k \boldsymbol{M}_{tk} = 1$ for each t. However, this transform gives equal weights to all the previous b time instances. In the second matrix (M2), we have the entries as:

$$\boldsymbol{M}_{tk} \stackrel{\text{def}}{=} \begin{cases} \frac{1}{k} & \text{if } \max(1, t-b+1) \leq k \leq t, \\ 0 & \text{otherwise.} \end{cases} \tag{B.2}$$

This transform gives exponential weights to the b previous time instances, weighing recent ones more, and the weights decay exponentially for older time instances.

B.3 Additional experimental results

Since the graphs in the considered datasets are directed, we also investigate the impact of symmetrizing the adjacency matrices, where the symmetrized version of an adjacency matrix $\boldsymbol{A}$ is defined as $\boldsymbol{A}_{\text{sym}} \stackrel{\text{def}}{=} 1/2(\boldsymbol{A} + \boldsymbol{A}^\top)$. Table B.1 shows the results. Our method outperforms the other methods on the Bitcoin OTC dataset and the chess dataset, and performs similarly but slightly worse than the best performing methods on the Bitcoin Alpha and Reddit datasets. Overall, it seems like symmetrizing the adjacency matrices leads to lower performance.

Table B.1: Results for edge classification when adjacency matrices have been symmetrized. Performance measures are F1 score† or accuracy*. A higher value is better.

	Dataset			
Method	Bitcoin OTC†	Bitcoin Alpha†	Reddit†	Chess*
WD-GCN	0.1009	0.1319	**0.2173**	0.4321
EvolveGCN	0.0913	**0.2273**	0.1942	0.4091
GCN	0.0769	0.1538	0.1966	0.4369
TM-GCN - M1	**0.3103**	0.2207	0.2071	**0.4713**

B.4 Additional details and proofs

Here, we give additional details regarding the tensor M-product framework. We also present a few additional theoretical results and proofs.

B.4.1 Additional details

A benefit of the tensor M-product framework is that many standard matrix concepts can be generalized in a straightforward manner. Definitions 69–72 extend the matrix concepts of diagonality, identity, transpose and orthogonality to tensors [34, 113].

Definition 69 (f-diagonal)**.** A tensor $\boldsymbol{\mathcal{X}} \in \mathbb{R}^{N\times N\times T}$ is said to be **f-diagonal** if each frontal slice $\boldsymbol{\mathcal{X}}_{::t}$ is diagonal.

Definition 70 (Identity tensor)**.** Let $\hat{\boldsymbol{\mathcal{J}}} \in \mathbb{R}^{N\times N\times T}$ be defined facewise as $\hat{\boldsymbol{\mathcal{J}}}_{::t} = \boldsymbol{I}$, where $\boldsymbol{I}$ is the matrix identity. The M-product **identity tensor** $\boldsymbol{\mathcal{J}} \in \mathbb{R}^{N\times N\times T}$ is then defined as $\boldsymbol{\mathcal{J}} \overset{\text{def}}{=} \hat{\boldsymbol{\mathcal{J}}} \times_3 \boldsymbol{M}^{-1}$.

Definition 71 (Tensor transpose)**.** The transpose of a tensor $\boldsymbol{\mathcal{X}}$ is defined as $\boldsymbol{\mathcal{X}}^\top \overset{\text{def}}{=} \boldsymbol{\mathcal{Y}} \times_3 \boldsymbol{M}^{-1}$, where $\boldsymbol{\mathcal{Y}}_{::t} = (\boldsymbol{\mathcal{X}} \times_3 \boldsymbol{M})_{::t}^\top$ for each $t \in \{1, \ldots, T\}$.

Definition 72 (Orthogonal tensor)**.** A tensor $\boldsymbol{\mathcal{X}} \in \mathbb{R}^{N\times N\times T}$ is said to be **orthogonal** if $\boldsymbol{\mathcal{X}} \star \boldsymbol{\mathcal{X}}^\top = \boldsymbol{\mathcal{X}}^\top \star \boldsymbol{\mathcal{X}} = \boldsymbol{\mathcal{J}}$.

Leveraging these concepts, a tensor eigendecomposition can now be defined [34, 113]:

Definition 73 (Tensor eigendecomposition)**.** Let $\boldsymbol{\mathcal{X}} \in \mathbb{R}^{N\times N\times T}$ be a tensor and assume that each frontal slice $(\boldsymbol{\mathcal{X}} \times_3 \boldsymbol{M})_{::t}$ is symmetric. We can then eigendecompose these as $(\boldsymbol{\mathcal{X}} \times_3 \boldsymbol{M})_{::t} = \hat{\boldsymbol{\mathcal{Q}}}_{::t}\hat{\boldsymbol{\mathcal{D}}}_{::t}\hat{\boldsymbol{\mathcal{Q}}}_{::t}^\top$, where $\hat{\boldsymbol{\mathcal{Q}}}_{::t} \in \mathbb{R}^{N\times N}$ is orthogonal and $\hat{\boldsymbol{\mathcal{D}}}_{::t} \in \mathbb{R}^{N\times N}$ is diagonal. We assume that the eigenvalues along the diagonal of each $\hat{\boldsymbol{\mathcal{D}}}_{::t}$ are ordered in descending order, i.e., $\hat{\boldsymbol{\mathcal{D}}}_{nnt} \geq \hat{\boldsymbol{\mathcal{D}}}_{mmt}$ whenever $n < m$. The **tensor eigendecomposition** of $\boldsymbol{\mathcal{X}}$ is defined as $\boldsymbol{\mathcal{X}} \overset{\text{def}}{=} \boldsymbol{\mathcal{Q}} \star \boldsymbol{\mathcal{D}} \star \boldsymbol{\mathcal{Q}}^\top$, where $\boldsymbol{\mathcal{Q}} \overset{\text{def}}{=} \hat{\boldsymbol{\mathcal{Q}}} \times_3 \boldsymbol{M}^{-1}$ is orthogonal, and $\boldsymbol{\mathcal{D}} \overset{\text{def}}{=} \hat{\boldsymbol{\mathcal{D}}} \times_3 \boldsymbol{M}^{-1}$ if f-diagonal.

Illustration Figure B.1 (left) illustrates the unfolding operation. Figure B.1 (right) shows how, once unfolded, the matrix unfold($\boldsymbol{\mathcal{X}}$) is multiplied from the left by $\boldsymbol{M}$, which has a lower triangular banded structure. The output is then folded back up into a tensor by doing the inverse operation of that illustrated in Figure B.1.

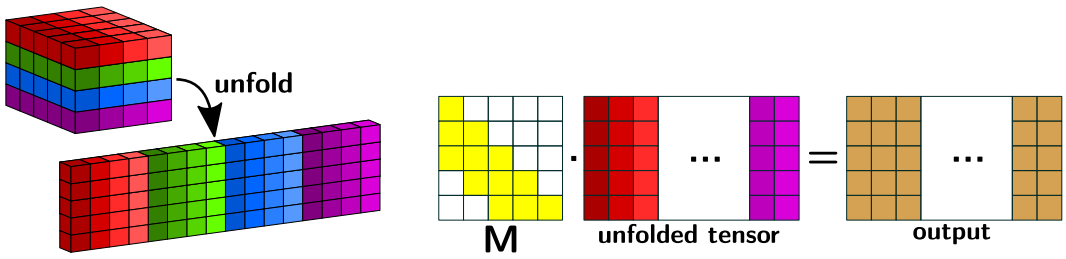

Figure B.1: Illustration of (left) unfold operation applied to $4 \times 4 \times 5$ tensor, and (right) matrix product between $\boldsymbol{M}$ and the unfolded tensor.

B.4.2 Additional results

Here, we present a few more results related to our analysis in Section 6.4.1. Much like the spectrum of a normalized graph Laplacian is contained in $[0, 2]$, the tensor spectrum of $\boldsymbol{\mathcal{L}}$ satisfies a similar property when $\boldsymbol{M}$ is chosen appropriately.

Proposition 74 (Spectral bound)**.** *The entries of* $\hat{\boldsymbol{\mathcal{D}}} = \boldsymbol{\mathcal{D}} \times_3 \boldsymbol{M}$ *lie in* $[0, 2]$ *for the first* $\boldsymbol{M}$ *matrix* $M1$*.*

Proof. Each $\boldsymbol{\mathcal{A}}_{::t}$ has a spectrum contained in $[-1, 1]$. Since $\boldsymbol{\mathcal{A}}_{::t}$ is symmetric, it follows that $\|\boldsymbol{\mathcal{A}}_{::t}\|_2 \leq 1$. Consequently,

$$\|(\boldsymbol{\mathcal{A}} \times_3 \boldsymbol{M})_{::t}\|_2 = \Big\| \sum_{j=1}^{T} \boldsymbol{M}_{tj} \boldsymbol{\mathcal{A}}_{::j} \Big\|_2 \leq \sum_{j=1}^{T} |\boldsymbol{M}_{tj}| \|\boldsymbol{\mathcal{A}}_{::j}\|_2 \leq 1,$$

where we used the fact that $\sum_j |\boldsymbol{M}_{tj}| = 1$. So since the frontal slices $(\boldsymbol{\mathcal{A}} \times_3 \boldsymbol{M})_{::t}$ are symmetric, they each have a spectrum in $[-1, 1]$. It follows that each frontal slice

$$(\boldsymbol{\mathcal{L}} \times_3 \boldsymbol{M})_{::t} = \boldsymbol{I} - (\boldsymbol{\mathcal{A}} \times_3 \boldsymbol{M})_{::t}$$

has a spectrum contained in $[0, 2]$, which means that the entries of $\hat{\boldsymbol{\mathcal{D}}}$ all lie in $[0, 2]$. □

Following the work by [113], three-dimensional tensors in $\mathbb{R}^{M \times N \times T}$ can be viewed as operators on $N \times T$ matrices, with those matrices "twisted" into tensors in $\mathbb{R}^{N \times 1 \times T}$. With this in mind, we define a tensor variant of the graph Fourier transform.

Definition 75 (Tensor-tube M-product)**.** Let $\boldsymbol{\mathcal{X}} \in \mathbb{R}^{I \times J \times T}$ and $\boldsymbol{\theta} \in \mathbb{R}^{1 \times 1 \times T}$. Analogously to the definition of the matrix-scalar product, we define $\boldsymbol{\mathcal{X}} \star \boldsymbol{\theta}$ via $(\boldsymbol{\mathcal{X}} \star \boldsymbol{\theta})_{ij:} \stackrel{\text{def}}{=} \boldsymbol{\mathcal{X}}_{ij:} \star \boldsymbol{\theta}$.

Definition 76 (Tensor graph Fourier transform)**.** Let $\boldsymbol{\mathcal{X}} \in \mathbb{R}^{N \times F \times T}$ be a tensor, and let $\boldsymbol{\mathcal{Q}}$ be defined as in Definition 54. We define a **tensor graph Fourier transform** F via $F(\boldsymbol{\mathcal{X}}) \stackrel{\text{def}}{=} \boldsymbol{\mathcal{Q}}^\top \star \boldsymbol{\mathcal{X}} \in \mathbb{R}^{N \times F \times T}$.

This is analogous to the definition of the matrix graph Fourier transform. This defines a convolution like operation for tensors similar to spectral graph convolution [38]. Each lateral slice $\boldsymbol{\mathcal{X}}_{:j:}$ is expressible in terms of the set $\{\boldsymbol{\mathcal{Q}}_{:n:}\}_{n=1}^N$ as follows:

$$\boldsymbol{\mathcal{X}}_{:j:} = \boldsymbol{\mathcal{Q}} \star \boldsymbol{\mathcal{Q}}^\top \star \boldsymbol{\mathcal{X}}_{:j:} = \sum_{n=1}^{N} \boldsymbol{\mathcal{Q}}_{:n:} \star (\boldsymbol{\mathcal{Q}}^\top \star \boldsymbol{\mathcal{X}}_{:j:})_{n1:},$$

where each $(\boldsymbol{\mathcal{Q}}^\top \star \boldsymbol{\mathcal{X}}_{:j:})_{n1:} \in \mathbb{R}^{1 \times 1 \times T}$ can be considered a tubal scalar. In fact, the lateral slices $\boldsymbol{\mathcal{Q}}_{:n:}$ form a basis for the set $\mathbb{R}^{N \times 1 \times T}$ with product $\star$.

In the following, $\|\cdot\|$ will denote the Frobenius norm (i.e., the square root of the sum of the elements squared) of a matrix or tensor, and $\|\cdot\|_2$ will denote the matrix spectral norm. We first provide a few further results that clarify the algebraic properties of the M-product. Let $\mathbb{R}^{1 \times 1 \times T}$ denote the set of $1 \times 1 \times T$ tensors. Similarly, let $\mathbb{R}^{N \times 1 \times T}$ denote the set of $N \times 1 \times T$ tensors. Under the M-product framework, the set $\mathbb{R}^{1 \times 1 \times T}$ plays a role similar to that played by scalars in matrix algebra. With this in mind, the set $\mathbb{R}^{N \times 1 \times T}$ can be seen as the length-N vectors consisting of tubal elements of length T. Propositions 77 and 78 make this more precise.

Proposition 77 (Proposition 4.2 in [110])**.** *The set $\mathbb{R}^{1 \times 1 \times T}$ with product $\star$, which is denoted by $(\star, \mathbb{R}^{1 \times 1 \times T})$, is a commutative ring with identity.*

Proposition 78 (Theorem 4.1 in [110])**.** *The set $\mathbb{R}^{N \times 1 \times T}$ with product $\star$, which is denoted by $(\star, \mathbb{R}^{N \times 1 \times T})$, is a free module over the ring $(\star, \mathbb{R}^{1 \times 1 \times T})$.*

A free module is similar to a vector space. Like a vector space, it has a basis. Proposition 79 shows that the lateral slices of $\boldsymbol{\mathcal{Q}}$ in the tensor eigendecomposition form a basis for $(\star, \mathbb{R}^{N \times 1 \times T})$, similarly to how the eigenvectors in a matrix eigendecomposition form a basis.

Proposition 79. *The lateral slices $\boldsymbol{\mathcal{Q}}_{:n:} \in \mathbb{R}^{N \times 1 \times T}$ of $\boldsymbol{\mathcal{Q}}$ in Definition 73 form a basis for $(\star, \mathbb{R}^{N \times 1 \times T})$.*

Proof. Let $\boldsymbol{\mathcal{X}} \in \mathbb{R}^{N\times 1\times T}$. Note that

$$\boldsymbol{\mathcal{X}} = \boldsymbol{\mathcal{I}} \star \boldsymbol{\mathcal{X}} = \boldsymbol{\mathcal{Q}} \star \boldsymbol{\mathcal{Q}}^\top \star \boldsymbol{\mathcal{X}} = \sum_{n=1}^{N} \boldsymbol{\mathcal{Q}}_{:n:} \star \boldsymbol{\mathcal{V}}_{n1:},$$

where $\boldsymbol{\mathcal{V}} \stackrel{\text{def}}{=} \boldsymbol{\mathcal{Q}}^\top \star \boldsymbol{\mathcal{X}} \in \mathbb{R}^{N\times 1\times T}$. So the lateral slices of $\boldsymbol{\mathcal{Q}}$ are a generating set for $(\star, \mathbb{R}^{N\times 1\times T})$. Now suppose

$$\sum_{n=1}^{N} \boldsymbol{\mathcal{Q}}_{:n:} \star \boldsymbol{\mathcal{S}}_{n1:} = \boldsymbol{0},$$

for some $\boldsymbol{\mathcal{S}} \in \mathbb{R}^{N\times 1\times T}$. Then $\boldsymbol{0} = \boldsymbol{\mathcal{Q}} \star \boldsymbol{\mathcal{S}}$, and consequently

$$\boldsymbol{0} = (\boldsymbol{\mathcal{Q}} \times_3 \boldsymbol{M}) \triangle (\boldsymbol{\mathcal{S}} \times_3 \boldsymbol{M}).$$

Since each frontal face of $\boldsymbol{\mathcal{Q}} \times_3 \boldsymbol{M}$ is an invertible matrix, this implies that each frontal face of $\boldsymbol{\mathcal{S}} \times_3 \boldsymbol{M}$ is zero, and hence $\boldsymbol{\mathcal{S}} = \boldsymbol{0}$. So the lateral slices of $\boldsymbol{\mathcal{Q}}$ are also linearly independent in $(\star, \mathbb{R}^{N\times 1\times T})$. □

B.4.3 Proofs of propositions in the main text

Proof of Proposition 54. Since each adjacency matrix $\boldsymbol{A}^{(t)}$ and each $\boldsymbol{\mathcal{I}}_{::t}$ is symmetric, each frontal slice $\boldsymbol{\mathcal{L}}_{::t}$ is also symmetric. Consequently,

$$(\boldsymbol{\mathcal{L}} \times_3 \boldsymbol{M})_{ij:} = \boldsymbol{\mathcal{L}}_{ij:} \times_3 \boldsymbol{M} = \boldsymbol{\mathcal{L}}_{ji:} \times_3 \boldsymbol{M} = (\boldsymbol{\mathcal{L}} \times_3 \boldsymbol{M})_{ji:},$$

so each frontal slice of $\boldsymbol{\mathcal{L}} \times_3 \boldsymbol{M}$ is symmetric, and therefore $\boldsymbol{\mathcal{L}}$ has an eigendecomposition. □

Lemma 80. *Let $\boldsymbol{\mathcal{X}} \in \mathbb{R}^{M\times N\times T}$ and let $\boldsymbol{M} \in \mathbb{R}^{T\times T}$ be invertible. Then*

$$\|\boldsymbol{\mathcal{X}}\| \leq \|\boldsymbol{M}^{-1}\|_2 \|\boldsymbol{\mathcal{X}} \times_3 \boldsymbol{M}\|.$$

Proof. We have

$$\|\boldsymbol{\mathcal{X}}\| = \|(\boldsymbol{\mathcal{X}} \times_3 \boldsymbol{M}) \times_3 \boldsymbol{M}^{-1}\| = \|\boldsymbol{M}^{-1} \operatorname{unfold}(\boldsymbol{\mathcal{X}} \times_3 \boldsymbol{M})\|$$

$$\leq \|\boldsymbol{M}^{-1}\|_2 \|\operatorname{unfold}(\boldsymbol{\mathcal{X}} \times_3 \boldsymbol{M})\| = \|\boldsymbol{M}^{-1}\|_2 \|\boldsymbol{\mathcal{X}} \times_3 \boldsymbol{M}\|,$$

where the inequality is a well-known relation that holds for all matrices. □

Proof of Proposition 57. We show the proof for the case when $\boldsymbol{M}$ is defined as in (B.1). However, this proof can easily be adapted to when $\boldsymbol{M}$ is defined as in (B.2) by adapting the interval $[0, 2]$ in (B.3) appropriately.

By Weierstrass approximation theorem, there exists an integer K and a set $\{\hat{\boldsymbol{\theta}}^{(k)}\}_{k=1}^{K} \subset \mathbb{R}^{1\times 1\times T}$ such that for all $t \in \{1, 2, \ldots, T\}$,

$$\sup_{x\in[0,2]} \left| f^{(t)}(x) - \sum_{k=0}^{K} x^k \hat{\boldsymbol{\theta}}^{(k)}_{11t} \right| < \frac{\varepsilon}{\|\boldsymbol{M}^{-1}\|_2 \sqrt{NT}}. \tag{B.3}$$

Let $\boldsymbol{\theta}^{(k)} \stackrel{\text{def}}{=} \hat{\boldsymbol{\theta}}^{(k)} \times_3 \boldsymbol{M}^{-1}$. Note that if $m \neq n$, then

$$\Big(\sum_{k=0}^{K} \boldsymbol{\mathcal{D}}^{\star k} \star \boldsymbol{\theta}^{(k)} \Big)_{mn:} = \sum_{k=0}^{K} ((\hat{\boldsymbol{\mathcal{D}}}^{\triangle k})_{mn:} \times_3 \boldsymbol{M}^{-1}) \star \boldsymbol{\theta}^{(k)} = \mathbf{0} = g(\boldsymbol{\mathcal{D}})_{mn:},$$

since $\hat{\boldsymbol{\mathcal{D}}} = \boldsymbol{\mathcal{D}} \times_3 \boldsymbol{M}$ is f-diagonal. So

$$\begin{aligned}
&\Big\| g(\boldsymbol{\mathcal{D}}) - \sum_{k=0}^{K} \boldsymbol{\mathcal{D}}^{\star k} \star \boldsymbol{\theta}^{(k)} \Big\|^2 = \sum_{n=1}^{N} \Big\| g(\boldsymbol{\mathcal{D}})_{nn:} - \sum_{k=0}^{K} (\boldsymbol{\mathcal{D}}^{\star k})_{nn:} \star \boldsymbol{\theta}^{(k)} \Big\|^2 \\
&\leq \|\boldsymbol{M}^{-1}\|_2^2 \sum_{n=1}^{N} \Big\| g(\boldsymbol{\mathcal{D}})_{nn:} \times_3 \boldsymbol{M} - \sum_{k=0}^{K} ((\boldsymbol{\mathcal{D}} \times_3 \boldsymbol{M})^{\triangle k})_{nn:} \triangle \hat{\boldsymbol{\theta}}^{(k)} \Big\|^2 \\
&= \|\boldsymbol{M}^{-1}\|_2^2 \sum_{n=1}^{N} \sum_{t=1}^{T} \Big| f^{(t)}((\boldsymbol{\mathcal{D}} \times_3 \boldsymbol{M})_{nnt}) - \sum_{k=0}^{K} (\boldsymbol{\mathcal{D}} \times_3 \boldsymbol{M})^{k}_{nnt} \hat{\boldsymbol{\theta}}^{(k)}_{11t} \Big|^2 \\
&< \varepsilon^2,
\end{aligned}$$

where the first inequality follows from Lemma 80, and the last inequality follows since $(\boldsymbol{\mathcal{D}} \times_3 \boldsymbol{M})_{nnt} \in [0, 2]$ due to Proposition 74 (that proposition can easily be adapted to the case when $\boldsymbol{M}$ is defined as in (B.2)). Taking square roots completes the proof. □

Note that, the above proof holds even when we do not apply the inverse transform $\boldsymbol{M}^{-1}$, since $\boldsymbol{M}^{-1}$ only shows up as the norm $\|\boldsymbol{M}^{-1}\|_2$ at the end and the whole proof is still consistent without it.

Appendix C

Supplementary material for Chapter 7

C.1 Missing proofs

In this section we give proofs for Lemma 63 and Theorem 64 in the main manuscript. We first state some results that we will use in these proofs.

Lemma 81 is a variant of Lemma 1 by Drineas et al. [72] but for multiple right hand sides. The proof by Drineas et al. [72] remains essentially identical with only minor modifications to account for the multiple right hand sides.

Lemma 81. *Let* $\mathrm{OPT} \stackrel{\mathrm{def}}{=} \min_{\boldsymbol{X}} \|\boldsymbol{A}\boldsymbol{X} - \boldsymbol{Y}\|_{\mathrm{F}}$ *be a least squares problem where* $\boldsymbol{A} \in \mathbb{R}^{I \times R}$ *and* $I > R$*, and let* $\boldsymbol{U} \in \mathbb{R}^{I \times \mathrm{rank}(\boldsymbol{A})}$ *contain the left singular vectors of* $\boldsymbol{A}$*. Moreover, let* $\boldsymbol{U}^{\perp}$ *be an orthogonal matrix whose columns span the space perpendicular to* $\mathrm{range}(\boldsymbol{U})$ *and define* $\boldsymbol{Y}^{\perp} \stackrel{\mathrm{def}}{=} \boldsymbol{U}^{\perp}(\boldsymbol{U}^{\perp})^{\top}\boldsymbol{Y}$*. Let* $\boldsymbol{S} \in \mathbb{R}^{J \times I}$ *be a matrix satisfying*

$$\sigma^2_{\min}(\boldsymbol{S}\boldsymbol{U}) \geq \frac{1}{\sqrt{2}},$$
$$\|\boldsymbol{U}^{\top}\boldsymbol{S}^{\top}\boldsymbol{S}\boldsymbol{Y}^{\perp}\|^2_{\mathrm{F}} \leq \frac{\varepsilon}{2}\mathrm{OPT}^2,$$

for some $\varepsilon \in (0, 1)$*. Then, it follows that*

$$\|\boldsymbol{A}\tilde{\boldsymbol{X}} - \boldsymbol{Y}\|_{\mathrm{F}} \leq (1 + \varepsilon)\mathrm{OPT},$$

where $\tilde{\boldsymbol{X}} \stackrel{\mathrm{def}}{=} \arg\min_{\boldsymbol{X}} \|\boldsymbol{S}\boldsymbol{A}\boldsymbol{X} - \boldsymbol{S}\boldsymbol{Y}\|_{\mathrm{F}}$*.*

Lemma 82 is a slight restatement of Theorem 2.11 in the monograph by Woodruff [212]. The statement in that monograph has a constant 144 instead of 8/3. However, we found that 8/3 is

sufficient under the assumption that $\varepsilon \in (0,1)$. The proof given by Woodruff [212] otherwise remains the same.

Lemma 82. *Let $\boldsymbol{A} \in \mathbb{R}^{I \times R}$, $\varepsilon \in (0,1)$, $\eta \in (0,1)$ and $\beta \in (0,1]$. Suppose*

$$J > \frac{8}{3} \frac{R \ln(2R/\eta)}{\beta \varepsilon^2}$$

and that $\boldsymbol{S} \sim \mathcal{D}(J, \boldsymbol{q})$ is a leverage score sampling matrix for $(\boldsymbol{A}, \beta)$ (see Definition 62 in the main manuscript). Then, with probability at least $1 - \eta$, the following holds:

$$(\forall i \in [\operatorname{rank}(\boldsymbol{A})]) \quad 1 - \varepsilon \leq \sigma_i^2(\boldsymbol{S}\boldsymbol{U}) \leq 1 + \varepsilon,$$

where $\boldsymbol{U} \in \mathbb{R}^{I \times \operatorname{rank}(\boldsymbol{A})}$ contains the left singular vectors of $\boldsymbol{A}$.

Lemma 83 is a part of Lemma 8 by Drineas et al. [70].

Lemma 83. *Let $\boldsymbol{A}$ and $\boldsymbol{B}$ be matrices with I rows, and let $\beta \in (0,1]$. Let $\boldsymbol{q} \in \mathbb{R}^I$ be a probability distribution satisfying*

$$\boldsymbol{q}(i) \geq \beta \frac{\|\boldsymbol{A}(i,:)\|_2^2}{\|\boldsymbol{A}\|_{\mathrm{F}}^2} \quad \text{for all } i \in [I].$$

If $\boldsymbol{S} \sim \mathcal{D}(J, \boldsymbol{q})$, then

$$\mathbb{E}\|\boldsymbol{A}^\top \boldsymbol{B} - \boldsymbol{A}^\top \boldsymbol{S}^\top \boldsymbol{S} \boldsymbol{B}\|_{\mathrm{F}}^2 \leq \frac{1}{\beta J} \|\boldsymbol{A}\|_{\mathrm{F}}^2 \|\boldsymbol{B}\|_{\mathrm{F}}^2.$$

In the following, $\otimes$ denotes the matrix Kronecker product; see e.g. Section 1.3.6 of Golub and Van Loan [83] for details.

Lemma 84. *For $n \in [N]$, the subchain $\mathcal{G}^{\neq n}$ satisfies*

$$\operatorname{range}(\boldsymbol{G}_{[2]}^{\neq n}) \subseteq \operatorname{range}(\boldsymbol{G}_{(2)}^{(n-1)} \otimes \cdots \otimes \boldsymbol{G}_{(2)}^{(1)} \otimes \boldsymbol{G}_{(2)}^{(N)} \otimes \cdots \otimes \boldsymbol{G}_{(2)}^{(n+1)}). \tag{C.1}$$

Proof. We have

$$\boldsymbol{G}_{[2]}^{\neq n}(\overline{i_{n+1} \cdots i_N i_1 \cdots i_{n-1}}, \overline{r_{n-1} r_n}) =$$

$$\sum_{\substack{r_1, \ldots, r_{n-2} \\ r_{n+1}, \ldots, r_N}} \boldsymbol{G}_{(2)}^{(n+1)}(i_{n+1}, \overline{r_n r_{n+1}}) \cdots \boldsymbol{G}_{(2)}^{(N)}(i_N, \overline{r_{N-1} r_N}) \boldsymbol{G}_{(2)}^{(1)}(i_1, \overline{r_N r_1}) \cdots \boldsymbol{G}_{(2)}^{(n-1)}(i_{n-1}, \overline{r_{n-2} r_{n-1}}).$$

From this, it follows that

$$\boldsymbol{G}_{[2]}^{\neq n}(:, \overline{r_{n-1}r_n}) =$$

$$\sum_{\substack{r_1,\ldots,r_{n-2}\\ r_{n+1},\ldots,r_N}} \boldsymbol{G}_{(2)}^{(n-1)}(:, \overline{r_{n-2}r_{n-1}}) \otimes \cdots \otimes \boldsymbol{G}_{(2)}^{(1)}(:, \overline{r_N r_1}) \otimes \boldsymbol{G}_{(2)}^{(N)}(:, \overline{r_{N-1}r_N}) \otimes \cdots \otimes \boldsymbol{G}_{(2)}^{(n+1)}(:, \overline{r_n r_{n+1}}).$$

The right hand side of this equation is a sum of columns in

$$\boldsymbol{G}_{(2)}^{(n-1)} \otimes \cdots \otimes \boldsymbol{G}_{(2)}^{(1)} \otimes \boldsymbol{G}_{(2)}^{(N)} \otimes \cdots \otimes \boldsymbol{G}_{(2)}^{(n+1)}. \tag{C.2}$$

Consequently, every column of $\boldsymbol{G}_{[2]}^{\neq n}$ is in the range of the matrix in (C.2), and the claim in (C.1) follows. □

Lemma 85. *Let $\boldsymbol{A}$ and $\boldsymbol{B}$ be two matrices with I rows such that* $\operatorname{range}(\boldsymbol{A}) \subseteq \operatorname{range}(\boldsymbol{B})$. *Then $\ell_i(\boldsymbol{A}) \leq \ell_i(\boldsymbol{B})$ for all $i \in [I]$.*

Proof. Let $\boldsymbol{Q} \in \mathbb{R}^{I \times \operatorname{rank}(\boldsymbol{B})}$ be an orthogonal matrix containing the left singular vectors of $\boldsymbol{B}$. Then there exists a matrix $\boldsymbol{M}$ such that $\boldsymbol{A} = \boldsymbol{Q}\boldsymbol{M}$. Let $\boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}^\top = \boldsymbol{M}$ be the thin SVD of $\boldsymbol{M}$ (i.e., such that $\boldsymbol{U}$ and $\boldsymbol{V}$ have only $\operatorname{rank}(\boldsymbol{M})$ columns and $\boldsymbol{\Sigma} \in \mathbb{R}^{\operatorname{rank}(\boldsymbol{M}) \times \operatorname{rank}(\boldsymbol{M})}$). Then $\boldsymbol{A} = \boldsymbol{Q}\boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}^\top = \boldsymbol{W}\boldsymbol{\Sigma}\boldsymbol{V}^\top$, where $\boldsymbol{W} \stackrel{\text{def}}{=} \boldsymbol{Q}\boldsymbol{U}$. Since

$$\boldsymbol{W}^\top \boldsymbol{W} = \boldsymbol{U}^\top \boldsymbol{Q}^\top \boldsymbol{Q}\boldsymbol{U} = \boldsymbol{I},$$

$\boldsymbol{W}$ is orthogonal, so $\boldsymbol{W}\boldsymbol{\Sigma}\boldsymbol{V}^\top = \boldsymbol{A}$ is a thin SVD of $\boldsymbol{A}$. It follows that

$$\ell_i(\boldsymbol{A}) = \|\boldsymbol{W}(i,:)\|_2^2 = \|\boldsymbol{Q}(i,:)\boldsymbol{U}\|_2^2 \leq \|\boldsymbol{Q}(i,:)\|_2^2 \|\boldsymbol{U}\|_2^2 = \|\boldsymbol{Q}(i,:)\|_2^2 = \ell_i(\boldsymbol{B}).$$

□

Remark 86. It is reasonable to assume that $\boldsymbol{\mathcal{G}}^{\neq n} \neq \boldsymbol{0}$ for all $n \in [N]$ during the execution of Algorithms 7 and 8. If this was not the case, the least squares problem for the nth iteration of the inner for loop in these algorithms would have a zero system matrix which would make the least squares problem meaningless. The assumption $\boldsymbol{\mathcal{G}}^{\neq n} \neq \boldsymbol{0}$ for all $n \in [N]$ also implies that $\boldsymbol{\mathcal{G}}^{(n)} \neq \boldsymbol{0}$ for all $n \in [N]$. This means that $\operatorname{rank}(\boldsymbol{G}_{(2)}^{(n)}) \geq 1$ and $\operatorname{rank}(\boldsymbol{G}_{[2]}^{\neq n}) \geq 1$ for all $n \in [N]$, so that the various divisions with these quantities in this paper are well-defined.

Lemma 87 is a restatement of Lemma 63 in the main manuscript.

Lemma 87. *Let β_n be defined as*

$$\beta_n \stackrel{\text{def}}{=} \Big(R_{n-1}R_n \prod_{\substack{j=1\\ j\notin\{n-1,n\}}}^{N} R_j^2\Big)^{-1}.$$

For each $n \in [N]$, the vector $\boldsymbol{q}^{\neq n}$ is a probability distribution on $[\prod_{j\neq n} I_j]$, and $\boldsymbol{S} \sim \mathcal{D}(J, \boldsymbol{q}^{\neq n})$ is a leverage score sampling matrix for $(\boldsymbol{G}_{[2]}^{\neq n}, \beta_n)$.

Proof. All $\boldsymbol{q}^{\neq n}(i)$ are clearly nonnegative. Moreover,

$$\sum_{\substack{i_1,\dots,i_{n-1}\\ i_{n+1},\dots,i_N}} \boldsymbol{q}^{\neq n}(\overline{i_{n+1}\cdots i_N i_1 \cdots i_{n-1}}) = \prod_{\substack{j=1\\ j\neq n}}^{N} \sum_{i_j} \boldsymbol{p}^{(j)}(i_j) = \prod_{\substack{j=1\\ j\neq n}}^{N} 1 = 1,$$

since each $\boldsymbol{p}^{(j)}$ is a probability distribution. So $\boldsymbol{q}^{\neq n}$ is clearly also a probability distribution. Next, define the vector $\boldsymbol{p} \in \mathbb{R}^{\prod_{j\neq n} I_n}$ via

$$\boldsymbol{p}(i) \stackrel{\text{def}}{=} \frac{\ell_i(\boldsymbol{G}_{[2]}^{\neq n})}{\operatorname{rank}(\boldsymbol{G}_{[2]}^{\neq n})}.$$

To show that $\boldsymbol{S}$ is a leverage score sampling matrix for $(\boldsymbol{G}_{[2]}^{\neq n}, \beta_n)$, we need to show that $\boldsymbol{q}^{\neq n}(i) \geq \beta_n \boldsymbol{p}(i)$ for all $i = \overline{i_{n+1}\cdots i_N i_1 \cdots i_{n-1}} \in [\prod_{j\neq n} I_n]$. To this end, combine Lemmas 84 and 85 to get

$$\ell_i(\boldsymbol{G}_{[2]}^{\neq n}) \leq \ell_i(\boldsymbol{G}_{(2)}^{(n-1)} \otimes \cdots \otimes \boldsymbol{G}_{(2)}^{(1)} \otimes \boldsymbol{G}_{(2)}^{(N)} \otimes \cdots \otimes \boldsymbol{G}_{(2)}^{(n+1)}). \tag{C.3}$$

For each $n \in [N]$, let $\boldsymbol{U}^{(n)} \in \mathbb{R}^{I_n \times \operatorname{rank}(\boldsymbol{G}_{(2)}^{(n)})}$ contain the left singular vectors of $\boldsymbol{G}_{(2)}^{(n)}$. It is well-known [203] that the matrix

$$\boldsymbol{U}^{(n-1)} \otimes \cdots \otimes \boldsymbol{U}^{(1)} \otimes \boldsymbol{U}^{(N)} \otimes \cdots \otimes \boldsymbol{U}^{(n+1)}$$

contains the left singular vectors corresponding to nonzero singular values of

$$\boldsymbol{G}_{(2)}^{(n-1)} \otimes \cdots \otimes \boldsymbol{G}_{(2)}^{(1)} \otimes \boldsymbol{G}_{(2)}^{(N)} \otimes \cdots \otimes \boldsymbol{G}_{(2)}^{(n+1)}.$$

Consequently,

$$\begin{aligned}
&\ell_i(\boldsymbol{G}_{(2)}^{(n-1)} \otimes \cdots \otimes \boldsymbol{G}_{(2)}^{(1)} \otimes \boldsymbol{G}_{(2)}^{(N)} \otimes \cdots \otimes \boldsymbol{G}_{(2)}^{(n+1)}) \\
&= \big((\boldsymbol{U}^{(n-1)} \otimes \cdots \otimes \boldsymbol{U}^{(1)} \otimes \boldsymbol{U}^{(N)} \otimes \cdots \otimes \boldsymbol{U}^{(n+1)})(\boldsymbol{U}^{(n-1)} \otimes \cdots \otimes \boldsymbol{U}^{(1)} \otimes \boldsymbol{U}^{(N)} \otimes \cdots \otimes \boldsymbol{U}^{(n+1)})^\top\big)_{ii} \\
&= \big((\boldsymbol{U}^{(n-1)}\boldsymbol{U}^{(n-1)\top}) \otimes \cdots \otimes (\boldsymbol{U}^{(1)}\boldsymbol{U}^{(1)\top}) \otimes (\boldsymbol{U}^{(N)}\boldsymbol{U}^{(N)\top}) \otimes \cdots \otimes (\boldsymbol{U}^{(n+1)}\boldsymbol{U}^{(n+1)\top})\big)_{ii} \\
&= (\boldsymbol{U}^{(n-1)}\boldsymbol{U}^{(n-1)\top})_{i_{n-1}i_{n-1}} \cdots (\boldsymbol{U}^{(1)}\boldsymbol{U}^{(1)\top})_{i_1 i_1} (\boldsymbol{U}^{(N)}\boldsymbol{U}^{(N)\top})_{i_N i_N} \cdots (\boldsymbol{U}^{(n+1)}\boldsymbol{U}^{(n+1)\top})_{i_{n+1}i_{n+1}} \\
&= \ell_{i_{n-1}}(\boldsymbol{G}_{(2)}^{(n-1)}) \cdots \ell_{i_1}(\boldsymbol{G}_{(2)}^{(1)}) \ell_{i_N}(\boldsymbol{G}_{(2)}^{(N)}) \cdots \ell_{i_{n+1}}(\boldsymbol{G}_{(2)}^{(n+1)}),
\end{aligned} \tag{C.4}$$

where the first and fourth equalities follow from the definition of leverage score and the fact that

$$\|\boldsymbol{M}(i,:)\|_2^2 = (\boldsymbol{M}\boldsymbol{M}^\top)(i,i)$$

for any matrix $\boldsymbol{M}$, and the second equality follows from well-known properties of the Kronecker product [203]. Combining (C.3) and (C.4), we have

$$\ell_i(\boldsymbol{G}_{[2]}^{\neq n}) \leq \prod_{\substack{j=1 \\ j \neq n}}^{N} \ell_{i_j}(\boldsymbol{G}_{(2)}^{(j)}). \tag{C.5}$$

We therefore have

$$\boldsymbol{q}^{\neq n}(i) = \frac{\prod_{j \neq n} \ell_{i_j}(\boldsymbol{G}_{(2)}^{(j)})}{\prod_{j \neq n} \operatorname{rank}(\boldsymbol{G}_{(2)}^{(j)})} \geq \frac{\ell_i(\boldsymbol{G}_{[2]}^{\neq n})}{R_{n-1}R_n \prod_{j \in [N] \setminus \{n-1,n\}} R_j^2} = \beta_n \ell_i(\boldsymbol{G}_{[2]}^{\neq n}) \geq \beta_n \boldsymbol{p}(i)$$

as desired, where the first inequality follows from (C.5) and the fact that $\operatorname{rank}(\boldsymbol{G}_{(2)}^{(j)}) \leq R_{j-1}R_j$. □

The following is a restatement of Theorem 64 in the main manuscript. The proof is similar to that of Theorem 2 by Drineas et al. [72].

Theorem 88. *Let* $\boldsymbol{S} \sim \mathcal{D}(J, \boldsymbol{q}^{\neq n})$, $\varepsilon \in (0,1)$, $\delta \in (0,1)$ *and* $\tilde{\boldsymbol{\mathcal{Z}}} \overset{\text{def}}{=} \arg\min_{\boldsymbol{\mathcal{Z}}} \|\boldsymbol{S}\boldsymbol{G}_{[2]}^{\neq n}\boldsymbol{Z}_{(2)}^\top - \boldsymbol{S}\boldsymbol{X}_{[n]}^\top\|_{\mathrm{F}}$. *If*

$$J > \Big(\prod_{j=1}^{N} R_j^2\Big) \max\Big(\frac{16}{3(\sqrt{2}-1)^2} \ln\Big(\frac{4R_{n-1}R_n}{\delta}\Big), \frac{4}{\varepsilon\delta}\Big),$$

then the following holds with probability at least $1-\delta$*:*

$$\|\boldsymbol{G}_{[2]}^{\neq n}\tilde{\boldsymbol{Z}}_{(2)}^\top - \boldsymbol{X}_{[n]}^\top\|_{\mathrm{F}} \leq (1+\varepsilon) \min_{\boldsymbol{\mathcal{Z}}} \|\boldsymbol{G}_{[2]}^{\neq n}\boldsymbol{Z}_{(2)}^\top - \boldsymbol{X}_{[n]}^\top\|_{\mathrm{F}}. \tag{C.6}$$

Proof. Let $\boldsymbol{U} \in \mathbb{R}^{\prod_{j\neq n} I_j \times \operatorname{rank}(\boldsymbol{G}_{[2]}^{\neq n})}$ contain the left singular vectors of $\boldsymbol{G}_{[2]}^{\neq n}$. According to Lemma 87, $\boldsymbol{S}$ is a leverage score sampling matrix for $(\boldsymbol{G}_{[2]}^{\neq n}, \beta_n)$. Since $\boldsymbol{U}$ has at most $R_{n-1}R_n$ columns and

$$J > \frac{16}{3(\sqrt{2}-1)^2}\Big(\prod_{j=1}^{N} R_j^2\Big) \ln\Big(\frac{4R_{n-1}R_n}{\delta}\Big),$$

choosing $\varepsilon = 1 - 1/\sqrt{2}$ and $\eta = \delta/2$ in Lemma 82 therefore gives that

$$\sigma_{\min}^2(\boldsymbol{S}\boldsymbol{U}) \geq 1/\sqrt{2} \tag{C.7}$$

with probability at least $1 - \delta/2$. Similarly to Lemma 81, define $(\boldsymbol{X}_{[n]}^\top)^\perp \stackrel{\text{def}}{=} \boldsymbol{U}^\perp(\boldsymbol{U}^\perp)^\top \boldsymbol{X}_{[n]}^\top$ and $\mathrm{OPT} \stackrel{\text{def}}{=} \min_{\boldsymbol{Z}} \|\boldsymbol{G}_{[2]}^{\neq n}\boldsymbol{Z}_{(2)}^\top - \boldsymbol{X}_{[n]}^\top\|_{\mathrm{F}} = \|(\boldsymbol{X}_{[n]}^\top)^\perp\|_{\mathrm{F}}$. Since

$$\boldsymbol{q}^{\neq n}(i) \geq \beta_n \frac{\ell_i(\boldsymbol{G}_{[2]}^{\neq n})}{\operatorname{rank}(\boldsymbol{G}_{[2]}^{\neq n})} = \beta_n \frac{\|\boldsymbol{U}(i,:)\|_2^2}{\|\boldsymbol{U}\|_{\mathrm{F}}^2} \quad \text{for all } i \in [I],$$

and $\boldsymbol{U}^\top(\boldsymbol{X}_{[n]}^\top)^\perp = \boldsymbol{0}$, Lemma 83 gives that

$$\mathbb{E}\|\boldsymbol{U}^\top\boldsymbol{S}^\top\boldsymbol{S}(\boldsymbol{X}_{[n]}^\top)^\perp\|_{\mathrm{F}}^2 \leq \frac{1}{\beta_n J}\|\boldsymbol{U}\|_{\mathrm{F}}^2\|(\boldsymbol{X}_{[n]}^\top)^\perp\|_{\mathrm{F}}^2 \leq \frac{R_{n-1}R_n}{\beta_n J}\mathrm{OPT}^2.$$

By Markov's inequality,

$$\mathbb{P}(\|\boldsymbol{U}^\top\boldsymbol{S}^\top\boldsymbol{S}(\boldsymbol{X}_{[n]}^\top)^\perp\|_{\mathrm{F}}^2 > \varepsilon\mathrm{OPT}^2/2) \leq \frac{\mathbb{E}\|\boldsymbol{U}^\top\boldsymbol{S}^\top\boldsymbol{S}(\boldsymbol{X}_{[n]}^\top)^\perp\|_{\mathrm{F}}^2}{\varepsilon\mathrm{OPT}^2/2} \leq \frac{2}{\varepsilon J}\Big(\prod_{j\in[N]} R_j^2\Big) < \frac{\delta}{2}$$

since

$$J > \frac{4}{\varepsilon\delta}\Big(\prod_{j=1}^{N} R_j^2\Big).$$

Consequently,

$$\|\boldsymbol{U}^\top\boldsymbol{S}^\top\boldsymbol{S}(\boldsymbol{X}_{[n]}^\top)^\perp\|_{\mathrm{F}}^2 \leq \frac{\varepsilon}{2}\mathrm{OPT}^2 \tag{C.8}$$

with probability at least $1 - \delta/2$. By a union bound, it follows that both (C.7) and (C.8) are true with probability at least $1 - \delta$. From Lemma 81 it therefore follows that (C.6) is true with probability at least $1 - \delta$. □

C.2 Detailed complexity analysis

We provide a detailed complexity analysis in this section to show how we arrived at the numbers in Table 7.1 in the main manuscript.

C.2.1 TR-ALS

In our calculations below, we refer to steps in Algorithm 7 in the main paper, consequently ignoring any cost associated with e.g. normalization and checking termination conditions.

Upfront costs of TR-ALS:

- **Line 1: Initializing cores.** This depends on how the initialization of the cores is done. We assume they are randomly drawn, e.g. from a Gaussian distribution, resulting in a cost $O(NIR^2)$.

Costs per outer loop iteration of TR-ALS:

- **Line 4: Compute unfolded subchain tensor.** If the $N-1$ cores are dense and contracted in sequence, the cost is

$$R^3(I^2 + I^3 + \cdots + I^{N-1}) \leq R^3(NI^{N-2} + I^{N-1}) \leq 2R^3 I^{N-1} = O(I^{N-1}R^3),$$

  where we use the assumption $N < I$ in the second inequality. Doing this for each of the N cores in the inner loop brings the cost to $O(NI^{N-1}R^3)$.

- **Line 5: Solve least squares problem.** We consider the cost when using the standard QR-based approach described in Section 5.3.3 in the book by Golub and Van Loan [83]. The matrix $\boldsymbol{G}_{[2]}^{\neq n}$ is of size $I^{N-1} \times R^2$. Doing a QR decomposition of this matrix costs $O(I^{N-1}R^4)$. Updating the right hand sides and doing back substitution costs $O(I(I^{N-1}R^2 + R^4)) = O(I^N R^2)$, where we used the assumption that $R^2 < I$. The leading order cost for solving the least squares problem is therefore $O(I^N R^2)$. Doing this for each of the N cores in the inner loop brings the cost to $O(NI^N R^2)$.

It follows that the overall leading order cost of TR-ALS is $NIR^2 + \#\text{iter} \cdot NI^N R^2$.

C.2.2 rTR-ALS

In our calculations below, we refer to steps in Algorithm 1 by Yuan et al. [220] with the TR decomposition step in their algorithm done using TR-ALS.

Cost of initial Tucker compression:

- **Line 4: Draw Gaussian matrix.** Drawing these N matrices costs $O(NI^{N-1}K)$.
- **Line 5: Compute random projection.** Computing N projections costs $O(NI^NK)$.
- **Line 6: QR decomposition.** Computing the QR decomposition of an $I \times K$ matrix N times costs $O(NIK^2)$.
- **Line 7: Compute Tucker core.** The cost of compressing each dimension of the input tensor in sequence is

$$I^NK + I^{N-1}K^2 + I^{N-2}K^3 + \cdots + IK^N = O(NI^NK),$$

  where we assume that $K < I$.

Cost of TR decompositions:

- **Line 9: Compute TR decomposition of Tucker core.** Using TR-ALS, this costs $O(NKR^2 + \#\text{iter} \cdot NK^NR^2)$ as discussed in Section C.2.1.
- **Line 11: Compute large TR cores.** This costs in total $O(NIKR^2)$ for all cores.

Combining these costs and recalling the assumption $R^2 < I$, we get an overall leading order cost for rTR-ALS of $NI^NK + \#\text{iter} \cdot NK^NR^2$.

C.2.3 TR-SVD

In our calculations below, we refer to steps in Algorithm 1 by Mickelin and Karaman [153]. We consider a modified version of this algorithm which takes a target rank $(R_1, \ldots, R_N)$ as input instead of an accuracy upper bound ε. For simplicity, we ignore all permutation and reshaping costs, and focus only on the leading order costs which are made up by the SVD calculations.

- **Line 3: Initial SVD.** This is an economy sized SVD of an $I \times I^{N-1}$ matrix, which costs $O(I^{N+1})$; see the table in Figure 8.6.1 of Golub and Van Loan [83] for details.

- **Line 10: SVD in for loop.** The size of the matrix being decomposed will be $IR \times I^{N-k}R$. The cost of computing an economy sized SVD of each for $k = 2, \ldots, N-1$ is

$$R^3(I^N + I^{N-1} + \cdots + I^3) \leq 2R^3 I^N,$$

  where we use the assumption that $N < I$.

Adding these costs up, we get a total cost of $O(I^{N+1} + I^N R^3)$.

C.2.4 TR-SVD-Rand

In our calculations below, we refer to steps in Algorithm 7 by Ahmadi-Asl et al. [2]. For simplicity, we ignore all permutation and reshaping costs, and focus only on the leading order costs. In particular, note that the QR decompositions that come up are of relatively small matrices and therefore relatively cheap. Moreover, we assume that the oversampling parameter P is small enough to ignore.

- **Line 2: Compute projection.** The matrix $\boldsymbol{C}$ is of size $I \times I^{N-1}$ and the matrix $\boldsymbol{\Omega}$ is of size $I^{N-1} \times R^2$. The cost of computing their product is $O(I^N R^2)$.

- **Line 6: Computing tensor-times-matrix (TTM) product.** $\boldsymbol{\mathcal{X}}$ is an N-way tensor of size $I \times \cdots \times I$, and $\boldsymbol{Q}^{(1)}$ is of size $I \times R^2$, so this contraction costs $O(I^N R^2)$.

- **Line 11: Compute projections in for loop.** Each of these costs $I^{N-n+1}R^3$. The total cost for all iterations for $n = 2, \ldots, N-1$ is therefore

$$R^3(I^{N-1} + I^{N-2} + \cdots + I^2) \leq 2R^3 I^{N-1},$$

  where we used the assumption that $N < I$.

- **Line 15: Compute TTM product in for loop.** Each of these costs $I^{N-n+1}R^3$. The total cost for all iterations is therefore

$$R^3(I^{N-1} + I^{N-2} + \cdots + I^2) \leq 2R^3 I^{N-1},$$

  where we used the assumption that $N < I$.

Adding these costs up, we get a total cost of $O(I^N R^2)$.

C.2.5 TR-ALS-Sampled

In our calculations below, we refer to steps in Algorithm 8 in the main paper.

Upfront costs of TR-ALS-Sampled:

- **Line 1: Initializing cores.** This is the same as for TR-ALS, namely $O(NIR^2)$.
- **Line 2: Compute distributions.** For each $n = 2, \ldots, N$, this involves computing the economic SVD of an $I \times R^2$ matrix for a cost $O(IR^4)$, and then computing $\boldsymbol{p}^{(n)}$ from the left singular vectors for a cost of $O(IR)$. This yields a total cost for this line of $O(NIR^4)$.

We ignore the cost of the sampling in Line 6 since it is typically very fast.

Costs per outer loop iteration of TR-ALS-Sampled:

- **Line 7: Compute sampled unfolded subchain.** The main cost for this line is computing the product of a sequence of $N - 1$ matrices of size $R \times R$, for each of the J sampled slices (see Figure 7.2 in the main paper). This costs $O(NR^3J)$ per inner loop iteration, or $O(N^2R^3J)$ per outer loop iteration.
- **Line 8: Sample input tensor.** This step requires copying $O(IJ)$ elements from the input tensor. For one iteration of the outer loop, this is $O(NIJ)$ elements. In practice this step together with the least squares solve are the two most time consuming since it involves sampling from a possibly very large array.
- **Line 9: Solve least squares problem.** The cost is computed in the same way as the least squares solve in TR-ALS, but with the large dimension I^{N-1} replaced by J. The cost per least squares problem is therefore $O(IJR^2)$, or $O(NIJR^2)$ for one iteration of the outer loop.
- **Line 10: Update distributions.** For one iteration of the outer loop this costs $O(NIR^4)$.

Adding these costs and simplifying, we arrive at a cost

$$O(NIR^4 + \#\text{iter} \cdot NIJR^2). \tag{C.9}$$

If ε and δ are small enough, then (7.5) simplifies to $J > 4R^{2N}/(\varepsilon\delta)$. Plugging this into (C.9) gives us the complexity in Table 7.1.

C.3 Links to datasets

- The Pavia University dataset was downloaded from
  `http://lesun.weebly.com/hyperspectral-data-set.html`.
- The Washington DC Mall dataset was downloaded from
  `https://engineering.purdue.edu/~biehl/MultiSpec/hyperspectral.html`.
- The Park Bench video was downloaded from
  `https://www.pexels.com/video/man-sitting-on-a-bench-853751`.
  The video, which is in color, was made into grayscale by averaging the three color bands.
- The Tabby Cat video was downloaded from
  `https://www.pexels.com/video/video-of-a-tabby-cat-854982/`.
  The video, which is in color, was made into grayscale by averaging the three color bands.
- The Red Truck images are part of the COIL-100 dataset, which was downloaded from
  `https://www1.cs.columbia.edu/CAVE/software/softlib/coil-100.php`.

C.4 Additional experiment details

Here we provide additional experiment details, including how the number of ALS iterations and appropriate sketch rates are determined for the experiments in Section 7.5.1 of the main paper.

C.4.1 Randomly generated data

First experiment When determining the number of ALS iterations, TR-ALS is run until the change in relative error is below 1e−6 or for a maximum of 500 iterations, whichever is satisfied first. The sample size J for TR-ALS-Sampled is started at 200 and incremented by 100. The embedding dimension K for rTR-ALS is started at $I/10$ and incremented by $I/20$. Incrementation is done until the error for each method is smaller than 1.2 times the TR-ALS error.

Second experiment The sketch rate J for TR-ALS-Sampled is now incremented by 1000. Incrementation is done until the error for each method is smaller than 1.02 times the TR-ALS error. A smaller factor is used compared to the first experiment since the TR-ALS error is much larger. The other settings remain the same as in the first experiment.

Remark 89 (Performance of SVD-based methods)**.** The SVD-based methods typically require much higher ranks than the ALS-based methods. To achieve a similar error to the ALS-based methods in Figures 7.3 (a) and 7.4 (a) (0.0031 and 0.94, respectively) the **original unaltered implementation** of TR-SVD by Mickelin and Karaman [153] requires average TR ranks (i.e., $(R_1 + R_2 + R_3)/3$) in the range of 82–233 and 35–84, respectively. TR-SVD-Rand does poorly for the same reason.

Remark 90 (Empirical vs. theoretical complexity)**.** As discussed in Section 7.4.3, the benefit of our proposed method is that it has a lower complexity than the competing methods. In particular, it avoids the I^N factors in the complexity expression. It may therefore seem surprising that our method is not the fastest in the experiments. For example, in Figure 7.3 (b) our method is always slower than TR-SVD-Rand, and in Figure 7.4 (b) it is always slower than both TR-SVD and TR-SVD-Rand. The reason for this seeming discrepancy is that we only consider a small range of I values ($I \in [100, 500]$) in those figures and therefore the plots will not necessarily reflect the leading order complexities that are given in Table 7.1. The main cost in TR-SVD-Rand is matrix multiplication which is very efficient, so the hidden constant in the complexity for TR-SVD-Rand is small, and this helps explain why it is faster than our method for the relatively small I values used

in Figures 7.3 and 7.4. Similar comments also apply to the other experiments.

C.4.2 Highly oscillatory functions

To determine the number of iterations for the ALS-based methods, we run TR-ALS until the change in relative error is less than 1e−3 or for a maximum of 100 iterations, whichever is satisfied first. The sample size J for TR-ALS-Sampled is started at $2R^2$ and incremented by 100. The embedding dimension K for rTR-ALS is started at 2 and incremented by 1. If $K \geq I_n$, no compression is applied to the nth dimension. The incrementation is done until the error for each method is smaller than 1.1 times the TR-ALS error.

C.4.3 Image and video data

To determine the number of iterations for the ALS-based methods, we run TR-ALS until the change in relative error is less than 1e−3 or for a maximum of 100 iterations, whichever is satisfied first. The sample size J for TR-ALS-Sampled is started at $2R^2$ and incremented by 1000. The embedding dimension K for rTR-ALS is started at $\max_{n\in[N]} I_n/10$ and incremented by $\max_{n\in[N]} I_n/20$. If $K \geq I_n$, no compression is applied to the nth dimension. The incrementation is done until the error for each method is smaller than 1.1 times the TR-ALS error.

In the experiment on the reshaped tensors, each mode (except the mode representing color channels in the Red Truck dataset) of the original tensor is split into two new modes. Some datasets are also truncated somewhat to allow for this reshaping. The details are given below.

- **Pavia Uni.** is first truncated to size $600 \times 320 \times 100$. This is done by discarding the last elements in each mode via `X = X(1:600, 1:320, 1:100)` in Matlab. The tensor is then reshaped into a $24 \times 25 \times 16 \times 20 \times 10 \times 10$ tensor. This is done by splitting each original mode into two modes via `X = reshape(X,24,25,16,20,10,10)` in Matlab.
- **DC Mall** is truncated to size $1280 \times 306 \times 190$ and then reshaped into a $32 \times 40 \times 18 \times 17 \times 10 \times 19$ tensor similarly to how the Pavia Uni. dataset is truncated and reshaped.

- **Park Bench** does not require any truncation. The original tensor is reshaped into a $24 \times 45 \times 32 \times 60 \times 28 \times 13$ tensor similarly to how the Pavia Uni. dataset is reshaped.
- **Tabby Cat** does not require any truncation. The original tensor is reshaped into a $16 \times 45 \times 32 \times 40 \times 13 \times 22$ tensor similarly to how the Pavia Uni. dataset is reshaped.
- **Red Truck** does not require any truncation. The original tensor is reshaped into a $8 \times 16 \times 8 \times 16 \times 3 \times 8 \times 9$ tensor. Here, all modes have been split into two modes, except for the mode corresponding to the three color channels which is left as it is. This is done via `X = reshape(X,8,16,8,16,3,8,9)` in Matlab.

Detailed results for the experiments on the reshaped tensors are shown in Table C.1.

Table C.1: Decomposition results for reshaped real datasets with target rank $R = 10$. The SVD-based methods cannot handle any of these reshaped datasets since they require $R_0 R_1 \leq I_1$. The ALS-based methods all fail on the reshaped Park Bench dataset due to Matlab running out of memory. Time is in seconds.

	Pavia Uni.		DC Mall		Park Bench		Tabby Cat		Red Truck	
Method	Error	Time	Error	Time	Error	Time	Error	Time	Error	Time
TR-ALS	0.28	1372.2	0.29	3947.7	✗	✗	0.15	6629.0	0.25	546.3
rTR-ALS	0.31	944.7	0.31	2575.0	✗	✗	0.17	3416.4	0.26	423.5
TR-ALS-S. (proposal)	0.31	3.4	0.31	5.8	✗	✗	0.17	2.3	0.27	5.8
TR-SVD	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗
TR-SVD-Rand	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗

C.4.4 Rapid feature extraction for classification

The ALS-based algorithms all run until the change in relative error is below 1e−4. The embedding dimension K for rTR-ALS is 20 for the first and second modes, and 200 for the fourth mode. No compression is applied to the third mode since it is already so small. TR-ALS-Sampled uses the sketch rate $J = 1000$.

Appendix D

Supplementary material for Chapter 8

D.1 Implementation of thresholding method for BMF

Zhang et al. [223, 224] present two algorithms for BMF. The penalty based version (Algorithm 1 in [223]) is available in existing Python implementations[1] . We have not been able to find an implementation of their thresholding algorithm (Algorithm 2 in [223]). For their thresholding algorithm, they give two alternative solution approaches: Discretization and gradient descent. They only use the gradient descent based variant in their experiments, since they say discretization is too computationally expensive. However, as we show below, if the discretized variant is implemented carefully, it can find the optimal solution to the thresholding approach in $O(mnr^2 \min(m, n))$ time, which is fast enough for our experiments.

For a matrix $\boldsymbol{W} \in \mathbb{R}^{m \times r}$, we define g to be the function that outputs a vector $\boldsymbol{v} = g(\boldsymbol{W}) \in \mathbb{R}^{mr}$ such that $\boldsymbol{v}$ contains all the elements in $\boldsymbol{W}$, ordered in descending order, i.e., such that $v_i \geq v_j$ whenever $i < j$. When two entries in $\boldsymbol{W}$ have the same value, which of them comes first in $\boldsymbol{v}$ does not matter, as long as this is decided in some consistent fashion. We also assume that the functions f_r and f_c take an entry v_k as input and return the row and column position, respectively, of this entry in $\boldsymbol{W}$. For example, if the number v_k corresponds to an entry in position (i, j) in $\boldsymbol{W}$, then

[1] See e.g. `http://nimfa.biolab.si` and `https://github.com/ChrisSchinnerl/pymf3`.

$f_r(v_k) = i$ and $f_c(v_k) = j$. The thresholding function $\theta : \mathbb{R} \to \{0, 1\}$ is defined as

$$\theta(x) \stackrel{\text{def}}{=} \begin{cases} 1 & \text{if } x \geq 0, \\ 0 & \text{if } x < 0. \end{cases}$$

The number η is any positive constant. Algorithm 11 provides our implementation of the thresholding method by [223]. We use a variant of it coded in Python in our experiments.

Algorithm 11: Efficient Implementation of Algorithm 2 in [224]

Data: Matrices $\boldsymbol{A}$, $\boldsymbol{W}$, $\boldsymbol{H}$ such that $\boldsymbol{A} \approx \boldsymbol{W}\boldsymbol{H}$ is a nonnegative matrix factorization
Result: Binary matrices $\widetilde{\boldsymbol{W}}$, $\widetilde{\boldsymbol{H}}$ such that $\boldsymbol{A} \approx \widetilde{\boldsymbol{W}}\widetilde{\boldsymbol{H}}$ is a binary matrix factorization

```
/* Initialization */
```
$\widetilde{\varepsilon} = \|\boldsymbol{A}\|_{\mathrm{F}}^2$
$\widetilde{\boldsymbol{W}} = \boldsymbol{0}_{m \times r}$
$\widetilde{\boldsymbol{H}} = \boldsymbol{0}_{r \times n}$
$\boldsymbol{v} = g(\boldsymbol{W})$
$\boldsymbol{k} = g(\boldsymbol{H})$
```
/* Search through grid */
```
for $p = v_2, \ldots, v_{mr}, v_{mr} + \eta$ **do**
  $\boldsymbol{W}' = \theta(\boldsymbol{W} - p)$
  for $q = k_2, \ldots, k_{nr}, k_{nr} + \eta$ **do**
    $\boldsymbol{H}' = \theta(\boldsymbol{H} - q)$
    $\boldsymbol{X} = \boldsymbol{W}'\boldsymbol{H}'$
    $\varepsilon = \|\boldsymbol{A} - \boldsymbol{X}\|_{\mathrm{F}}^2$
    Set flag to true if $\boldsymbol{X} \in \{0, 1\}^{m \times n}$, false otherwise
    if flag is false then
      break // Since $\boldsymbol{X}$ won't be binary for subsequent qs
    else if $\varepsilon < \widetilde{\varepsilon}$ **then**
      $\widetilde{\boldsymbol{W}} = \boldsymbol{W}'$
      $\widetilde{\boldsymbol{H}} = \boldsymbol{H}'$
      $\widetilde{\varepsilon} = \varepsilon$
    end
  end
end
return $\widetilde{\boldsymbol{W}}$ and $\widetilde{\boldsymbol{H}}$

The reason this algorithm works is that there are only $mr + 1$ ways to threshold $\boldsymbol{W}$ and only $nr + 1$ ways to threshold $\boldsymbol{H}$. The nested for loops search through all the thresholdings that result in nonzero values of $\boldsymbol{X}$. Moreover, since each entry of $\boldsymbol{X}$ is nondecreasing during the iterations of the inner for loop, if $\boldsymbol{X}$ is nonbinary for some p and some $q = k_i$, it will be nonbinary for that same

p and all subsequent $q = k_j$ where $j > i$. And this is why it is fine to break out of the inner for loop, since none of the following iterates are going to lead to valid factorizations. Algorithm 11 is presented at a higher level to make it easier to follow. We now discuss some implementation details that make the algorithm faster and allow us to achieve $O(mnr^2 \min(m, n))$ run time.

- Line 7: Setting $\boldsymbol{W}' = \mathbf{0}_{m \times r}$ before the outer for loop, it is sufficient to make the update $w'_{f_r(p) f_c(p)} = 1$ on this line.
- Line 9: Setting $\boldsymbol{H}' = \mathbf{0}_{r \times n}$ before the inner for loop, it is sufficient to make the update $h'_{f_r(q) f_c(q)} = 1$ on this line.
- Line 10: Setting $\boldsymbol{X} = \mathbf{0}_{m \times n}$ before the inner for loop, it is sufficient to make the update $\boldsymbol{x}_{*f_c(q)} = \boldsymbol{x}_{*f_c(q)} + \boldsymbol{w}'_{*f_r(q)}$ on this line.
- Line 11: Assuming that the column $\boldsymbol{x}_{*f_c(q)}$ was saved prior to making the update on line 10 in a vector $\boldsymbol{t}$, we can now compute

$$\varepsilon = \varepsilon + \|\boldsymbol{a}_{*f_c(q)} - \boldsymbol{x}_{*f_c(q)}\|_{\mathrm{F}}^2 - \|\boldsymbol{a}_{*f_c(q)} - \boldsymbol{t}\|_{\mathrm{F}}^2.$$

- Line 12: The value of the flag can be computed by evaluating the truth of $[\boldsymbol{x}_{*f_c(q)} \in \{0, 1\}^m]$.
- Lines 16 and 17: Instead of updating all entries in the two matrices, it is sufficient to just keep track of the pair (i, j) corresponding to the $p = v_i$ and $q = k_j$ that resulted in the new best decomposition. The matrices $\widetilde{\boldsymbol{W}}$ and $\widetilde{\boldsymbol{H}}$ to be returned can then be computed right before the return statement.

Assume $m \leq n$. The complexity of each inner loop iteration in the algorithm is $O(m)$. This is then repeated $O(mnr^2)$ times over all inner and outer loop iterations. The total cost is therefore $O(mnr^2 \cdot m)$. If $n \leq m$, we can transpose the input matrix before applying the algorithm, resulting in a complexity of $O(mnr^2 \cdot n)$. The complexity is therefore $O(mnr^2 \min(m, n))$, as claimed.

D.2 Details on baseline method

Algorithm 12 describes our baseline method in detail. On lines 1 and 2, $\boldsymbol{U}$ and $\boldsymbol{V}$ are initialized to zero matrices. On line 4, the current residual $\boldsymbol{E}$ is computed. On line 5, the least index i' is computed such that the sum of the i'th row of $\boldsymbol{E}$ is equal to the max row sum of $\boldsymbol{E}$. A similar index is computed for the columns of the residual on line 6. The update on lines 8 and 9 eliminate the densest column in the residual. Similarly, the update on lines 11 and 12 eliminate the densest row in the residual. The first if statement determines if a row or column is eliminated in such a way that the number reduction in the residual is maximized. Finally, the second if statement checks if a simple rank-1 approximation consisting of only ones yields a better approximation than the row/column elimination procedure.

Algorithm 12: Baseline method

Data: Matrix $\boldsymbol{A}$, target rank r
Result: Binary matrices $\boldsymbol{U}, \boldsymbol{V}$ such that $\boldsymbol{A} \approx \boldsymbol{U}\boldsymbol{V}^\top$ is a binary matrix factorization

1 Set $\boldsymbol{U} = \boldsymbol{0}_{m\times r}$
2 Set $\boldsymbol{V} = \boldsymbol{0}_{n\times r}$
3 **for** $k \in [r]$ **do**
4 Set $\boldsymbol{E} = \boldsymbol{A} - \boldsymbol{U}\boldsymbol{V}^\top$ `// Compute residual`
5 Let $i' = \min\{i \in [m] \ : \ \sum_j e_{ij} = \max_\ell \sum_j e_{\ell j}\}$
6 Let $j' = \min\{j \in [n] \ : \ \sum_i e_{ij} = \max_\ell \sum_i e_{i\ell}\}$
`/* Eliminate row/column in residual with most nonzeros */`
7 **if** $\sum_j e_{i'j} < \sum_i e_{ij'}$ **then**
8 Set $\boldsymbol{u}_{*k} = \boldsymbol{e}_{*j'}$
9 Set $v_{j'k} = 1$
10 **else**
11 Set $u_{i'k} = 1$
12 Set $\boldsymbol{v}_{*k} = \boldsymbol{e}_{i'*}$
13 **end**
14 **end**
`/* Check if trivial rank-1 approximation of all 1's is better */`
15 **if** $\|\boldsymbol{A} - \boldsymbol{1}_{m\times n}\|_{\mathbf{F}}^2 < \|\boldsymbol{A} - \boldsymbol{U}\boldsymbol{V}^\top\|_F^2$ **then**
16 Set $\boldsymbol{U} = [\boldsymbol{1}_{m\times 1} \ , \ \boldsymbol{0}_{m\times(r-1)}]$
17 Set $\boldsymbol{V} = [\boldsymbol{1}_{n\times 1} \ , \ \boldsymbol{0}_{n\times(r-1)}]$
18 **end**
19 **return** $\boldsymbol{U}, \boldsymbol{V}$

D.3 Algorithm for generating binary matrices

In this section, we present the algorithm we use for generating $\boldsymbol{A}$ with an exact rank-r decomposition in the synthetic experiments. It is given in Algorithm 13. The factor matrices $\boldsymbol{U}$ and $\boldsymbol{V}$ are randomly initialized with entries drawn independently from Bernoulli distributions with success probabilities p_U and p_V, respectively. We use $p_U = p_V = 0.7$ in our experiments. At this point, if $r > 1$, it may be the case that the product $\boldsymbol{A} = \boldsymbol{U}\boldsymbol{V}^\top$ is not binary. The purpose of the while loop is to eliminate nonzero entries in $\boldsymbol{U}$ and $\boldsymbol{V}$ until $\boldsymbol{U}\boldsymbol{V}^\top$ is binary. ρ is a function which returns the average of the entries in a matrix, e.g., $\rho(\boldsymbol{U}) = \sum_{ik} u_{ik}/(nr)$. For each iteration of the while loop, a nonzero entry is eliminated in the factor matrix with the highest entry average. Eventually, $\boldsymbol{A} = \boldsymbol{U}\boldsymbol{V}^\top$ will be binary, at which point the while loop terminates and $\boldsymbol{A}$ is returned.

Algorithm 13: Algorithm for generating binary matrix which has an exact rank-r BMF

Data: Matrix dimensions m, n, rank r, target initial densities $p_U, p_V \in (0, 1)$
Result: Binary matrix $\boldsymbol{A}$ which has an exact rank-r BMF

Draw all entries $u_{ik} \sim \text{Bernoulli}(p_U)$ and $v_{jk} \sim \text{Bernoulli}(p_V)$ independently
Set $\boldsymbol{A} = \boldsymbol{U}\boldsymbol{V}^\top$
while $\boldsymbol{A}$ **has an entry exceeding 1 do**
  if $\rho(\boldsymbol{U}) \geq \rho(\boldsymbol{V})$ **then**
    Let $\phi = \{i \in [m] \ : \ \exists j \in [n] \text{ satisfying } a_{ij} > 1 \text{ and } \sum_k u_{ik} > 2\}$
    Draw index $i' \in \phi$ uniformly at random
    Choose nonzero entry in the row $\boldsymbol{u}_{i'*}$ uniformly at random and set it to 0
  else
    Let $\phi = \{j \in [n] \ : \ \exists i \in [m] \text{ satisfying } a_{ij} > 1 \text{ and } \sum_k v_{jk} > 2\}$
    Draw index $j' \in \phi$ uniformly at random
    Choose nonzero entry in the row $\boldsymbol{v}_{j'*}$ uniformly at random and set it to 0
  end
  Set $\boldsymbol{A} = \boldsymbol{U}\boldsymbol{V}^\top$
end
return $\boldsymbol{A}$