UNIVERSITY OF CALIFORNIA,
IRVINE

Efficient Tensor Operations via Compression and Parallel Computation

DISSERTATION

submitted in partial satisfaction of the requirements
for the degree of

DOCTOR OF PHILOSOPHY

in Electrical and Computer Engineering

by

Yang Shi

Dissertation Committee:
Professor Sameer Singh, Chair
Professor Animashree Anandkumar
Professor Gopi Meenakshisundaram

2019

DEDICATION

To my parents, Lifu Jin and Hongwei Shi.
To my kitties, Oreo and Reeses.

Contents

List of Figures

List of Tables

List of Algorithms

ACKNOWLEDGMENTS

First of all, I would like to thank my advisor Animashree Anandkumar for her continuous support and guidance. She has been an excellent model as a researcher and a mentor. I have learned so much from her about how to be a good researcher: how to set a goal, make a plan and accomplish it step by step. She has always been inspiring and uplifting. Her insightful thoughts and advises always make me feel that I have so much to learn. I really appreciate all the efforts she made to make me improve myself. I would also like to thank my co-advisor Sameer Singh and my dissertation committee member Gopi Meenakshisundaram for their great mentorship and insightful comments.

I would like to thank my collaborators, Niranjan UN, Preteek Jain, Cris Cecka, Sheng Zha and Tommaso Furlanello. It was pleasant to work with them and learn from them. I would also like to thank my labmates, Furong Huang, Majid Janzamin, Hanie Sedghi, Forough Arabshahi, Kamyar Azizzadenesheli, Anqi Liu and Sahin Lale.

I would like to thank my friends for their company during my study journey. I would like to thank Chengyan Xu, Chenyang Zhu, and Yang Feng.

Most importantly, I would like to thank my family. They have always been supportive. I wish to thank my parents, Lifu Jin and Hongwei Shi, for their love and encouragement, without whom I would never have accomplished so much.

I thank the proceedings of Machine Learning Research for giving me permission to include Chapter 2 in my dissertation which was originally published therein. I thank Springer press for permission to include Chapter 4. I thank IEEE for giving me permission to include Chapter 5.

This dissertation was supported through Air Force grant FA9550-15-1-0221, NSF Career Award CCF-1254106 and ONR Award N00014-15-1-2737.

CURRICULUM VITAE

Yang Shi

EDUCATION

Doctor of Philosophy in Electrical and Computer Engineering **2019**
University of California, Irvine *Irvine, CA*

Master of Science in Electrical and Computer Engineering **2014**
University of Pennsylvania *Philadelphia, PA*

Bachelor of Science in Electrical Engineering **2012**
Nanjing University of Science and Technology *Nanjing, Jiangsu*

RESEARCH & WORK EXPERIENCE

Graduate Research Assistant **Jan. 2015–June. 2019**
University of California, Irvine *Irvine, CA*

Applied Scientist Intern **Jan. 2017–Dec. 2017**
Amazon AWS *Palo Alto, CA*

Graduate Research Assistant **Sept. 2013–June 2014**
University of Pennsylvania *Philadelphia, PA*

Research Assistant **March 2012–May 2012**
Technical University of Munich *Munich,Germany*

TEACHING EXPERIENCE

Teaching Assistant **2013–2014**
University of Pennsylvania *Philadelphia, PA*

REFEREED CONFERENCE PUBLICATIONS

Question-type-guided Attention in Visual Question Answering **Sept. 2018**
Yang Shi, Tommaso Furlanello, Sheng Zha, Animashree Anandkumar
European Conference on Computer Vision

Tensor Contractions with Extended BLAS Kernel on CPU and GPU **Dec. 2016**
Yang Shi, U.N. Niranjan, Animashree Anandkumar, Cris Cecka
IEEE International Conference on High Performance Computing, Data and Analytics

Tensor vs Matrix Methods: Robust Tensor Decomposition under Block Sparse Perturbations **May 2016**
Animashree Anandkumar, Prateek Jain, Yang Shi, U.N. Niranjan
Artificial Intelligence and Statistics Conference

REFEREED WORKSHOP PUBLICATIONS

Multi-dimensional Count Sketch: Dimension Reduction That Retains Efficient Tensor Operations **Dec. 2018**
Yang Shi, Animashree Anandkumar
NeurIPS Integration of Deep Learning Theories Workshop

Compact Tensor Pooling for Visual Question Answering **July 2017**
Yang Shi, Tommaso Furlanello, Animashree Anandkumar
CVPR Visual Question Answering Workshop

SOFTWARE

Question Type guided Attention for VQA Link
Python algorithm for VQA

A tutorial for VQA using MXNet Link
Jupyter notebook for VQA

Extended BLAS kernels for Tensor Contraction Link
C++ algorithm for tensor contraction

ABSTRACT OF THE DISSERTATION

Efficient Tensor Operations via Compression and Parallel Computation

By

Yang Shi

Doctor of Philosophy in Electrical and Computer Engineering

University of California, Irvine, 2019

Professor Sameer Singh, Chair

Linear algebra is the foundation of machine learning, especially for handling big data. We want to extract useful information that can represent the behavior of the data. For data with underlying known structures, it is straightforward to apply algorithms that maintain that structure. For instance, singular value decomposition (SVD) is one way to approximate low-rank matrices. The generalized SVD, tensor decomposition, is the crux of model estimation for tensors. However, not all data has a trivial structure. Multi-modality data that contains information from different sources can be complex and hard to extract the structure. A data-independent randomized algorithm, such as sketching, is the solution for this case. Under both scenarios, the information extraction process may be statistically challenging as the problems are non-convex optimization problems. More importantly, the large size and the high-dimensionality of the data have been significant obstacles in discovering hidden variables and summarizing them. Thus, how to improve high-dimensional data computation efficiency is vitally important.

This thesis contains the theoretical analysis for learning the underlying information from high-dimensional structured or non-structured data via tensor operations such as tensor decomposition and tensor sketching. It is easy to consider tensors as multi-dimensional vectors or matrices and apply vector/matrix-based algorithms to find the solution. However, these methods omit multi-dimensionality of the data and can be computational inefficient than considering the tensor as a whole. We show the superiority of our approximation algorithms over these methods from computation and memory efficiency point of views.

This thesis also discusses optimizing tensor operation computations from the high-performance computing aspect. Conventional methods treat tensors as flattened matrices or vectors. Operations between tensors may require lots of permutations and reshapes. We propose new tensor algebra computation routines that avoid the prepossessing as much as possible. The value of this approach and its applications are recognized by NVIDIA. The proposed interface exists in the CuBlas 8.0.

Chapter 1

Introduction

The trinity of machine learning and deep learning consists of data, algorithm, and computation. Data is vitally important because without data, we can not verify the correctness of the model we designed. The data is inherently multi-dimensional. We call multi-dimensional data tensors. The algorithm is the core of the model. Different tasks and data require different algorithms. Last but not least, computation is gaining more attention. As the models expand and more data be collected, computation power is the crux of modern technology development. Specifically, how to approximate tensors with few parameters using different algorithms and how to ensure efficient tensor computations in machine learning and deep learning are the main concerns of this thesis.

Given the importance of data approximation in the aspect of computation efficiency and privacy concerns, many methods have been developed to use fewer parameters in the estimation of the original data. As one might expect, data with a particular structure may require a specific approximation method. Truncated singular value decomposition (SVD) is an approximation method to represent the rotation and rescale of the data. Rotation and rescale represent the structure of the data in such a case. However, this decomposition may

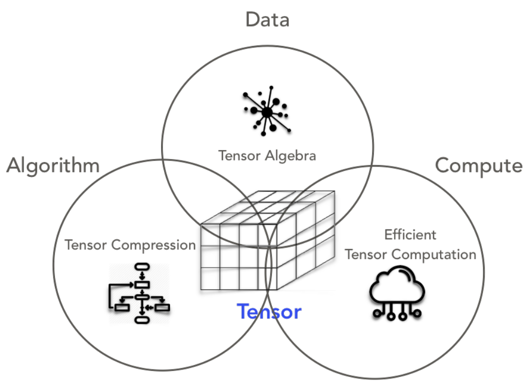


Figure 1.1: Trinity of AI.

not fit for specific data that has sparsity/non-negativity constraints. In contrast to matrix techniques which make stringent assumptions on underlying structure, sketching is designed for compressing vector-valued data with almost no assumptions.

In this thesis, we first discuss how to find hidden structures from high-dimensional data using latent variable models. Specifically, we use the method of moment: a technique to discover hidden parameters from the observed aggregated statistics of the data. This algorithm requires tensor decomposition: decomposition of the higher-order moments. Secondly, we propose a new sampling method that does not depend on the underlying data structure. This hash-based sketching method shows advantages in many applications. In both cases, we focus on comparing our algorithms with existing models which consider tensors as slices of matrices and vectors. Thirdly, we present a real-world application: Visual Question Answering, where data from different feature channels have to be combined through tensor operations. We propose a new data structure so that it explores efficient feature representations for multi-modality models.

Data dimensionality reduction is one way to improve computation efficiency when the data is used in a huge network, such as deep neural nets. However, this might not satisfy the computation speed requirement. Computation capability is the bottleneck. Conventional computing methods treat tensors as flattened matrices or vectors. Operations between tensors may require lots of permutations and reshapes. We propose new tensor algebra computation routines that avoid the prepossessing as much as possible.

The rest of this chapter is organized as follows. We first summarize the problems we analyzed and our contributions in Section 1.1. In Section 1.2, we provide background for tensor algebra since it is fundamental for the whole thesis. Finally, we discuss the organization of the dissertation in Section 1.3.

1.1 Summary of Contributions

1.1.1 Tensor Robust Principle Component Analysis

In Chapter 2, we consider the tensor robust principle component analysis, which recovers a low rank tensor subject to gross corruptions. Given an input tensor $T = L^* + S^*$, we aim to recover both L^* and S^*, where L^* is a low rank tensor with CP-form: $L^* = \sum_{i=1}^{r} \sigma_i^* u_i \otimes u_i \otimes u_i$ and S^* is a sparse tensor. The above problem arises in numerous applications such as image and video denoising, multi-task learning, and robust learning of latent variable models with grossly-corrupted moments.

One can solve the robust tensor problem using matrix methods. Robust matrix PCA can be applied either to each matrix slice of the tensor or to the matrix obtained by flattening the tensor. However, such matrix methods ignore the tensor algebraic constraints or the CP-rank constraints, which differ from the matrix rank constraints.

We propose a non-convex iterative method, termed RTD, that maintains low rank and sparse estimates $\widehat{L}$, $\widehat{S}$, which are alternately updated. The low rank estimate $\widehat{L}$ is updated through the eigenvector computation of $T - \widehat{S}$, and the sparse estimate is updated through (hard) thresholding of the residual $T - \widehat{L}$. As a main result, we prove convergence to the global

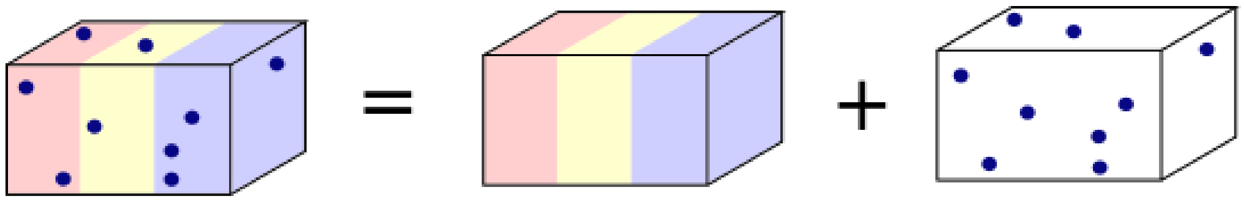


Figure 1.2: Tensor robust principle component analysis.

optimum $\{L^*, S^*\}$ for RTD under an incoherence assumption on L^*, and a bounded sparsity level for S^*. We compare our RTD method with matrix robust PCA, carried out either on matrix slices of the tensor, or on the *flattened* tensor. We prove our RTD method is superior and can handle higher sparsity levels in the noise tensor S^*, when it is block structured.

1.1.2 Higher-order Count Sketch

Sketching is a randomized dimensionality-reduction method that aims to preserve relevant information in large-scale datasets. Count sketch(CS) is a simple popular sketch which uses a randomized hash function to achieve compression. In Chapter 3, we propose a novel extension known as Higher-order Count Sketch (HCS).

While count sketch uses a single hash function, HCS uses multiple (smaller) hash functions for sketching. HCS reshapes the input (vector) data into a higher-order tensor and employs a tensor product of the random hash functions to compute the sketch. This results in an exponential saving (with respect to the order of the tensor) in the memory requirements of the hash functions, under certain conditions on the input data. Furthermore, when the input data itself has an underlying structure in the form of various tensor representations such as the Tucker decomposition, we obtain significant advantages. We derive efficient (approximate) computation of various tensor operations such as tensor products and tensor contractions directly on the sketched data. Thus, HCS is the first sketch to fully exploit the multi-dimensional nature of higher-order tensors.

We compare HCS and CS for tensor product and tensor contraction compression using

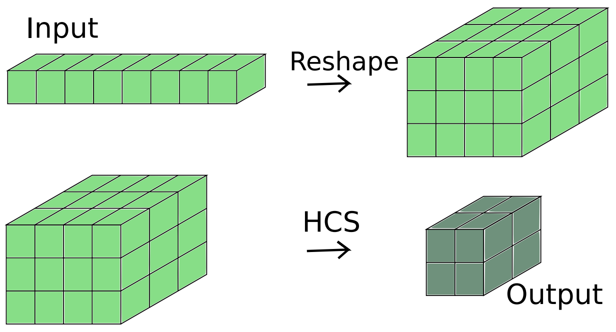


Figure 1.3: Higher-order count sketch reshapes input vector into higher-order tensor and sketches it into a (smaller) tensor of the same order.

synthetic data. HCS outperforms CS in terms of computation efficiency and memory usage: it uses 200× less compression time and 40× less memory while keeping the same recovery error, compared to CS. We apply HCS to tensorized neural networks where we replace fully connected layers with sketched tensor operations. We achieve nearly state of the art accuracy with significant compression on the image classification benchmark.

1.1.3 Feature Pooling through Tensor Operations

In Chapter 4, we consider a deep learning application, named Visual Question Answering (VQA). VQA focus on providing a natural language answer given any image and any free-form natural language question. It requires integration of feature maps with drastically different structures and focus of the correct regions. Image descriptors have structures at multiple spatial scales, while lexical inputs inherently follow a temporal sequence and naturally cluster into semantically different question types.

A lot of previous works use complex models to extract feature representations but neglect to use high-level information summary such as question types in learning. In this work, we propose Question Type-guided Attention (QTA). It utilizes the information of question type to dynamically balance between bottom-up and top-down visual features, respectively extracted from ResNet and Faster R-CNN networks. We experiment with multiple VQA

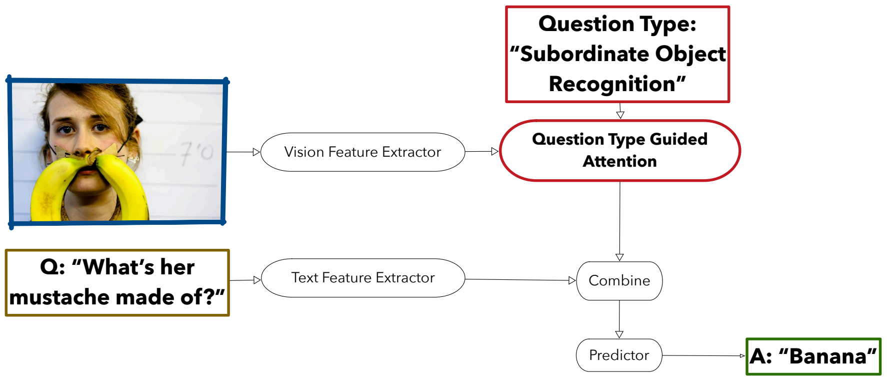

Question Type: "Subordinate Object Recognition"
Question Type Guided Attention
Vision Feature Extractor
Q: "What's her mustache made of?"
Text Feature Extractor
Combine
Predictor
A: "Banana"

Figure 1.4: General VQA network with QTA.

architectures with extensive input ablation studies over the TDIUC dataset and show that QTA systematically improves the performance by more than 5% across multiple question type categories such as "Activity Recognition", "Utility" and "Counting" on TDIUC dataset. By adding QTA on the state-of-art model MCB, we achieve 3% improvement in overall accuracy. Finally, we propose a multi-task extension to predict question types which generalizes QTA to applications that lack question type, with a minimal performance loss.

1.1.4 Efficient Tensor Contraction Primitives

In Chapter 5, we propose and study library-based communication-avoiding approaches for performing tensor contractions. Conventional approaches for computing general tensor contractions rely on matricization, the logical or explicit restructuring of the data so that the computation can be performed with a sequence of Basic Linear Algebra Subroutine (Blas) library calls.

We introduce a new Blas primitive, known as StridedBatchedGemm, that allows the majority of tensor contractions to be computed without any explicit memory motion. We begin by focusing on single-index contractions involving all the possible configurations of second-order and third-order tensors. Through systematic benchmarking, we demonstrate

Table 1.1: An example of tensor contraction computation procedure using conventional and our new tensor contraction primitive

Task	Conventional	Our method
$C_{mnp} = A_{km}B_{pkn}$	$A_{km} \rightarrow A_{mk}$ $B_{pkn} \rightarrow B_{kpn}$ $C_{mnp} \rightarrow C_{mpn}$ $C_{mpn} \rightarrow A_{mk}B_{kpn}$ $C_{mpn} \rightarrow C_{mnp}$	$C_{m[n]p} = A^T_{km}B^T_{pk[n]}$

that our approach can achieve 10x speedup on a K40c GPU and 2x speedup on dual-socket Haswell-EP CPUs, using Mkl and CuBlas respectively, for small and moderate tensor sizes. This is relevant in many machine learning applications such as deep learning, where tensor sizes tend to be small, but require numerous tensor contraction operations to be performed successively. We also demonstrate performance improvement using our approach in direct benchmarks to an application study: the Tucker decomposition. We show that using our kernels yields atleast an order of magnitude speedup as compared to state-of-the-art libraries.

1.2 Tensor Preliminaries

We denote vectors by lowercase letters, matrices and higher-order tensors(multi-dimensional data structures) by uppercase letters. The *order* of a tensor is the number of modes it admits. For example, $T \in \mathbb{R}^{n_1 \times \cdots \times n_N}$ is an Nth-order tensor because it has N modes. A scalar is a zeroth-order tensor, a vector is a first-order tensor, a matrix is a second-order tensor with the rows being the first mode and columns being the second mode, and a three-way array (say $A \in \mathbb{R}^{m \times n \times p}$) is a third-order tensor with the first, second and third modes indexed by m, n, and p, respectively. *fibers* is the higher-order analogue of a matrix row or column in tensors. A fiber is obtained by fixing all but one of the indices of the tensor. For example, for a third order tensor $T \in \mathbb{R}^{n \times n \times n}$, the mode-1 fiber is given by $T(:, j, l)$. Similarly, *slices* are obtained by fixing all but two of the indices of the tensor. For example, for the third

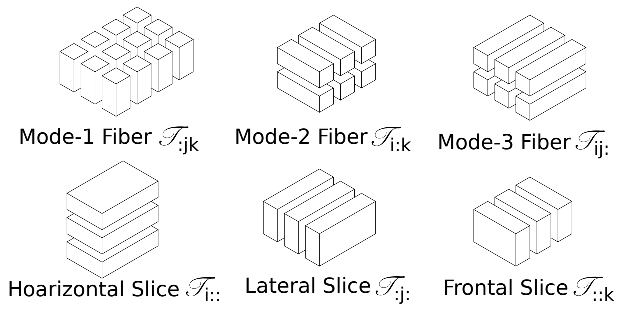

Mode-1 Fiber $\mathscr{T}_{:jk}$
Mode-2 Fiber $\mathscr{T}_{i:k}$
Mode-3 Fiber $\mathscr{T}_{ij:}$
Hoarizontal Slice $\mathscr{T}_{i::}$
Lateral Slice $\mathscr{T}_{:j:}$
Frontal Slice $\mathscr{T}_{::k}$

Figure 1.5: Fibers and slices of a third-order tensor.

order tensor T, the slices along third mode are given by $T(:,:,l)$. A *flattening* of tensor T along mode k is a matrix M whose columns correspond to mode-k fibres of T. We show different ways to slice a third-order tensor in Figure 1.5.

Matrix product between $A \in \mathbb{R}^{m\times r}$ and $B \in \mathbb{R}^{r\times n}$ is defined as $C = AB = \sum_{i=1}^{r} A_{:i} \otimes B_{i:}$, $C \in \mathbb{R}^{m\times n}$. **Tensor contraction** (used more often as Einstein summation in physics community) can be seen as an extension of matrix product in higher-dimensions. We define a general tensor contraction between $A \in \mathbb{R}^{a_1\times\cdots\times a_p}$ and $B \in \mathbb{R}^{b_1\times\cdots\times b_q}$ as

$$C_{\mathcal{L}} = A_{\mathcal{P}} B_{\mathcal{Q}} = A_{\mathcal{M}\mathcal{R}} B_{\mathcal{R}\mathcal{N}} = \sum_{\mathcal{R}} A_{:\mathcal{R}} \otimes B_{\mathcal{R}:} \tag{1.1}$$

where $\mathcal{P}, \mathcal{Q}, \mathcal{L}$ are ordered sequences of indices such that $\mathcal{P} = \{a_1 \times \cdots \times a_p\}$, $\mathcal{Q} = \{b_1 \times \cdots \times b_q\}$, $\mathcal{L} = (\mathcal{P} \cup \mathcal{Q}) \backslash (\mathcal{P} \cap \mathcal{Q})$, $\mathcal{R} = \mathcal{P} \cap \mathcal{Q}$, $\mathcal{M} = \mathcal{P} \backslash (\mathcal{P} \cap \mathcal{Q})$, $\mathcal{N} = \mathcal{Q} \backslash (\mathcal{P} \cap \mathcal{Q})$. The indices in $\mathcal{R}$ are called contracted indices. The indices in $\mathcal{L}$ are called free indices. *The p-mode matrix product* of a tensor $T \in \mathbb{R}^{n_1\times\cdots\times n_N}$ with a matrix $U \in \mathbb{R}^{m\times n_p}$ is denoted by $T_{\times p}U$and is of size $n_1 \times \cdots n_{p-1} \times m \times n_{p+1} \times \cdots \times n_N$. Element-wise it calculates: $(T_{\times p}U)_{i_1\cdots i_{p-1} j i_{p+1}\cdots i_N} = \sum_{i_p=1}^{n_p} T_{i_1\cdots i_N} U_{j i_p}$.

Tensor product is known as outer product in vectors case. It computes every bilinear composition from inputs. We denote the operation with $\otimes$. The tensor product result has dimension equal to the product of the dimensions of the inputs. An Nth-order tensor is a result of the tensor product of N vector spaces: $A = v_1 \otimes \cdots \otimes v_N \in \mathbb{R}^{n_1\times\cdots\times n_N}$, where

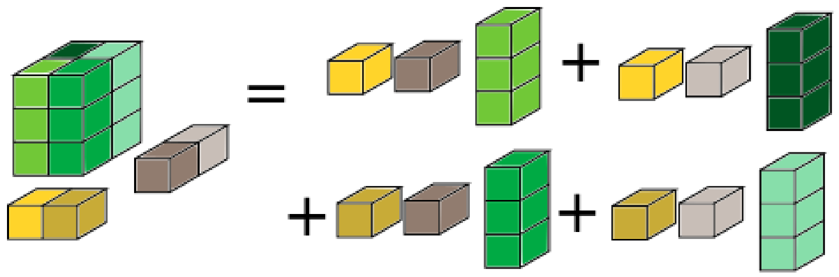


Figure 1.6: Tensor contraction example: given $A \in \mathbb{R}^{2\times2\times3}$, $u \in \mathbb{R}^{2\times1}, v \in \mathbb{R}^{2\times1}$, $A_{\times1}u_{\times2}v \in \mathbb{R}^{1\times1\times3}$.

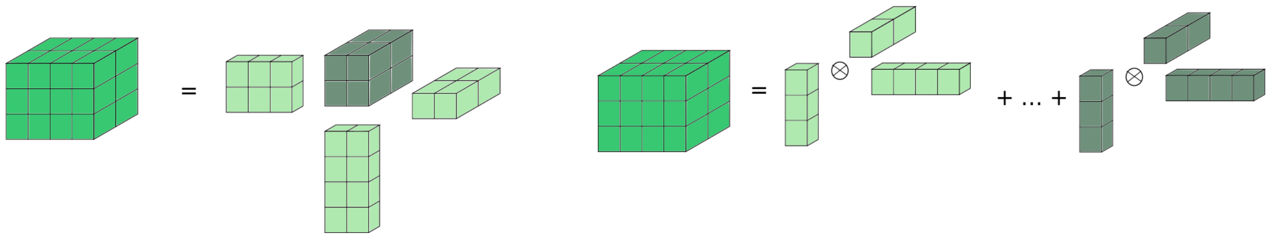


Figure 1.7: Tucker(left) and CP(right) decomposition of a third-order tensor.

$v_i \in \mathbb{R}^{n_i}, i \in 1,\dots,N$.

Tensor decomposition is an extension of matrix decomposition to higher orders. The Tucker decomposition [98] is analogous to principal component analysis. It decomposes a tensor as a core tensor contracted with a matrix along each mode. For instance, a third-order tensor $T \in \mathbb{R}^{n_1\times n_2\times n_3}$ has the Tucker decomposition:

$$T = G_{\times1}U_{\times2}V_{\times3}W \tag{1.2}$$

where $G \in \mathbb{R}^{r_1\times r_2\times r_3}$, $U \in \mathbb{R}^{n_1\times r_1}$, $V \in \mathbb{R}^{n_2\times r_2}$, and $W \in \mathbb{R}^{n_3\times r_3}$. CANDECOMP/-PARAFAC(CP) [42] is a special case of a Tucker-form tensor, where the core tensor is a sparse tensor that only has non-zero values on the superdiagnoal. It can be represented as a sum of rank-1 tensors. In previous example, $T = \sum_{i=1}^{r} G_{iii}U_i \otimes V_i \otimes W_i$. Figure 1.7 shows the Tucker-form and the CP-form of a third-order tensor.

1.3 Organization of the Dissertation

In my thesis, I will first discuss about how to find hidden structures for low-rank tensors with gross corruptions in Chapter 2. In Chapter 3, I propose Higher-order Count Sketch(HCS), a data-independent algorithm that can find a compact representation of the original tensor. Moreover, it allows for certain operations to be accurately carried out in the low-dimensional sketched space. I will switch to analyze multimodality feature data in a real-world application in Chapter 4. In particular, I work on data with visual and lexical information. I show how to effectively find compact features for this type of data using sketching techniques. Finally, I discuss efficient tensor contraction methods from high-performance computing view of point in Chapter 5. This chapter introduces a new computing kernel, that allows the majority of tensor contractions to be computed without explicit memory motion. Thus, it improves the computation efficiency dramatically.

Chapter 2

Robust Principle Component Analysis through Tensor Decomposition

In this chapter, we develop a robust tensor decomposition method, which recovers a low rank tensor subject to gross corruptions. Given an input tensor $T = L^* + S^*$, we aim to recover both L^* and S^*, where L^* is a low rank tensor and S^* is a sparse tensor.

$$T = L^* + S^*, \quad L^* = \sum_{i=1}^{r} \sigma_i^* u_i \otimes u_i \otimes u_i \tag{2.1}$$

The above form of L^* is known as the Candecomp/Parafac or the CP-form. We assume that L^* is a rank-r orthogonal tensor, i.e., $\langle u_i, u_j \rangle = 1$ if $i = j$ and 0 otherwise. The above problem arises in numerous applications such as image and video denoising [49], multi-task learning, and robust learning of latent variable models (LVMs) with grossly-corrupted moments.

The matrix version of (2.1), viz., decomposing a matrix into sparse and low rank matrices, is known as robust principal component analysis (PCA). It has been studied extensively [22, 20, 47, 75]. Both convex [22, 20] as well as non-convex [75] methods have been proposed with provable recovery.

One can, in fact, attempt to solve the robust tensor problem in (2.1) using matrix methods. In other words, robust matrix PCA can be applied either to each matrix slice of the tensor, or to the matrix obtained by flattening the tensor. However, such matrix methods ignore the tensor algebraic constraints or the CP rank constraints, which differ from the matrix rank constraints. There are however a number of challenges to incorporating the tensor CP rank constraints. Enforcing a given tensor rank is NP-hard [44], unlike the matrix case, where low rank projections can be computed efficiently. Moreover, finding the best convex relaxation of the tensor CP rank is also NP-hard [44], unlike the matrix case, where the convex relaxation of the rank, viz., the nuclear norm, can be computed efficiently.

2.1 Summary of Results

Proposed method: We propose a non-convex iterative method, termed RTD, that maintains low rank and sparse estimates $\widehat{L}$, $\widehat{S}$, which are alternately updated. The low rank estimate $\widehat{L}$ is updated through the eigenvector computation of $T-\widehat{S}$, and the sparse estimate is updated through (hard) thresholding of the residual $T-\widehat{L}$.

Tensor Eigenvector Computation: Computing eigenvectors of $T-\widehat{S}$ is challenging as the tensor can have arbitrary "noise" added to an orthogonal tensor and hence the techniques of [8] and similar work do not apply as they only guarantee an approximation to the eigenvectors up to the "noise" level. Results similar to the shifted power method of [56] should apply for our problem, but their results hold in an arbitrarily small centered at the true eigenvectors, and the size of the ball is typically not well-defined. In this work, we provide a simple variant of the tensor power method based on gradient ascent of a regularized variational form of the eigenvalue problem of a tensor. We show that our method converges to the true eigenvectors at a linear rate when initialized within a reasonably small ball around eigenvectors. See Theorem 2.2 for details. We believe that our method and analysis for the tensor eigenvector computation should be of independent interest as well.

Guaranteed recovery: As a main result, we prove convergence to the global optimum $\{L^*, S^*\}$ for RTD under an incoherence assumption on L^*, and a bounded sparsity level for S^*. These conditions are similar to the conditions required for the success of matrix robust PCA. We also prove fast linear convergence rate for RTD, i.e. we obtain an additive ϵ-approximation in $O(\log(1/\epsilon))$ iterations.

Superiority over matrix robust PCA: We compare our RTD method with matrix robust PCA, carried out either on matrix slices of the tensor, or on the *flattened* tensor. We prove our RTD method is superior and can handle higher sparsity levels in the noise tensor S^*, when it is block structured. Intuitively, each block of noise represents correlated noise which persists for a subset of slices in the tensor. For example, in a video if there is an occlusion then the occlusion remains fixed in a small number of frames. In the scenario of moment-based estimation, S^* represents gross corruptions of the moments of some multivariate distribution, and we can assume that it occurs over a small subset of variables.

We prove that our tensor methods can handle a much higher level of block sparse perturbations, when the overlap between the blocks is controlled (e.g. random block sparsity). For example, for a rank-1 tensor, our method can handle $O(n^{17/12})$ corrupted entries per fiber of the tensor (i.e. row/column of a slice of the tensor). In contrast, matrix robust PCA methods only allows for $O(n)$ corrupted entries, and this bound is tight [75]. We prove that even better gains are obtained for RTD when the rank r of L^* increases, and we provide precise results in this chapter. Thus, our RTD achieves best of both the worlds: better accuracy and faster running times.

We conduct extensive simulations to empirically validate the performance of our method and compare it to various matrix robust PCA methods. Our synthetic experiments show that our tensor method is 2-3 times more accurate, and about 8-14 times faster, compared to matrix decomposition methods. On the real-world *Curtain* dataset, for the activity detection or the foreground filtering task, our tensor method obtains better recovery with a 10% speedup.

Overview of techniques: At a high level, the proposed method RTD is a tensor analogue of the non-convex matrix robust PCA method in [75]. However, both the algorithm (RTD) as well as the analysis of RTD is significantly challenging due to two key reasons: a) there can be significantly more structure in the tensor problem that needs to be exploited carefully using structure in the noise (for example, we propose and analyze a block structure on the noise), b) unlike matrices, tensors can have an exponential number of eigenvectors [21].

In fact, for tensors, it is non-trivial to establish recovery of a given eigenvector using computationally efficient methods. It is not immediately clear how sparse perturbations S^* affect the recovery of the low rank tensor L^*, as our algorithm progresses.

We would like to stress that we need to establish convergence to the globally optimal solution $\{L^*, S^*\}$, and not just to a local optimum, despite the non-convexity of the decomposition problem. Intuitively, if we are in the basin of attraction of the global optimum, it is natural to expect that the estimates $\{\widehat{L}, \widehat{S}\}$ under RTD are progressively refined, and get closer to the true solution $\{L^*, S^*\}$. However, characterizing this basin, and the conditions needed to ensure we "land" in this basin is non-trivial and novel.

As mentioned above, our method alternates between finding low rank estimate $\widehat{L}$ on the residual $T - \widehat{S}$ and viceversa. The main steps in our proof are as follows: *(i)* For updating the low rank estimate, we propose a modified tensor power method, and prove that it converges to one of the eigenvectors of $T - \widehat{S}$. In addition, the recovered eigenvectors are "close" to the components of L^*. *(ii)* When the sparse estimate $\widehat{S}$ is updated through hard thresholding, we prove that the support of $\widehat{S}$ is contained within that of S^*. *(iii)* We make strict progress in each epoch, where $\widehat{L}$ and $\widehat{S}$ are alternately updated.

In order to prove the first part, we establish that the proposed method performs gradient ascent on a regularized variational form of the eigenvector problem. We then establish that the regularized objective satisfies local strong convexity and smoothness. We also establish that by having a polynomial number of initializations, we can recover vectors that are "reasonably" close to eigenvectors of $T - \widehat{S}$. Using the above two facts, we establish a linear convergence to the true eigenvectors of $T - \widehat{S}$, which are close to the components of L^*.

For step *(ii)* and *(iii)*, we show that using an intuitive block structure in the noise, we can bound the affect of noise on the eigenvectors of the true low-rank tensor L^* and show that the proposed iterative scheme refines the estimates of L^* and converge to L^* at a linear rate.

2.2 Related Work

Robust matrix decomposition: In the matrix setting, the above problem of decomposition into sparse and low rank parts is popularly known as *robust PCA*, and has been studied in a number of works ([22],[20]). The popular method is based on convex relaxation, where the low rank penalty is replaced by nuclear norm and the sparsity is replaced by the ℓ_1 norm. However, this technique is not applicable in the tensor setting, when we consider the *CP rank*. There is no convex surrogate available for the CP rank, and in fact, determining the CP rank of a general tensor is NP-hard [44]. In this chapter, we consider non-convex methods for robust tensor decomposition.

Recently a non convex method for robust PCA is proposed in [75]. It involves alternating steps of PCA and thresholding of the residual. Our proposed tensor method can be seen as a tensor analogue of the method in [75]. However, the analysis is very different, since the optimization landscape for tensor decomposition differs significantly from that of the matrix.

Convex Robust Tucker decomposition: Previous works which employ convex surrogates for tensor problems, such as tensor completion or robust tensor decomposition, employ a different notion of rank, known as the *Tucker* rank or the *multi*-rank, e.g. [97, 34, 48, 59, 39, 40, 36]. However, the notion of a multi-rank is based on ranks of the matricization or flattening of the tensor, and thus, this method does not exploit the tensor algebraic constraints. The problem of robust tensor PCA is specifically tackled in [40, 36]. In [36], convex and non-convex methods are proposed based on Tucker rank, but there are no guarantees on if it yields the original tensors L^* and S^* in (2.1). In [40], *Schatten*-1 norm is used for low rank part, and they prove success under restricted eigenvalue conditions. However, these conditions are opaque and it is not clear regarding the level of sparsity that can be handled. Moreover, computing the Schatten-1 norm for large tensors is infeasible. The other works consider the recovery problem with missing entries. This is different from the robust decomposition problem, where we do not know the locations of the corruptions.

Sum of squares: Barak et al [14] recently consider CP-tensor completion using algorithms based on the sum of squares hierarchy. However, these algorithms are expensive (a high order polynomial in the observed dimension, and possibly even exponential time when there are many missing entries). In contrast, in this chapter, we consider simple iterative methods based on the power method that are efficient and scalable for large datasets. It is however unclear if the sum of squares algorithm improves the result for the block sparse model considered here.

Robust tensor decomposition: Shah et al [86] consider robust tensor decomposition method using a randomized convex relaxation formulation. Under their random sparsity

Algorithm 1 $(\widehat{L},\ \widehat{S})$ = RTD (T, δ, r, β): Tensor Robust PCA

```
1: Input: Tensor $T \in \mathbb{R}^{n\times n\times n}$, convergence criterion $\delta$, target rank $r$, thresholding scale
   parameter $\beta$. $P_l(A)$ denote estimated rank-$l$ approximation of tensor $A$, and let $\sigma_l(A)$
   denote the estimated $l^{\text{th}}$ largest eigenvalue using Procedure 2. $HT_\zeta(A)$ denotes hard-
   thresholding, i.e. $\mathcal{H}_\zeta(A))_{ijk} = A_{ijk}$ if $|A_{ijk}| \geq \zeta$ and 0 otherwise.
2: Set initial threshold $\zeta_0 \leftarrow \beta\sigma_1(T)$ and estimates $S^{(0)} = \mathcal{H}_{\zeta_0}(T - L^{(0)})$.
3: for Stage $l = 1$ to $r$ do
4:     for $t = 0$ to $\tau = 10\log\left(n\beta\left\|T - S^{(0)}\right\|_2/\delta\right)$ do
5:         $L^{(t+1)} = P_l(T - S^{(t)})$.
6:         $S^{(t+1)} = \mathcal{H}_\zeta(T - L^{(t+1)})$.
7:         $\zeta_{t+1}=\beta(\sigma_{l+1}(T-S^{(t+1)}) + \left(\frac{1}{2}\right)^t \sigma_l(T - S^{(t+1)}))$.
8:     If $\beta\sigma_{l+1}(L^{(t+1)}) < \frac{\delta}{2n}$, then return $L^{(\tau)}, S^{(\tau)}$, else reset $S^{(0)} = S^{(\tau)}$
9: Return: $\widehat{L} = L^{(\tau)}, \widehat{S} = S^{(\tau)}$
```

model, their algorithm provides guaranteed recovery as long as the number of non-zeros per fibre is $O(\sqrt{n})$. This is in contrast to our method which potentially tolerates upto $O(n^{17/12})$ non-zero sparse corruptions per fibre.

2.3 Proposed Algorithm

Notations: Let $[n] := \{1, 2, \ldots, n\}$, and $\|v\|$ denote the ℓ_2 norm of vector v. For a matrix or a tensor M, $\|M\|$ refers to spectral norm and $\|M\|_\infty$ refers to maximum absolute entry.

RTD **method:** We propose non-convex algorithm RTD for robust tensor decomposition, described in Algorithm 1. The algorithm proceeds in stages, $l = 1, \ldots, r$, where r is the target rank of the low rank estimate. In l^{th} stage, we consider alternating steps of low rank projection $P_l(\cdot)$ and hard thresholding of the residual, $\mathcal{H}(\cdot)$. For computing $P_l(\widetilde{L})$, that denotes the l leading eigenpairs of $\widetilde{L}$, we execute gradient ascent on a function $f(v) = \widetilde{L}(v, v, v) - \lambda\|v\|^4$ with multiple restarts and deflation (see Procedure 2).

Procedure 2 $\{\widehat{L}_l, (\widehat{u_j}, \lambda_j)_{j\in[l]}\} = P_l(T)$: GradAscent (Gradient Ascent Method)

```
1: Input: Symmetric tensor $T \in \mathbb{R}^{n\times n\times n}$, target rank $l$, exact rank $r$, $N_1$ number of initializations or restarts, $N_2$ number of power iterations for each initialization. Let $T_1 \leftarrow T$.
2: for $j = 1, \ldots, r$ do
3:     for $i = 1, \ldots, N_1$ do
4:         $\theta \sim \mathcal{N}(0, I_n)$. Compute top singular vector $u$ of $T_j(I, I, \theta)$. Initialize $v_i^{(1)} \leftarrow u$. Let $\lambda = T_j(u, u, u)$.
5:         repeat
6:             $v_i^{(t+1)} \leftarrow T_j(I, v_i^{(t)}, v_i^{(t)})/\|T_j(I, v_i^{(t)}, v_i^{(t)})\|_2$    ▷ Run power method to land in spectral ball
7:             $\lambda_i^{(t+1)} \leftarrow T_j(v_i^{(t+1)}, v_i^{(t+1)}, v_i^{(t+1)})$
8:         until $t = N_2$
9:         Pick the best: reset $i \leftarrow \arg\max_{i\in[N_1]} T_j(v_i^{(t+1)}, v_i^{(t+1)}, v_i^{(t+1)})$ and $\lambda_i = \lambda_i^{(t+1)}$ and $v_i = v_i^{(t+1)}$.
10:        Deflate: $T_j \leftarrow T_j - \lambda_i v_i \otimes v_i \otimes v_i$.
11: for $j = 1, \ldots, r$ do
12:     repeat
13:         Gradient Ascent iteration: $v_j^{(t+1)} \leftarrow v_j^{(t)} + \frac{1}{4\lambda(1+\lambda/\sqrt{n})} \cdot \left(T(I, v_j^{(t)}, v_j^{(t)}) - \lambda\|v_j^{(t)}\|^2 v_j^{(t)}\right)$.
14:     until convergence (linear rate, refer Lemma 2.3).
15:     Set $\widehat{u}_j = v_j^{(t+1)}$, $\lambda_j = T(v_j^{(t+1)}, v_j^{(t+1)}, v_j^{(t+1)})$
    return Estimated top $l$ out of all the top $r$ eigenpairs $(\widehat{u}_j, \lambda_j)_{j\in[l]}$, and low rank estimate $\widehat{L}_l = \sum_{i\in[l]} \lambda_i \widehat{u}_j \otimes \widehat{u}_j \otimes \widehat{u}_j$.
```

Computational complexity: In RTD, at the l^{th} stage, the l-eigenpairs are computed using Algorithm 2, whose complexity is $O(n^3 l N_1 N_2)$. The hard thresholding operation $\mathcal{H}_\zeta(T - L^{(t+1)})$ requires $O(n^3)$ time. We have $O\left(\log\left(\frac{n\|T\|}{\delta}\right)\right)$ iterations for each stage of the RTD algorithm and there are r stages. By Theorem 2.2, it suffices to have $N_1 = \widetilde{O}\left(n^{1+c}\right)$ and $N_2 = \widetilde{O}\left(1\right)$, and where $\widetilde{O}(\cdot)$ represents $O(\cdot)$ up to polylogarithmic factors and c is a small constant. Hence, the overall computational complexity of RTD is $\widetilde{O}\left(n^{4+c} r^2\right)$.

2.4 Theoretical Guarantees

In this section, we provide guarantees for the RTD proposed in the previous section for RTD in (2.1). Even though we consider a symmetric L^* and S^* in (2.1), we can easily extend the results to asymmetric tensors, by embedding them into symmetric tensors, on lines of [84].

In general, it is not possible to have a unique recovery of low-rank and sparse components. For example, if the input tensor T is both sparse and low rank, then there is no unique decomposition (e.g. $T = e_1^{\otimes 3}$). Instead, we assume the following conditions in order to guarantee uniqueness:

(L) L^* is a rank-r orthogonal tensor in (2.1), i.e., $\langle u_i, u_j \rangle = \delta_{i,j}$, where $\delta_{i,j} = 1$ iff $i = j$ and 0 o.w. L^* is μ-incoherent, i.e., $\|u_i\|_\infty \leq \frac{\mu}{n^{1/2}}$ and $\sigma_i^* > 0$, $\forall 1 \leq i \leq r$.

The above conditions of having an incoherent low rank tensor L^* are similar to the conditions for robust matrix PCA. For tensors, the assumption of an orthogonal decomposition is limiting, since there exist tensors whose CP decomposition is non-orthogonal [8]. We later discuss how our analysis can be extended to non-orthogonal tensors. We now list the conditions for sparse tensor S^*.

The tensor S^* is block sparse, where each block has at most d non-zero entries along any fibre and the number of blocks is B. Let Ψ be the support tensor that has the same sparsity pattern as S^*, but with unit entries, i.e. $\Psi_{i,j,k} = 1$ iff. $S^*_{i,j,k} \neq 0$ for all $i, j, k \in [n]$. We now make assumptions on sparsity pattern of Ψ. Let η be the maximum fraction of overlap between any two block fibres ψ_i and ψ_j. In other words, $\max_{i \neq j} \langle \psi_i, \psi_j \rangle < \eta d$. **(S)** Let d be the sparsity level along any fibre of a block and let B be the number of blocks. We require

$$\begin{aligned} &\Psi = \sum_{i=1}^{B} \psi_i \otimes \psi_i \otimes \psi_i, \|\psi_i\|_0 \leq d, \; \psi_i(j) = 0 \text{ or } 1, \forall i \in [B], \; j \in [n], \\ &d = O(n/r\mu^3)^{2/3}, B = O(\min(n^{2/3} r^{1/3}, \; \eta^{-1.5})) \end{aligned} \tag{2.2}$$

We assume a block sparsity model for S^* above. Under this model, the support tensor Ψ which encodes sparsity pattern, has rank B, but not the sparse tensor S^* since the entries are allowed to be arbitrary. We also note that we set d to be $n^{2/3}$ for ease of exposition and show one concrete example where our method significantly outperforms matrix based robust PCA methods.

As discussed in the introduction, it may be advantageous to consider tensor methods for robust decomposition only when sparsity across the different matrix slices are aligned/structured in some manner, and a block sparse model is a natural structure to consider. We later demonstrate the precise nature of superiority of tensor methods under block sparse perturbations.

For the above mentioned sparsity structure, we set $\beta = 4\mu^3 r/n^{3/2}$ in our algorithm. We now establish that our proposed algorithm RTD recovers the low rank and sparse components under the above conditions, thereby establishing convergence to the globally optimal solution.

Theorem 2.1 (Convergence to global optimum for RTD). *Let L^*, S^* satisfy (L) and (S), and $\beta = 4\frac{\mu^3 r}{n^{3/2}}$. The outputs $\widehat{L}$ (and its parameters $\widehat{u}_i$ and $\widehat{\lambda}_i$) and $\widehat{S}$ of Algorithm 1 satisfy w.h.p.:*

$$\|\widehat{u}_i - u_i\|_\infty \le \delta/\mu^2 r n^{1/2} \sigma^*_{\min}, \tag{2.3}$$

$$|\widehat{\lambda}_i - \sigma^*_i| \le \delta, \quad \forall i \in [n], \left\|\widehat{L} - L^*\right\|_F \le \delta, \quad \|\widehat{S} - S^*\|_\infty \le \delta/n^{3/2}, \tag{2.4}$$

$$\textit{and} \quad \operatorname{supp} \widehat{S} \subseteq \operatorname{supp} S^*. \tag{2.5}$$

Comparison with matrix methods: We now compare with the matrix methods for recovering the sparse and low rank tensor components in (2.1). Robust matrix PCA can be performed either on all the matrix slices of the input tensor $M_i := T(I, I, e_i)$, for $i \in [n]$, or on the *flattened* tensor T (see the definition in Section 2.3). We can either use convex relaxation methods [22, 20, 47] or non-convex methods [75] for robust matrix PCA.

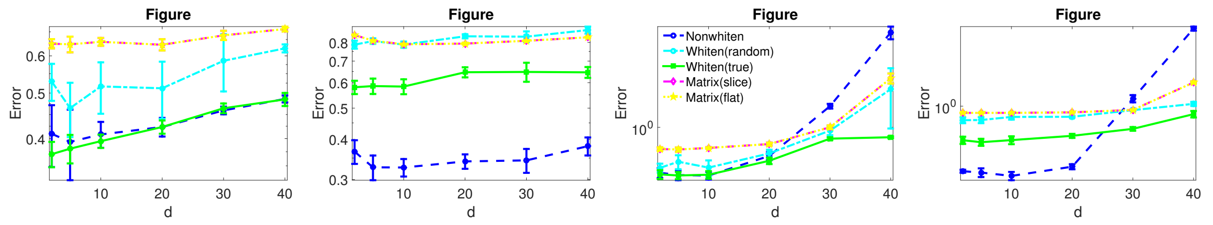
Recall that η measures the fraction of overlapping entries between any two different block fibres, i.e. $\max_{i\neq j}\langle\psi_i,\psi_j\rangle < \eta d$, where ψ_i are the fibres of the block components of tensor Ψ in (2.2) which encodes the sparsity pattern of S^* with 0-1 entries. A short proof is given in Appendix A.2.1.

Corollary 1 (Superiority of tensor methods)**.** *The proposed tensor method* RTD *can handle perturbations S^* at a higher sparsity level compared to performing matrix robust PCA on either matrix slices or the flattened tensor using guaranteed methods in [47, 75] when the (normalized) overlap between different blocks satisfies $\eta = O(r/n)^{2/9}$.*

Simplifications under random block sparsity: We now obtain transparent results for a random block sparsity model, where the components ψ_i in (2.2) for the support tensor Ψ are drawn uniformly among all d-sparse vectors. In this case, the incoherence η simplifies as $\eta \overset{w.h.p}{=} O(\frac{d}{n})$ when $d > \sqrt{n}$ and $\eta \overset{w.h.p}{=} O\ (1/\sqrt{n})$, o.w. Thus, the condition on B in (2.2) simplifies as $B = O(\min(n^{2/3}r^{1/3}, (n/d)^{1.5}))$ when $d > \sqrt{n}$ and $B = O(\min(n^{2/3}r^{1/3}, n^{0.75}))$ o.w. Recall that as before, we require sparsity level of a fibre in any block as $d = O(n/r\mu^3)^{2/3}$. For simplicity, we assume that $\mu = O(1)$ for the remaining section.

We now explicitly compute the sparsity level of S^* allowed by our method and compare it to the level allowed by matrix based robust PCA.

Corollary 2 (Superiority of tensor methods under random sparsity)**.** *Let D_{RTD} be the number of non-zeros in S^* ($\|S^*\|_0$) allowed by* RTD *under the analysis of Theorem 2.1 and let D_{matrix} be $\|So\|_0$ allowed using the standard matrix robust PCA analysis. Also, let S^* be*



(a) (b) (c) (d)

Figure 2.1: (a) Error comparison of different methods with deterministic sparsity, rank 5, varying d. (b) Error comparison of different methods with deterministic sparsity, rank 25, varying d. (c) Error comparison of different methods with block sparsity, rank 5, varying d. (d) Error comparison of different methods with block sparsity, rank 25, varying d. Error $= \|L^* - L\|_F / \|L^*\|_F$. The curve labeled 'T-RPCA-W(slice)' refers to considering recovered low rank part from a random slice of the tensor T by using matrix non-convex RPCA method as the whiten matrix, 'T-RPCA-W(true)' is using true second order moment in whitening, 'M-RPCA(slice)' treats each slice of the input tensor as a non-convex matrix-RPCA(M-RPCA) problem, 'M-RPCA(flat)' reshapes the tensor along one mode and treat the resultant as a matrix RPCA problem. All four sub-figures share same curve descriptions.

sampled from the block sparsity model. Then, the following holds:

$$\frac{D_{\text{RTD}}}{D_{\text{matrix}}} = \begin{cases} \Omega\left(n^{1/6} r^{4/3}\right), & \textit{for } r < n^{0.25}, \qquad (2.6) \\ \Omega\left(n^{5/12} r^{1/3}\right), & \textit{o.w.} \qquad (2.7) \end{cases}$$

Unstructured sparse perturbations S^*: If we do not assume block sparsity in (S), but instead assume unstructured sparsity at level D, i.e. the number of non zeros along any fibre of S^* is at most D, then the matrix methods are superior. In this case, for success of RTD, we require that $D = O\left(\frac{\sqrt{n}}{r\mu^3}\right)$ which is worse than the requirement of matrix methods $D = O(\frac{n}{r\mu^2})$. Our analysis suggests that if there is no structure in sparse patterns, then matrix methods are superior. This is possibly due to the fact that finding a low rank tensor decomposition requires more stringent conditions on the noise level. At the same time, when there is no block structure, the tensor algebraic constraints do not add significantly new information. However, in the experiments, we find that our tensor method RTD is superior to matrix methods even in this case, in terms of both accuracy and running times.

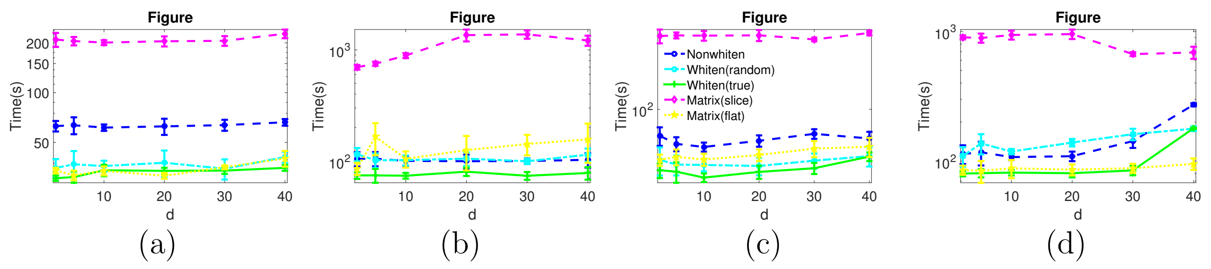


Figure 2.2: (a) Running time comparison of different methods with deterministic sparsity, rank 5, varying d. (b) Running time comparison of different methods with deterministic sparsity, rank 25, varying d. (c) Running time comparison of different methods with block sparsity, rank 5, varying d. (d) Running time comparison of different methods with block sparsity, rank 25, varying d. Curve descriptions are same as in Figure 1.

Analysis of Procedure 2 Our proof of Theorem 2.1 depends critically on an assumption that Procedure 2 indeed obtains the top-r eigen-pairs. We now concretely prove this claim. Let $\widetilde{L}$ be a symmetric tensor which is a perturbed version of an orthogonal tensor L^*, $\widetilde{L} = L^* + E \in \mathbb{R}^{n\times n\times n}$, $\quad L^* = \sum_{i\in[r]} \sigma_i^* u_i^{\otimes 3}$, where $\sigma_1^* \geq \sigma_2^* \dots \sigma_r^* > 0$ and $\{u_1, u_2, \dots, u_r\}$ form an orthonormal basis.

Recall that N_1 is the number of initializations to seed the power method, N_2 is the number of power iterations, and δ is the convergence criterion. For any $\xi \in (0,1)$, and $l \leq r$, assume the following

$$\|E\| \leq O(\sigma_l^*/\sqrt{n}),\ N_1 = O(n^{1+c}\log(1/\xi)), N_2 \geq \Omega(\log(k) + \log(\sigma_{\max}^*/\|E\|)) \tag{2.8}$$

where c is a small constant. We now state the main result for recovery of components of L^* when Procedure 2 is applied to $\widetilde{L}$.

Theorem 2.2 (Gradient Ascent method)**.** *Let $\widetilde{L} = L^* + E$ be as defined above with $\|E\| \leq O(\sigma_r^*/\sqrt{n})$. Then, applying Procedure 2 with deflations on $\widetilde{L}$ with target rank $l \leq r$, yields l eigen-pairs of $\widetilde{L}$, given by $(\lambda_1, \widehat{u}_1), (\lambda_2, \widehat{u}_2), \dots, (\lambda_l, \widehat{u}_l)$, up to arbitrary small error $\delta > 0$ and with probability at least $1-\xi$. Moreover, there exists a permutation π on $[l]$ such that: $\forall j \in [l]$, $|\sigma_{\pi(j)}^* - \lambda_j| \leq O\left(\|E\| + \delta\right), \|u_{\pi(j)} - \widehat{u}_j\| \leq O((\|E\|/\sigma_{\pi(j)}^*) + \delta)$.*

While [8, Thm. 5.1] considers power method, here we consider the power method followed by a gradient ascent procedure. With both methods, we obtain outputs $(\lambda_i, \widehat{u}_i)$ which are "close" to the original eigen-pairs of (σ_i^*, u_i) of L^*. However, the crucial difference is that Procedure 2 outputs $(\lambda_i, \widehat{u}_i)$ correspond to specific eigen-pairs of input tensor $\widetilde{L}$, while the outputs of the usual power method have no such property and only guarantees accuracy upto $O(\|E\|_2)$ error. We critically require the eigen property of the outputs in order to guarantee contraction of error in RTD between alternating steps of low rank decomposition and thresholding.

The analysis of Procedure 2 has two phases. In the first phase, we prove that with N_1 initializations and N_2 power iterations, we get close to true eigenpairs of L^*, i.e. (σ_i^*, u_i) for $i \in [l]$. After this, in the second phase, we prove convergence to eigenpairs of $\widetilde{L}$.

The proof for the first phase is on lines of proof in [8], but with improved requirement on error tensor E in (2.8). This is due to the use of SVD initializations rather than random initializations to seed the power method, and its analysis is given in [9].

Proof of the second phase follows using two observations: a) Procedure 2 is just a simple gradient ascent of the following program: $f(v) = \widetilde{L}(v, v, v) - \frac{3}{4}\lambda\|v\|_2^4$, b) with-in a small distance to the eigenvectors of $\widetilde{L}$, $f(v)$ is strongly concave and as well as strongly smooth with appropriate parameters. See below lemma for a detailed proof of the above claim. Hence, using our initialization guarantee from the phase-one, Procedure 2 converges to a δ approximation to eigen-pair of $\widetilde{L}$ in time $O(\log(1/\delta))$ and hence, Theorem 2.2 holds.

Lemma 2.3. *Let $f(v) = \widetilde{L}(v, v, v) - \frac{3}{4}\lambda\|v\|_2^4$. Then, f is $\sigma_i^*(1 - \frac{300\sigma_r^*}{\sqrt{n}})$-strongly concave and $\sigma_i^*(1 + \frac{300\sigma_r^*}{\sqrt{n}})$ strongly smooth at all points (v, λ) s.t. $\|v - u_i\| \leq \frac{10}{\sqrt{n}}$ and $|\lambda - \sigma_i^*| \leq \frac{10\sigma_r^*}{\sqrt{n}}$, for some $1 \leq i \leq r$. Procedure 2 converges to an eigenvector of $\widetilde{L}$ at a linear rate.*

Proof. Consider the gradient and Hessian of f w.r.t. v:

$$\nabla f = 3\widetilde{L}(I, v, v) - 3\lambda\|v\|^2 v, \tag{2.9}$$

$$H = 6\widetilde{L}(I, I, v) - 6\lambda v v^\top - 3\lambda\|v\|^2 I. \tag{2.10}$$

We first note that the stationary points of f indeed correspond to eigenvectors of $\widetilde{L}$ with eigenvalues $\lambda\|v\|^2$. Moreover, when $|\lambda - \sigma_i^*| \leq \frac{10\sigma_r^*}{\sqrt{n}}$ and $\|v - u_i\| \leq \frac{10}{\sqrt{n}}$, we have:

$$\|H - (-3\sigma_i^* I)\|_2 \leq 30\frac{\sigma_r^*}{\sqrt{n}} + 180\frac{\sigma_r^*}{\sqrt{n}}.$$

Recall that $\widetilde{L} = L^* + E$, where L^* is an orthogonal tensor and $\|E\|_2 \leq \sigma_r^*/\sqrt{n}$. Hence, there exists one eigenvector in each of the above mentioned set, i.e., set of (v, λ) s.t. $|\lambda - \sigma_i^*| \leq \frac{10\sigma_r^*}{\sqrt{n}}$ and $\|v - u_i\| \leq \frac{10}{\sqrt{n}}$. Hence, the standard gradient ascent procedure on f would lead to convergence to an eigenvector of $\widetilde{L}$. ■

Extending to non-orthogonal low rank tensors: In (L), we assume that the low rank tensor L^* in (2.1) is orthogonal. We can also extend to non-orthogonal tensors L^*, whose components u_i are linearly independent. Here, there exists an invertible transformation W known as *whitening* that orthogonalizes the tensors [8]. We can incorporate whitening in Procedure 2 to find low rank tensor decomposition, within the iterations of RTD. The performance of RTD will then depend on various norms of the whitening matrix W and the sparsity level that can be handled is degraded, depending on the extent of non-orthogonality. We leave the analysis for future work.

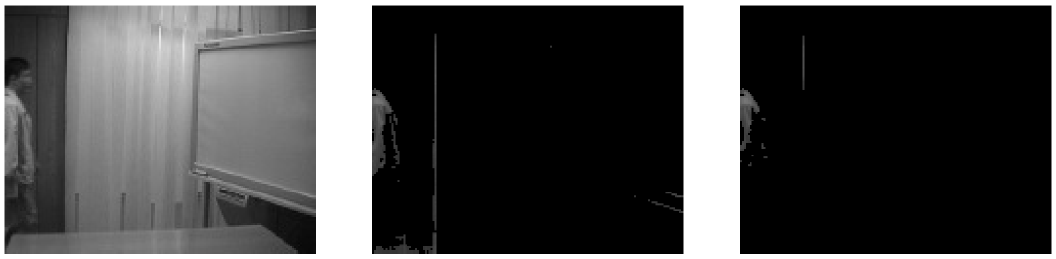

(a) (b) (c)

Figure 2.3: Foreground filtering or activity detection in the *Curtain* video dataset. (a): Original image frame. (b): Foreground filtered (sparse part estimated) using tensor method; time taken is $5.1s$. (c): Foreground filtered (sparse part estimated) using matrix method; time taken is $5.7s$.

2.5 Experiments

We now present an empirical study of our method. The goal of this study is three-fold: a) establish that our method indeed recovers the low-rank and sparse parts correctly, without significant parameter tuning, b) demonstrate that whitening during low rank decomposition gives computational advantages, c) show that our tensor methods are superior to matrix based RPCA methods in practice.

Our pseudo-code (Algorithm 1) prescribes the threshold ζ in Step 5, which depends on the knowledge of the singular values of the low rank component L^*. Instead, in the experiments, we set the threshold at the $(t+1)$ step of l^{th} stage as $\zeta = \mu\sigma_{l+1}(T - S^{(t)})/n^{3/2}$. We found that the above thresholding, in the tensor case as well, provides exact recovery while speeding up the computation significantly.

2.5.1 Synthetic Dataset

The low-rank part $L^* = \sum_{i\in[k]} \lambda_i u_i^{\otimes 3}$ is generated from a factor matrix $U \in \mathbb{R}^{n\times k}$ whose entries are i.i.d. samples from $\mathcal{N}(0,1)$. For deterministic sparsity setting, $\mathrm{supp}(S^*)$ is generated by setting each entry of $[n] \times [n] \times [n]$ to be non-zeros with probability d/n and each non-zero value S^*_{ijk} is drawn i.i.d. from the uniform distribution over $[r/(2n^{3/2}), r/n^{3/2}]$. For block sparsity setting, we randomly select B numbers of $[n] \times [1]$ vectors $v_i, i = 1...B$ in which each entry is chosen to be non-zero with probability d/n. The value of non-zero entry is assigned similar to the one in deterministic sparsity case. Each of this vector will form a subtensor($v_i^{\otimes 3}$) and those subtensors form the whole S. For increasing incoherence of L^*, we randomly zero-out rows of U and then re-normalize them. For the CP-decomposition, we use the alternating least squares (ALS) method available in the tensor toolbox [13]. Note that although we proposed and analyzed the gradient ascent method as in Procedure 2 for performing tensor decomposition for obtaining spectral convergence guarantees, we use the ALS procedure in practice since we found that empirically, ALS performs quite well and is convenient to use. For whitening, we use two different whitening matrices: a) the true second order moment from the true low-rank part, b) the recovered low rank part from a random slice of the tensor T by using matrix non-convex RPCA method. We compare our RTD with matrix non-convex RPCA in two ways: a) treat each slice of the input tensor as a matrix RPCA problem, b) reshape the tensor along one mode and treat the resultant as a matrix RPCA problem.

We vary sparsity of S^* and rank of L^* for RTD with a fixed tensor size. We investigate performance of our method, both with and without whitening, and compare with the state-of-the-art matrix non-convex RPCA algorithm. Our results for synthetic datasets is averaged over 5 runs. In Figure 2.1, we report relative error ($\|L^* - L\|_F / \|L^*\|_F$) by each of the methods allowing maximum number of iterations up to 100. Comparing (a) and (b) in Figure 2.1, we can see that with block sparsity, RTD is better than matrix based non-convex RPCA method when d is less than 20. If we use whitening, the advantage of RTD is more significant. But when rank becomes higher, the whitening method is not helpful. In Figure 2.2, we illustrate the computational time of each methods. Here we can see that whitening simplifies the problem and give us computational advantage. Besides, the running time for the one using tensor method is similar to the one using matrix method when we reshape the tensor as one matrix. We note that doing matrix slices increases the running time.

2.5.2 Real-world Dataset

To demonstrate the advantage of our method, we apply our method to the so-called real-world problem of *activity detection* or *foreground filtering* [64]. The goal of this task is to detect activities from a video coverage, which is a set of image frames that form a tensor. The people or objects moving in a video are said be engaging in some activities. In our robust decomposition framework, these moving people or objects correspond sparse (foreground) perturbations which we wish to filter out. The static ambient background is of lesser interest since nothing changes and therefore is not interesting.

We selected one of datasets, namely the *Curtain* video dataset wherein a person walks in and out of the room between the frame numbers 23731 and 23963. We compare our tensor method with the state-of-the-art matrix method in [75] on the set of 233 frames where the activity happens. In our tensor method, we preserve the multi-modal nature of videos and consider the set of image frames without vectorizing them. For the matrix method, we follow the setup of [75] by reshaping each image frame into a vector and stacking them together. We set the convergence criterion to 10^{-3} and run both the methods. Our tensor method yields a 10% speedup and obtains a noticeably better visual recovery for the same convergence accuracy as shown in Figure 2.3, ie, the person entering the room is captured in entirety and more detail by the tensor method as compared to the matrix method.

2.6 Conclusion

We proposed a non-convex alternating method for decomposing a tensor into low rank and sparse parts. We established convergence to the globally optimal solution under natural conditions such as incoherence of the low rank part and bounded sparsity levels for the sparse part. We prove that our proposed tensor method can handle perturbations at a much higher sparsity level compared to robust matrix methods. Our simulations show superior performance of our tensor method, both in terms of accuracy and computational time. Some future directions are analyzing: (1) our method with whitening (2) the setting where grossly corrupted samples arrive in streaming manner.

Chapter 3

Higher-order Count Sketch

Modern machine learning involves processing of large-scale datasets. Dimensionality-reduction methods attempt to compress the input data while preserving relevant information. Sketching is a popular class of such techniques which aims to reduce memory and computational requirements by using simple randomized hash functions that map the input data to a reduced-sized output space. Count Sketch (CS) [24] is a simple sketch that has been applied in many settings such as estimation of internet packet streams [30] and tracking most frequent items in a database [29]. It uses a simple data-independent random hash function and random signs to combine the input data elements. Despite its simplicity, it enjoys many desirable properties such as unbiased estimation and the ability to approximately perform certain operations directly on the low-dimensional sketched space, e.g., vector inner products and outer products. However, CS is memory inefficient when the data is large. The bottleneck is that it needs to generate a hash table as large as the data size.

Another drawback of CS is that it assumes vector-valued data and does not exploit further structure if data is multi-dimensional. But many modern machine learning and data mining applications involve manipulating large-scale multi-dimensional data. For instance, data can be multi-modal or multi-relational (e.g., a combination of image and text), and intermediate computations can involve higher-order tensor operations (e.g., layers in a tensorized neural network). Memory, bandwidth, and computational requirements are usually bottlenecks when these operations are done at scale. Efficient dimensionality reduction schemes that exploit tensor structures can significantly alleviate this issue if they can find a compact representation while preserving accuracy.

3.1 Summary of Results

We extend count sketch to Higher-order Count Sketch (HCS), which is the first sketch to fully exploit the multi-dimensional nature of higher-order tensors. It reshapes the input (vector) data to a higher-order tensor of a fixed order. It utilizes multiple randomized hash functions: one for each mode of the tensor. The mapping in HCS is obtained by the tensor product of these hash functions. We show a memory saving in storing the hash map: if the input data size is $O(d)$ and HCS uses l-th order tensor for sketching, we reduce the hash memory requirements from $O(d)$ to $O(l\sqrt[l]{d})$, compared to the count sketch, under certain conditions.

The conditions for obtaining the best-case memory savings from HCS are related to the concentration of input entries with large magnitudes and require these large entries to be sufficiently spread out. Intuitively, this is because the hash indices in HCS are correlated and we cannot have all the input to be clustered together. If we are allowed multiple passes over the input data , a simple (in-place) reshuffle to spread out the large entries will fix this issue, and thus allows us to obtain maximum memory savings in storing hash functions.

When the input data has further structure as a higher-order tensor, HCS is able to exploit it. HCS allows for efficient (approximate) computations of tensor operations such as tensor products and tensor contractions by directly applying these operations on the sketched components. We obtain exponential saving with respect to the order of the tensor in the memory requirements for tensor product and contraction when compared to sketching using count sketch. We also show $O(r^{N-1})$ times improvement in computation and memory efficiency for computing a Nth-order rank-r Tucker tensor when compared to applying CS to each rank-1 component. The computation and memory improvement over CS of these operations are shown in Table 3.1.

We compare HCS and CS for tensor product and tensor contraction compression using synthetic data. HCS outperforms CS in terms of computation efficiency and memory usage: it uses 200× less compression time and 40× less memory while keeping the same recovery error, compared to CS. Besides, we apply HCS for approximating tensor operations in tensorized neural networks. These networks replace fully connected layers with multi-linear tensor algebraic operations. Applying HCS to tensor operations results in further compression while preserving accuracy. We obtain 90% test accuracy on CIFAR10 dataset with 80% memory savings on the last fully connected layer, compared to the baseline ResNet18.

3.2 Related Work

Singular value decomposition (SVD) is perhaps the most popular dimensionality reduction technique [32]. However, when data is not inherently low rank or has other constraints such as sparsity and non-negativity, SVD is not suitable. Other matrix decomposition techniques try to impose more structure on matrices [69, 19, 83].

In contrast to matrix techniques which make stringent assumptions on underlying structure, sketching is designed for compressing vector-valued data with almost no assumptions [15, 5, 103]. Count Sketch (CS) [24] was proposed to estimate the frequency of each element in a stream. Pagh [77] propose a fast algorithm to compute CS of an outer product of two vectors using FFT properties. They prove that the CS of the outer product is equal to the convolutions between the CS of each input. This allows for vector operations such as inner product and outer product to be directly computed in the sketch space. Since then, many variants of count sketch have been proposed that preserve different properties of underlying data. Min-hash [17] is a technique for estimating how similar two sets of data are. An extension of that is one-bit CP-hash [28] which generates concurrent hash table for multi-core processors. To make use of parallel computing resources, 2-of-3 cuckoo hashing [6] is proposed based on cuckoo hashing [78].

Sketching can also be applied to multi-dimensional data. Tensor sketch [82] is proposed to approximate non-linear kernels. It has been applied to approximately compute tensor CP decomposition [102, 107] and Tucker decomposition [70]. Gao *et al* [35] introduce compact bilinear pooling to estimate joint features from different sources. In Visual Question Answering task, people use compact bilinear pooling to compute joint features from language and image [33]. However, all these sketching techniques are sketching tensors into a vector, which destroys their multi-dimensional structure. This does not make it possible to do tensor operations efficiently in the sketched space.

In addition to sketching, efficient multi-dimensional data operation primitives can boost the computation performance. A Low-overhead interface is proposed for multiple small matrix multiplications on NVIDIA GPUs [50]. Ma *et al* [68, 87] optimize tensor matrix contraction on GPUs by avoiding data transformation. High-Performance Tensor Transposition [91] is one of the open-source library that performs efficient tensor contractions. In future, we can leverage these advances to further speed up tensor sketching operations.

Important tensor applications: We focus on tensor sketching because data is inherently multi-dimensional in many settings. In probabilistic model analysis, tensor decomposition is the crux of model estimation via the method of moments. A variety of models such as topic models, hidden Markov models, Gaussian mixtures etc., can be efficiently solved via the tensor decomposition techniques under certain mild assumptions [7]. Papalexakis *et al* [79] analyze spatio-temporal basketball data via tensor decomposition. Tensor methods are also relevant in deep learning. Yu *et al* [109] learn the nonlinear dynamics in recurrent neural networks directly using higher-order state transition functions through tensor train decomposition. Kossaifi *et al* [58] propose tensor contraction and regression layers in deep convolutional neural networks.

3.3 Preliminaries

Count Sketch Count Sketch(CS) [24] was first proposed to estimate most frequent data value in a streaming data.

Definition 3.1 (Count Sketch)**.** *Given two 2-wise independent random hash functions h:*$[n] \to [c]$ *and s:*$[n] \to \{\pm 1\}$*. Count Sketch of a vector* $u \in \mathbb{R}^n$ *is denoted by:*

$$CS(u) = \{CS(u)_1, \cdots, CS(u)_c\} \in \mathbb{R}^c$$

where $CS(u)_j := \sum_{h(i)=j} s(i)u_i$*.*

In matrix format, we can write it as $CS(u) = H(s \circ u)$, where $H \in \mathbb{R}^{c \times n}$, $H(j,i) = 1$, if $h(i) = j$, for $\forall i \in [n]$, otherwise $H(j,i) = 0$, $\circ$ is the sign for element-wise product. The estimation can be made more robust by taking b independent sketches of the input and calculating the median of the b estimators. Pagh [24] prove that the CS is an unbiased estimator with variance bounded by the 2-norm of the input. See Appendix B.2.1 for detailed proof. [77] use CS and propose a fast algorithm to compute count sketch of an outer product of two vectors.

$$\mathrm{CS}(u \otimes v) = \mathrm{CS}(u) * \mathrm{CS}(v) \tag{3.1}$$

The convolution operation (represented using $*$) can be transferred to element-wise product using FFT properties. Thus, the computation complexity reduces from $O(n^2)$ to $O(n + c\log c)$, if the vectors are of size n and the sketching size is c.

Some notations we use in the following sections are: $\widehat{u}$: decompression of u, $[n]$: set of $\{1, 2, \ldots, n\}$.

3.4 Higher-order Count Sketch on Vector-valued Data

Higher-order Count Sketch(HCS) Given a vector $u \in \mathbb{R}^d$, random hash functions h_k:$[n_k] \to [m_k]$, $k \in [l]$, random sign functions s_k:$[n_k] \to \{\pm 1\}$, $k \in [l]$, and $d = \prod_{k=1}^{l} n_k$, we propose HCS as:

$$\mathrm{HCS}(u)_{t_1,\cdots,t_l} := \sum_{h_1(i_1)=t_1,\ldots,h_l(i_l)=t_l} s_1(i_1)\cdots s_l(i_l) u_j \tag{3.2}$$

where $j = \sum_{k=2}^{l} i_k \prod_{p=1}^{k-1} n_p + i_1$. This is the index mapping between the vector with its reshaping result—a lth-order tensor with dimensions n_k on each mode, for $k \in [l]$.

Using tensor operations, we can denote HCS as:

$$\mathrm{HCS}(u) = (S \circ reshape(u))_{\times 1} H_1 \ldots_{\times l} H_l \tag{3.3}$$

Here, $S = s_1 \otimes \cdots \otimes s_l \in \mathbb{R}^{n_1 \times \cdots \times n_l}$, $H_k \in \mathbb{R}^{n_k \times m_k}$, $H_k(a,b) = 1$, if $h_k(a) = b$, otherwise $H_i(a,b) = 0$, for $\forall a \in [n_k], b \in [m_k], k \in [l]$. The $reshape(u)$ can be done in-place. We assume u is a vectorized layout of a lth-order tensor. To recover the original tensor, we have

$$\widehat{u}_j = s_1(i_1)\cdots s_l(i_l)\mathrm{HCS}(u)_{h_1(i_1),\cdots,h_l(i_l)} \tag{3.4}$$

Assume $\mathcal{T}_p$ is a pth-order tensor by fixing $l-p$ modes of a lth-order tensor $reshape(u)$ as shown in Figure 1.5:

Theorem 3.1 (HCS recovery analysis). *Given a vector $u \in \mathbb{R}^d$, assume T_p is the maximum frobenium norm of all $\mathcal{T}_p$, Equation 3.4 computes an unbiased estimator for u_{j*} with variance bounded by:*

$$\mathrm{Var}(\widehat{u}_{j*}) = O(\sum_{p=1}^{l}\frac{T_p^2}{m^p}) \tag{3.5}$$

Remarks Compared to CS, HCS requires less space for storing the hash functions. Each mode only requires a $m_k \times n_k$ sized hash matrix with n_k nonzero entries ($n_k = O(\sqrt[l]{d})$). Thus, HCS required $O(l\sqrt[l]{d})$ for hash memory while CS requires $O(d)$. If we choose $l = o(d)$, then $O(d) \gg O(l\sqrt[l]{d})$ and we save memory from using HCS.

The dominant term in Equation 3.5 will be $\|u\|_2^2/m^l$ as long as all large entries are not clustered close to one another. Notice that CS has variance bounded by $\|u\|_2^2/c$. We require $O(m^l) = O(c)$ to guarantee the same recovery, and that will lead to a total output memory $O(c)$ for HCS. However, due to the correlations between hash indices across different modes, HCS needs larger sketching space compared to CS if large entries are clustered. In the worst case, when large magnitude data all locate in one fiber of $reshape(u)$, HCS has variance bounded by $\|u\|_2^2/m$. We require $O(m) = O(c)$ for the same recovery error. HCS output is of size $O(c^l)$ while CS output's size is $O(c)$.

We present a simple way to reshuffle the data in-place. Step1: Sort u in descending order.

Step2: Rearrange sorted array in designed space $n_1 \times \ldots \times n_l$ such that it goes diagonally from top to bottom and then consecutive anti-diagonally from bottom to top. Step3: Rearrange the data according to Step2 (column-wise fiber by fiber). We assume all data is well distributed in the rest analysis.

Another concern in HCS is how to choose the order of the reshaping tensor (parameter l). If the data values are fairly evenly distributed, we should select l as large as possible (but sublinear in d).

3.5 Higher-order Count Sketch on Tensors

In the previous section, we discuss how to sketch a vector-valued data using HCS. In this section, we focus on tensor-valued data. In order to use CS on higher-order tensors, we either view the tensor as a set of vectors and sketch along each fiber of the tensor or we flatten the tensor as a vector and apply CS on it. Hence, CS do not exploit tensors. Moreover, operations between tensors have to be performed on sketched vectors. This process is inefficient. But, with the help of HCS, we can compute various operations such as tensor products and tensor contractions by directly applying operations on the sketched components.

It is straightforward to apply HCS on tensors. Given a tensor $T \in \mathbb{R}^{n_1 \times \cdots \times n_N}$, random hash functions h_k:$[n_k] \to [m_k]$, $k \in [N]$, and random sign functions s_k:$[n_k] \to \{\pm 1\}$, $k \in [N]$, HCS computes: $\text{HCS}(T) = (S \circ T)_{\times 1} H_1 \ldots_{\times N} H_N$. To recover the original tensor, we have: $\widehat{T} = S \circ \text{HCS}(T)_{\times 1} H_1^T, \cdots_{\times N} H_N^T$. S and H_i are defined as same as in Section 3.4.

3.5.1 Tensor Product

Tensor product is known as outer product in vectors case. It computes every bilinear composition from inputs. We denote the operation with $\otimes$. It has been used in a wide range of applications such as bilinear models [95]. Pagh [77] shows that the count sketch of an outer product equals the convolution between the count sketch of each input vector as shown in Equation 3.1. Furthermore, the convolution operation in the time domain can be transferred to the element-wise product in the frequency domain. We extend the outer product between vectors to tensor product.

Lemma 3.2. *Given a pth-order tensor A, a qth-order tensor B, assume $p > q$:*

$$\begin{aligned} \mathrm{HCS}(A \otimes B) &= \mathrm{HCS}(A) * \mathrm{HCS}(B) \\ &= IFFT(FFT(\mathrm{HCS}(A)) \circ FFT(\mathrm{HCS}(B))) \end{aligned} \tag{3.6}$$

FFT and $IFFT$ are p-dimensional Fourier transform and inverse Fourier transform if the input is a *pth*-order tensor. The proof is given in Appendix B.2.2.

We use the Kronecker product, which is a generalization of the outer product from vectors to matrices to compare tensor product approximation using HCS and CS.

Assume inputs are $A, B \in \mathbb{R}^{n\times n}$: Given Lemma 3.2, this approximation requires $O(n^2)$ to complete HCS(A), HCS(B) and $O(m^2 \log m)$ to complete 2D Fourier Transform if the HCS sketching parameter is m for each mode. It requires $O(m^2)$ memory for final representation and $O(n)$ for hashing parameters along each mode.

Baseline CS operation We flatten A and B as vectors and apply CS on the vector outer product. The computation complexity is $O(n^2 + c \log c)$ and the memory complexity is $O(c + n^2)$. It needs $O(n^2)$ for hashing memory because we have $O(n^2)$ elements in the vectorized matrix.

HCS requires approximately $O(n)$ times less memory comparing to CS for two $n \times n$ matrix Kronecker product. See Table 3.1 for detailed comparisons.

Table 3.1: Computation and memory analysis of various operation estimation (Results select sketch size to maintain the same recovery error for CS and HCS)

Operator	Computation	Memory
$\text{CS}(A \otimes B)$	$O(n^2 + c \log c)$	$O(c + n^2)$
$\text{HCS}(A \otimes B)$	$O(n^2 + c \log c)$	$O(c + n)$
$\text{CS}(AB)$	$O(nr + cr \log c)$	$O(c + n + cr)$
$\text{HCS}(AB)$	$O(nr + cr)$	$O(c + n + \sqrt{c}r)$
$\text{CS}(\textit{Tucker}(T))$	$O(nr^3 + cr^3 \log c)$	$O(c + n + cr^3)$
$\text{HCS}(\textit{Tucker}(T))$	$O(nr + cr^3)$	$O(c + n + \sqrt[3]{c}r)$

3.5.2 Tensor Contraction

Matrix product between $A \in \mathbb{R}^{m \times r}$ and $B \in \mathbb{R}^{r \times n}$ is defined as $C = AB = \sum_{i=1}^{r} A_{:i} \otimes B_{i:}$, $C \in \mathbb{R}^{m \times n}$. *Tensor contraction* (used more often as Einstein summation in physics community) can be seen as an extension of matrix product in higher-dimensions. It is frequently used in massive network computing. It is defined in Section 1.2.

Lemma 3.3. *Given tensors $A \in \mathbb{R}^{\mathcal{P}}$, $B \in \mathbb{R}^{\mathcal{Q}}$, contraction indices $\mathcal{L}$, if hash matrices $H_i = I$, $\forall i \in \mathcal{L}$:*

$$\text{HCS}(A_{\mathcal{P}} B_{\mathcal{Q}}) = \text{HCS}(A)\text{HCS}(B) \tag{3.7}$$

We require the hash matrices for the contraction modes be identity matrices. In other words, we are not compressing along the modes that are being multiplied. The proof is in Appendix B.2.3.

Baseline CS operation To apply CS on tensor contraction, we have to apply CS to each addition term in Equation 1.1. Take matrix product as an example:

$$\mathrm{CS}(AB) = \mathrm{CS}(\sum_{i=1}^{r} A_{:i} \otimes B_{i:}) = \sum_{i=1}^{r} \mathrm{CS}(A_{:i} \otimes B_{i:}) = \sum_{i=1}^{r} \mathrm{CS}(A_{:i}) * \mathrm{CS}(B_{i:}) \tag{3.8}$$

We summarize computation and memory requirements for matrix product in Table 3.1.

3.5.3 Tucker-form Tensor

Tensor decomposition is an extension of matrix decomposition to higher-orders. Tensor decomposition has been applied in many field such as data mining [57] and latent variable models [7]. We define Tucker-form tensor decomposition in Section 1.2.

Lemma 3.4. *Given a Tucker tensor $T = G_{\times 1} U_{\times 2} V_{\times 3} W \in \mathbb{R}^{n\times n\times n}$, where $G \in \mathbb{R}^{r\times r\times r}$: The higher-order CS of a Tucker-form tensor can be accomplished by performing HCS on each factor:*

$$\mathrm{HCS}(T) = G_{\times 1}\mathrm{HCS}(U)_{\times 2}\mathrm{HCS}(V)_{\times 3}\mathrm{HCS}(W) \tag{3.9}$$

Baseline CS operation To apply CS to a Tucker-form tensor, we rewrite the decomposition as a sum of rank-1 tensors:

$$\mathrm{CS}(T) = \sum_{a=1}^{r}\sum_{b=1}^{r}\sum_{c=1}^{r} G_{abc}\mathrm{CS}(U_a \otimes V_b \otimes W_c) \tag{3.10}$$

where U_a, V_b, W_c are a^{th}, b^{th}, c^{th} column of U, V, W respectively.

We show computation and memory analysis in Table 3.1. In addition, a CP-form tensor can be sketched in the same way as we described above when using HCS. For using CS: instead of summing over all G values, we sum over only r number of G values. The analysis can also be easily extended to higher-order tensors.

We summarize the general tensor product and tensor contraction estimation process in Table 3.2.

Table 3.2: General tensor operation estimation (Assume A is a set of indices with length p, B is a set of indices with length q, each index value $O(n)$, assume the size of R is l with each index value $O(r)$, $g = \max(p, q)$)

Tensor Product: $\mathcal{A} \in \mathbb{R}^A$, $\mathcal{B} \in \mathbb{R}^B$		
Operator	Computation	Memory
$\text{CS}(\mathcal{A} \otimes \mathcal{B}) = \text{CS}(vec(\mathcal{A}) \otimes vec(\mathcal{B}))$	$O(n^g + c\log c)$	$O(c + n^g)$
$\text{HCS}(\mathcal{A} \otimes \mathcal{B}) = \text{HCS}(\mathcal{A}) * \text{HCS}(\mathcal{B})$	$O(n^g + c\log c)$	$O(c + gn)$
Tensor Contraction: $\mathcal{A} \in \mathbb{R}^A$, $\mathcal{B} \in \mathbb{R}^B$ with contraction indices R		
Operator	Computation	Memory
$\text{CS}(\mathcal{A}\mathcal{B}) = \sum_R \text{CS}(A_{:R} \otimes B_{R:})$	$O(r^l n^g + cr^l \log c)$	$O(c + cr^l + n^g)$
$\text{HCS}(\mathcal{A}\mathcal{B}) = \text{HCS}(\mathcal{A})\text{HCS}(\mathcal{B})$	$O(r^l n^g + cr^l)$	$O(c + c^{\frac{g}{p+q}} r^l + gn)$

3.6 Experiments

The goals of this section are: evaluate HCS for data compression; demonstrate the advantages of HCS in various tensor operation estimations, compared to CS; present potential application of HCS in deep learning tasks. All synthetic data experiments are run on a MAC with Intel Core i5 processor. Section 3.6.3 is run on a NVIDIA DGX station with Tesla V100 Tensor Core GPU.

3.6.1 HCS for Unevenly-distributed Data

In Section 3.4, we point out that unevenly-distributed data value may affect the performance of HCS. We generate a matrix $A \in \mathbb{R}^{50\times 50}$, where every entry is sampled from a uniform distribution between -1 and 1, except the elements in the second column, which are filled with value 100. We compress this data using HCS and CS. The compression ratio is calculated as $2500/m^2$ where m is the sketching dimension along each two mode for HCS. CS sketching dimension $c = m^2$. We rearrange the data so that values in the matrix are evenly distributed, and we run the HCS and CS again. We compare the relative error$(\frac{\|\widehat{A}-A\|_F}{\|A\|_F})$ in Figure 3.1. HCS performs poorly before rearranging the data. But after the rearrangment, HCS performs as good as CS, which corresponds to our analysis.

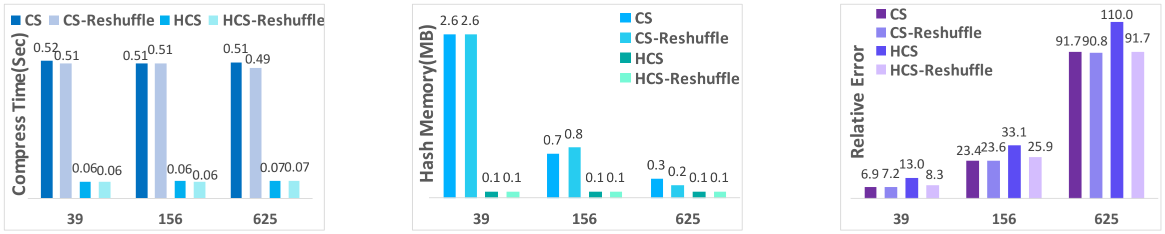


Figure 3.1: Running time, memory and error comparison for unevenly-distributed data (x-axis shows the compression ratio).

3.6.2 Tensor Operations

Kronecker product: We compress Kronecker products using HCS and CS. We compute $A \otimes B$, where $A, B \in \mathbb{R}^{30\times30}$. All inputs are randomly generated from the uniform distribution[-5,5]. The result is obtained by independently running the sketching 20 times and choosing the median. Keeping the same compression ratio, HCS has slightly higher recovery error than CS. But HCS is systematically better in computation speed and memory usage compared to CS in Figure 3.2.

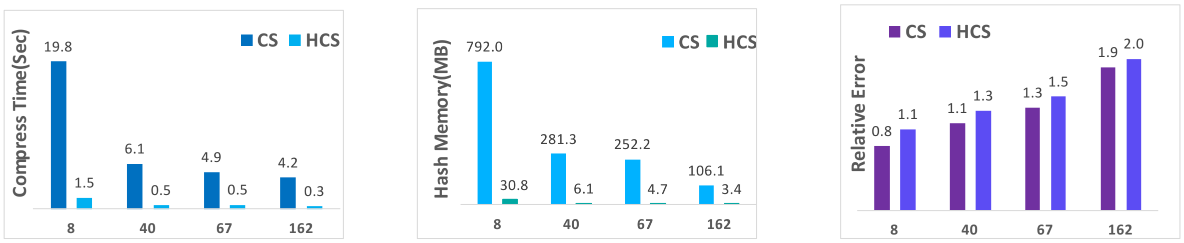


Figure 3.2: Running time, memory and error comparison for Kronecker product.

Tensor contraction: Given $A \in \mathbb{R}^{30\times30\times40}$, $B \in \mathbb{R}^{40\times30\times30}$, we compute $AB \in \mathbb{R}^{30\times30\times30\times30}$: the third mode of A contract with the first mode of B. We compress and decompress the contraction as demonstrated in Section 3.5.2. All entries are sampled independently and uniformly from [0,10]. We repeat the sketching 20 times and use the median as the final estimation. Overall, HCS outperforms CS in time, memory and recovery error aspects as shown in Figure 3.3. When the compression ratio is 8, HCS is 200x faster than CS and uses 40x less memory while keeping almost the same recovery error. HCS is more efficient in real computation because it performs compact contraction in matrix/tensor format, while CS requires computation on each slide of the input tensor.

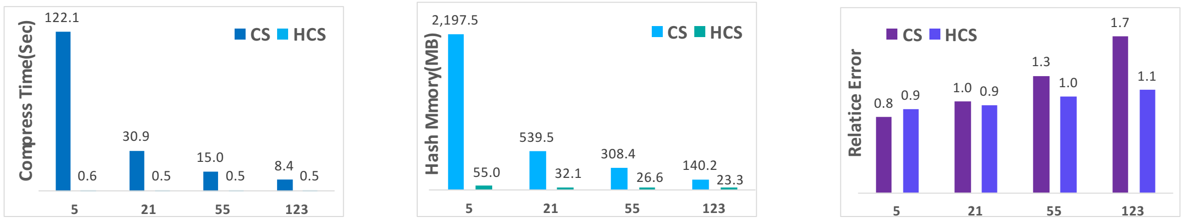


Figure 3.3: Running time, memory and error comparison for tensor contraction.

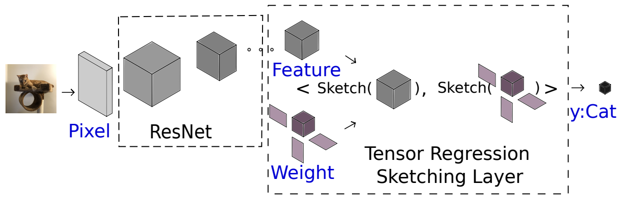


Figure 3.4: Tensor regression layer with sketching.

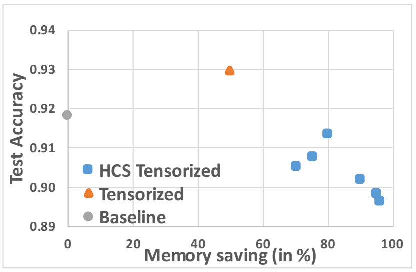


Figure 3.5: Test accuracy on CIFAR 10.

3.6.3 Tensor Regression Network

To demonstrate the versatility of our method, we combine it by integrating it into a tensor regression network for object classification. Tensor regression layer (TRL) [58] is proposed to learn a Tucker-form tensor weight for the high-order activation tensor. We sketch the Tucker tensor weight using Equation 3.9. We use a ResNet18, from which the fully-connected layers are removed, and are replaced by our proposed sketched tensor regression layer. The network structure is illustrated in Figure 3.4. The space saving is calculated as $1 - \frac{P_T}{P_B}$ where P_T and P_B are the number of parameters in the last layer in the tensorized network and the baseline. In Figure 3.5, tensorized network outperforms the baseline(original Resnet18) while using 50% less memory in the last layer. With HCS, we can further reduce 30% more memory requirement while keeping the prediction as good as the baseline.

3.7 Conclusion

In this chapter, we extend count sketch to a new sketching technique, called higer-order count sketching (HCS). HCS gains an exponential saving (with respect to the order of the tensor) in the memory requirements of the hash functions and allows efficient approximation of various tensor operations such as tensor products and tensor contractions. Some interesting future works are how to choose the optimal tensor order for input (vector) data when we have limited information about the data and how to extend other hash algorithms such as simhash [26], minhash [17] and cuckoo hashing [78] to tensors. We are also interested in analyzing the performance differences on sparse and dense tensors using various sketching techniques. Providing HCS implementations within computation platforms such as MKL and CUDA is also part of the future work.

Chapter 4

Multi-modality Learning through Tensor Product

The relative maturity and flexibility of deep learning allow to build upon the success of computer vision [61] and natural language [46, 71] to face new complex and multimodal tasks. Visual Question Answering(VQA) [11] focus on providing a natural language answer given any image and any free-form natural language question. To achieve this goal, information from multiple modalities must be integrated. Image descriptors have structures at multiple spatial scales, while lexical inputs inherently follow a temporal sequence and naturally cluster into semantically different question types. Visual and lexical inputs are first processed using specialized encoding modules and then integrated through differentiable operators. Image features are usually extracted by convolution neural networks [31], while recurrent neural networks [93, 46] are used to extract question features. Additionally, attention mechanism [105, 108, 106] forces the system to *look at* informative regions in both text and vision. Attention weight is calculated from the correlation between language and vision features and then is multiplied to the original feature.

Previous works explore new features to represent vision and language. Pre-trained ResNet [43] and VGG [89] are commonly used in VQA vision feature extraction. The authors in [96] show that post-processing CNN with region-specific image features [10] such as Faster R-CNN can lead to an improvement of VQA performance. Instead of generating language feature from either sentence-level or word-level using LSTM [46] or word embedding, Lu *et al* [66] propose to model the question from word-level, phrase-level, and entire question-level in a hierarchical fashion.

Through extensive experimentation and ablation studies, we notice that the role of “raw” visual features from ResNet and processed region-specific features from Faster R-CNN is complementary and leads to improvement over different subsets of question types. However, we also notice that trivial information in VQA dataset: question/answer type is omitted in training. Generally, each sample in any VQA dataset contains one image file, one natural language question/answer and sometimes answer type. A lot of work use the answer type for result analysis [11] but neglect to use it during learning. TDIUC [53] is a recently released dataset that contains question type for each sample. Compared to answer type, question type has less variety and is easier to interpret when we only have the question.

The focus of this work is the development of an attention mechanism that exploits high-level semantic information on the question type to guide the visual encoding process. This procedure introduces information leakage between modalities before the classical integration phase that improves the performance on VQA task. Specifically, We introduce a novel VQA architecture **Question Type-guided Attention**(QTA) that dynamically gates the contribution of ResNet and Faster R-CNN features based on the question type. Our results with QTA allow us to integrate the information from multiple visual sources and obtain gains across all question types. A general VQA network with our QTA is shown in Figure 1.4.

4.1 Summary of Results

The contributions of this work are:(1) We propose question type-guided attention to balance between bottom-up and top-down visual features, which are respectively extracted from ResNet and Faster R-CNN networks. Our results show that QTA systematically improves the performance by more than 5% across multiple question type categories such as "Activity Recognition", "Utility" and "Counting" on TDIUC dataset. By adding QTA to the state-of-art model MCB, we achieve 3% improvement in overall accuracy. (2) We propose a multi-task extension that is trained to predict question types from the lexical inputs during training time that do not require ground truth labels during inference. We get more than 95% accuracy for the question type prediction while keeping the VQA task accuracy almost same as before. (3) Our analysis reveals some problems in the TDIUC VQA dataset. Though the "Absurd" question is intended to help reduce bias, it contains too many similar questions, specifically, questions regarding color. This will mislead the machine to predict wrong question types. Our QTA model gets 17% improvement on simple accuracy compared to the baseline in [53] when we exclude absurd questions in training.

4.2 Related Work

VQA task is first proposed in [11]. It focuses on providing a natural language answer given any image and any free-form natural language question. Collecting data and solving the task are equally challenging as they require the understanding of the joint relation between image and language without any bias.

Datasets VQA dataset v1 is first released by Antol *et al* [11]. The dataset consists of two subsets: real images and abstract scenes. However, the inherent structure of our world is biased and it results in a biased dataset. In another word, a specific question tends to have the same answer regardless of the image. For example, when people ask about the color of the sky, the answer is most likely blue or black. It is unusual to see the answer be yellow. This is the bottleneck when we give a yellow color sky and ask the machine to answer it. Goyal *et al* [38] release VQA dataset v2. This dataset pairs the same question with similar images that lead to different answers to reduce the sample bias. Zhang *et al* [110] also propose to reduce bias in abstract scenes dataset at question level. By extracting representative word tuples from questions, they can identify and control the balance for each question. Vizwiz [41] is another recently released dataset that uses pictures taken by blind people. Some pictures are of poor quality, and the questions are spoken. These data collection methods help reduce bias in the dataset.

Johnson *et al* [52] introduce Compositional Language and Elementary Visual Reasoning (CLEVR) diagnostic dataset that focuses on reasoning. Strub *et al* [92] propose a two-player guessing game: guess a target in a given image with a sequence of questions and answers. This requires both visual question reasoning and spatial reasoning.

The Task Driven Image Understanding Challenge dataset(TDIUC) [53] contains a total of over 1.6 million questions in 12 different types. It contains images and annotations from MSCOCO [65] and Visual genome [60]. The key difference between TDIUC and the previous VQA v1/v2 dataset is the categorization of questions: Each question belongs to one of the 12 categories. This allows a task-oriented evaluation such as per question-type accuracies. They also include an “Absurd” question category in which questions are irrelevant to the image contents to help balance the dataset.

Feature Selection VQA requires solving several tasks at once involving both visual and textual input: visual perception, question understanding, and reasoning. Usually, features are extracted respectively with convolutional neural networks [31] from the image, and with recurrent neural networks [93, 46] from the text.

Pre-trained ResNet and VGG are commonly used in VQA vision feature extraction. The authors in [96] show that post-processing CNN with region-specific image features [10] can lead to a n improvement of VQA performance. Specifically, they use pre-trained Faster R-CNN model to extract image features for VQA task. They won the VQA challenge 2017.

On the language side, pre-trained word embeddings such as Word2Vec [71] are used for text feature extraction. There is a discussion about the sufficiency of language input for VQA task. Agrawal *et al* [4] have shown that state-of-art VQA models converge to the same answer even if only given half of the question compared to if given the whole sentence. Teney *et al* [96] also questioned about the importance of word ordering in questions.

Generic Methods Information of both modalities are used jointly through means of combination, such as concatenation, product or sum. In [11], authors propose a baseline that combines LSTM embedding of the question and CNN embedding of the image via a point-wise multiplication followed by a multi-layer perceptron classifier.

Pooling Methods Pooling methods are widely used in visual tasks to combine information for various streams into one final feature representation. Common pooling methods such as average pooling and max pooling bring the property of translation invariance and robustness to elastic distortions at the cost of spatial locality. Bilinear pooling can preserve spatial information, which is performed with the outer product between two feature maps. However, this operation entails high output dimension $O(MN)$ for feature maps of dimension M and N. This exponential growth with respect to the number of feature maps renders it too costly to be applied to huge real image datasets. There have been several proposals for new pooling techniques to address this problem:

- Count sketch [25] is applied as a feature hashing operator to avoid dimension expanding in bilinear pooling. Given a vector $a \in \mathcal{R}^n$, random hash function $f \in \mathcal{R}^n$: $[n] \rightarrow [b]$ and binary variable $s \in \mathcal{R}^n$: $[n] \rightarrow \pm 1$, the **count sketch** [25] operator $cs(a, h, s) \in \mathcal{R}^b$

is:

$$cs(a, f, s)[j] = \sum_{f[i]=j} s[i]a[i], \quad j \in 1, \cdots, b \tag{4.1}$$

Gao *et al* [35] use convolution layers from two different neural networks as the local descriptor extractors of the image and combine them using count sketch. "α-pooling" [88] allows the network to learn the pooling strategy: a continuous transition between linear and polynomial pooling. They show that higher α gives larger gain for fine-grained image recognition tasks. However, as α goes up, the computation complexity increases in polynomial order.

- MCB [33] uses count sketch as a pooling method in VQA tasks and obtains the best results on VQA dataset v1 in VQA challenge 2016. They compute count sketch approximation of the visual and textual representation at each spatial location. Given text feature $v \in \mathcal{R}^L$ and image features $I \in \mathcal{R}^{C\times H\times W}$, Fukui *et al* [33] propose **MCB** as:

$$\begin{aligned} & MCB(I[:, h, w] \otimes v)[t_1, h, w] \\ & \qquad = (cs(I[:, h, w], f, s) * cs(v, f, s))[t_1, h, w] \\ & \qquad = IFFT1(FFT1(cs(I[:, h, w], f, s))[t_1, h, w] \circ \quad FFT1(cs(v, f, s))[t_1] \\ & \qquad h \in \{1, \cdots H\}, w \in \{1, \cdots W\}, t_1 \in \{1, \cdots, b\}) \end{aligned} \tag{4.2}$$

$\otimes$ denotes outer product. $\circ$ denotes element-wise product. $*$ denotes convolution operator. This procedure preserves spatial information in image feature. While this procedure preserves spatial information locally, outer-products are taken independently for each fiber of the activation tensor, and therefore do not include spatial context.

Attention Focusing on the objects in the image that are related to the question is the key to understand the correlation between the image and the question. Attention mechanism is used to address this problem. There are soft attention and hard attention [106] based on whether the attention term/loss function is differentiable or not. Yang *et al* [108] and Xu *et al* [105] proposed word guided spatial attention specifically for VQA task. Attention weight at each spatial location is calculated by the correlation between the embedded question word feature and the embedded visual features. The attended pixels will be at the maximum correlation.

4.3 Question Type Guided Visual Attention

Question type is very important in predicting the answer regardless if we have the corresponding image or not. For example, questions starting with "how many" will mostly lead to numerical answers. Agrawal *et al* [4] have shown that state-of-art VQA models converge to the same answer even if only given half of the question compared to if given the whole sentence. Besides that, inspired by [96], we are curious about combining bottom-up and top-down visual features in VQA task. To get a deep understanding of visual feature preference for different questions, we try to find an attention mechanism between these two. Since question type is representing the question, we propose Question Type-guided Attention(QTA).

Given several independent image features $F_1, F_2, \cdots F_k$, such as features from ResNet, VGG or Faster R-CNN, we concatenate them as one image feature: $F = [F_1, F_2, \cdots F_k] \in \mathbb{R}^M$. Assume there are N different question types, QTA is defined as $F \circ WQ$, where $Q \in R^N$ is the one-hot encoding of the question type, and $W \in R^{M \times N}$ is the hidden weight. We can learn the weight by back propagation through the network. In other words, we learn a question type embedding and use it as attention weight.

QTA can be used in both generic and complex pooling models. In Figure 4.1, we show a simple concatenation model with question type as input. We describe it in detail in Section 4.4.

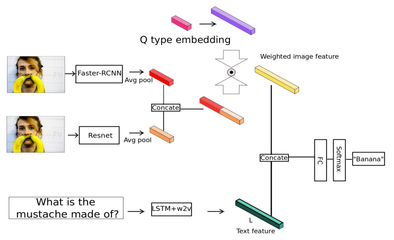


Figure 4.1: Concatenation model with QTA structure for VQA task(CATL-QTA^W in Section 4.4).

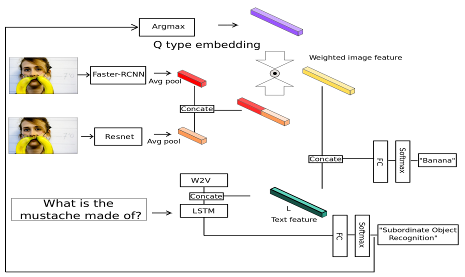


Figure 4.2: Concatenation model with QTA structure for multi-task(CATL-QTA-M-W2V in Section 4.4).

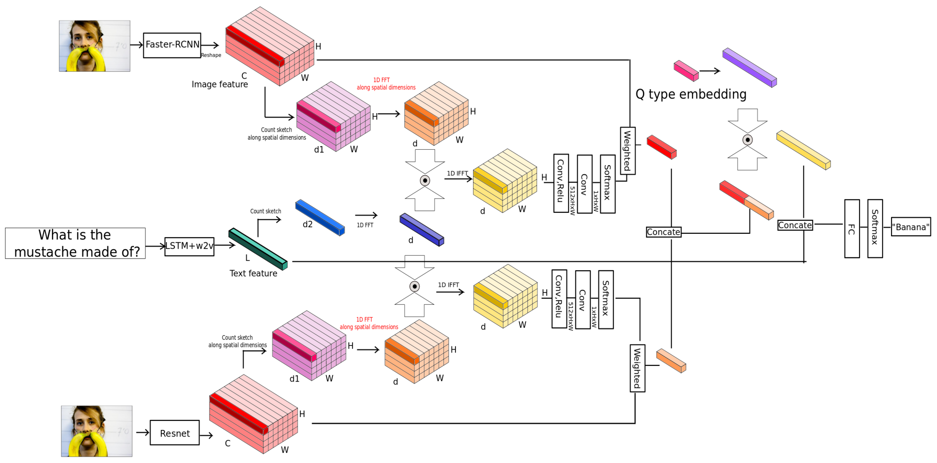


Figure 4.3: MCB model with QTA structure(MCB3-A in Section 4.4).

To fully exploit image features in different channels and preserve spatial information, we also propose MCB with question type-guided image attention in Figure 4.3.

One obvious limitation of QTA is that it requires question type label. In the real world scenario, the question type for each question may not be available. In this case, it is still possible to predict the question type from the text, and use it as input to the QTA network. Thus, we propose a multi-task model that focuses on VQA task along with the prediction of the question type in Figure 4.2. This model operates in the setting where true question type is available only at training time. In Section 4.5, we also show through experiment that it is a relatively easy task to predict the question type from question text, and thus making our method generalizable to those VQA settings that lack question type.

4.4 Experiments

In this section, we describe the dataset in Section 4.4.1, evaluation metrics in Section 4.4.2, model features in Section 4.4.3, and model structures are explained in Section 4.4.4.

4.4.1 Dataset

Our experiments are conducted on the Task Driven Image Understanding Challenge dataset (TDIUC) [53], which contains over 1.6 million questions in 12 different types. This dataset includes VQA v1 and Visual Genome, with a total of 122429 training images and 57565 test images. The annotation sources are MSCOCO (VQA v1), Visual genome annotations, and manual annotations. TDIUC introduces absurd questions that force an algorithm to determine if a question is valid for a given image. There are 1115299 total training questions and 538543 total test questions. The total number of samples is 3 times larger than that in VQA v1 dataset [11].

4.4.2 Evaluation Metrics

There are total 12 different question types in TDIUC dataset [53] as we mentioned in Section 4.2. We calculate the simple accuracy for each type separately and also report the arithmetic and harmonic means across all per question-type(MPT) accuracies.

4.4.3 Feature Representation

Image feature We use the output of ”pool5” of a 152-layer ResNet as an image feature baseline. The output dimension is $2048 \times 14 \times 14$. Faster R-CNN [85] focuses on object detection and classification. Teney *et al* [96] use it to extract object-oriented features for VQA dataset and show better performance compared to the ones using ResNet feature. We fix the number of detected objects to be 36 and extract the image features based on their pre-trained Faster R-CNN model. As a result, the extracted image feature is a 36×2048 matrix. To fit in MCB model, which requires spatial representation, we reshape it into a $6 \times 6 \times 2048$ tensor.

Text feature We use common word embedding library: 300-dim Word2Vec [71] as pre-trained text feature: we sum over the word embeddings for all words in the sentence. A two-layer LSTM is used as an end-to-end text feature extractor. We also use the encoder of google neural machine translation(NMT) system [104] as a pre-trained text feature and compare it with Word2Vec. The pre-trained NMT model is trained on UN parallel corpus 1.0 in MXnet [27]. Its BLEU score is 34. The output dimension of the encoder is 1024.

4.4.4 Models

Baseline models We have following baseline models: **CAT1**: A fully connected network classifier with one hidden layer with ReLu non-linearity, followed by a softmax layer. There are 8192 units in the hidden state. (We name it one-layer MLP for all the following experiments.) The input is a concatenated vector of one pre-trained question vector feature and one pre-trained image vector feature. **CAT1L**: A one-layer MLP given concatenated end-to-end 2-layer LSTM's last hidden state and one pre-trained image vector feature. In LSTM, the hidden state length is 1024. The word embedding dimension is 300. **CATL**: A one-layer MLP with concatenation of two pre-trained image vector features from ResNet and Faster R-CNN, and the last hidden layer of a 2-layer LSTM.

To check the complementarity of different features between ResNet and Faster R-CNN and show how they perform differently across question types, we set up baseline **CAT2**: A one-layer MLP given concatenated one pre-trained question vector feature and two independent pre-trained image vector features from ResNet and Faster R-CNN.

To further exam and explain our QTA proposal, we use more sophisticate feature integration operators as a strong baseline to compare with. **MCB-A**, as we mentioned in Section 4.2, is proposed in [33]. **RAU** [76] is a framework that combines the embedding, attention and predicts operation together inside a recurrent network. We reference results of these two models from [53].

QTA models From the baseline analysis, we realize that ResNet and Faster R-CNN features are complementary to each other. Using question type as guidance for image feature selection is the key to make image feature stronger. Therefore, we propose QTA networks as follows: **CAT-QTA**: A one-layer MLP given concatenated one pre-trained question vector feature and two weighted pre-trained image vector features from ResNet and Faster R-CNN. **CATL-QTA**: A one-layer MLP given concatenated output from a two-layer LSTM and two weighted pre-trained image vector features. **MCB-QTA**: Two MCB([33]) with spatial attention using ResNet and Faster-RCNN as image feature respectively. Text feature is a concatenation of output from a two-layer LSTM and Word2Vec question embedding and is shared for the two MCB parts. The out dimension of count sketch in the MCB is 8000. The resulting MCB outputs are then weighted by question type. Finally, the weighted representation is concatenated with the text feature before feeding to a one-layer MLP.

To check whether the model benefits from the QTA mechanism or from added question type information itself, we design a network that only uses question type embedding without attention. **CAT-QT**: A one-layer MLP given concatenated one pre-trained question vector feature, two independent pre-trained image vector features from ResNet and Faster R-CNN and a 1024-dim question type embedding. **CATL-QT**: A one-layer MLP given concatenated last hidden layer of a 2-layer LSTM, two independent pre-trained image vector features from

ResNet and Faster R-CNN and a 1024-dim question type embedding.

As mentions in Section 4.3, we propose a multi-task network for QTA in case we don't have question type label at inference. **CATL-QTA-M**:A multi-task model based on CATL-QTA. Text feature is a concatenation of LSTM and Word2Vec embedding. The output of LSTM is connected to a one-layer MLP to predict question type for the input question. The prediction result is then fed into QTA part through argmax.

4.5 Results and Analysis

We first focus in sections 4.5.1 and 4.5.2 on results concerning the complementarity of different features across question category types. For the visual domain, we explore the use of Faster R-CNN and ResNet features, while for the lexical domain we use NMT, LSTM and pre-trained Word2Vec features. We then analyze in subsection 4.5.3 the effect of explicitly introducing information about question type both as input and with QTA. Finally, in the remaining subsections, we extend the basic concatenation QTA model to MCB style pooling, introduce question type as both input and output during training such that the network can produce predicted question types during inference, and study more in depth the effect of the question category "Absurd" on the overall model performance across categories, which makes QTA generalizable to VQA settings that lack question types at test time.

4.5.1 Faster R-CNN and ResNet Features

Table 4.1 reports our extensive ablation analysis of simple concatenation models using multiple visual and lexical feature sources. From the results in the second and third columns, we see that overall the model with Faster R-CNN features outperform the one using ResNet features when using NMT features. We show in column 4 that the features sources are com-

Table 4.1: Benchmark results of concatenation models on TDIUC dataset using different image features and pre-trained language feature. 1: Use ResNet feature and SkipGram feature 2: Use ResNet feature and NMT feature 3: Use Faster R-CNN feature and NMT feature 4: Use ResNet feature and end-to-end LSTM feature 5: Use Faster R-CNN feature and end-to-end LSTM feature. N denotes that additional NMT embedding is concatenated to LSTM output. W denotes that additional Word2Vec embedding is concatenated to LSTM output(Following tables also use the same notation)

Accuracy(%)	**CAT1**^{1} [53]	**CAT1**^{2}	**CAT1**^{3}	**CAT2**	**CAT1L**^{4}	**CAT1L**^{5}	**CAT1L**^{4N}	**CAT1L**^{5N}	**CAT1L**^{4W}	**CAT1L**^{5W}
Scene Recognition	72.19	68.51	68.81	**69.06**	91.62	**92.27**	91.16	**92.33**	91.57	**92.45**
Sport Recognition	85.16	89.67	92.36	**93.15**	90.94	**93.84**	89.62	**93.52**	90.77	**94.05**
Color Attributes	43.69	32.90	34.35	**34.99**	45.62	**49.43**	44.07	**47.78**	47.33	**49.47**
Other Attributes	42.89	38.05	**39.76**	39.67	40.89	**43.49**	39.60	**42.35**	41.92	**45.19**
Activity Recognition	24.16	39.34	45.75	**46.87**	42.95	**49.25**	40.12	**44.11**	42.13	**49.25**
Positional Reasoning	25.15	25.63	27.16	**28.02**	26.22	**29.35**	24.17	**27.50**	25.72	**28.59**
Sub. Object Recognition	80.92	83.94	85.67	**86.78**	82.20	**85.06**	81.85	**84.47**	82.52	**85.05**
Absurd	96.96	94.98	94.77	**95.82**	**90.87**	87.10	**95.38**	93.28	**93.59**	91.95
Utility and Affordances	24.56	25.93	**27.78**	27.16	15.43	**25.93**	**25.31**	18.52	16.05	**17.28**
Object Presence	69.43	77.21	77.90	**78.29**	89.40	**91.14**	90.13	**91.95**	91.08	**91.81**
Counting	44.82	48.46	52.18	**52.57**	45.95	**50.27**	44.26	**49.24**	44.93	**51.30**
Sentiment Understanding	53.00	43.45	46.49	**47.28**	46.49	**48.72**	41.85	**42.81**	44.89	**46.01**
Overall (Arithmetic MPT)	55.25	55.67	57.57	58.31	59.05	62.15	58.96	60.66	59.38	**61.80**
Overall (Harmonic MPT)	44.13	45.37	47.99	48.44	44.09	**51.66**	46.84	46.84	44.42	47.70
Overall Accuracy	69.53	71.41	72.44	73.05	77.55	78.66	78.35	79.94	78.94	**80.16**

plementary, and their combination is better across most categories (in bold) with respect to the single source models of columns 2 and 3. In columns 5,6; 7,8 and 9,10 we replicate the same comparison between ResNet and R-CNN features using more sophisticate models to embed the lexical information. We reach more than 10 % accuracy increase, from 69.53 % to 80.16 % using a simple concatenation model with an accurate selection of the feature type.

4.5.2 Pre-trained and Jointly-trained Text Feature Extractors

The first four columns in Table 4.1 show the results of models with text features from NMT. To fully explore the text feature extractor in VQA system, we substitute the NMT pre-trained language feature extractor with a jointly-trained two layer LSTM model. The improved performance of jointly-training text feature extractor can be appreciated by comparing the results of the 4 left-most and right most columns. For example, comparing second column and fifth column in Table 4.1, we get 6% improvement using LSTM while keeping image

Table 4.2: QTA in concatenation models on TDIUC dataset

Accuracy(%)	**CATL**	**CATL-QTA**	**CATL**^{W}	**CATL-QTA**^{W}
Scene Recognition	93.18	93.45	93.31	93.80
Sport Recognition	94.69	95.45	94.96	95.55
Color Attributes	54.66	56.08	57.59	60.16
Other Attributes	48.52	50.30	52.25	54.36
Activity Recognition	53.36	58.43	54.59	60.10
Positional Reasoning	32.73	31.94	33.63	34.71
Sub. Object Recognition	86.56	86.76	86.52	86.98
Absurd	95.03	100.00	98.01	100.00
Utility and Affordances	29.01	23.46	29.01	31.48
Object Presence	93.34	93.48	94.13	94.55
Counting	50.08	49.93	52.97	53.25
Sentiment Understanding	56.23	56.87	62.62	64.38
Overall (Arithmetic MPT)	65.62	66.34	67.46	69.11
Overall (Harmonic MPT)	55.95	54.60	57.83	60.08
Overall Accuracy	82.23	83.62	83.92	85.03

feature and network same.

We obtain the best model by concatenating the output of the LSTM and the pre-trained NMT/Word2Vec feature, as shown in Table 4.1. It gives us 10% improvement for "Utility and Affordances" when we look at the fifth and seventh column. We find the use of Word2Vec is better than NMT feature in third and fourth columns in Table 4.3 and in last four columns in Table 4.1. We think the better performance of Word2Vec with respect to the NMT encoder, might be due to the more similar structure of single sentence samples of Word2Vec training set with those from classical VQA dataset with respect to those used for training NMT models.

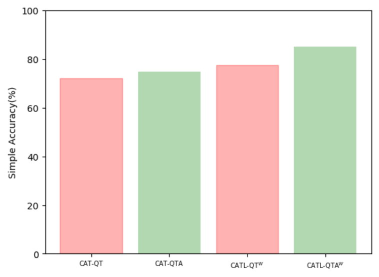


Figure 4.4: Evaluation of different ways to utilize information from question type.

4.5.3 QTA in Concatenation Models

We use QTA in concatenation models to study the effect of QTA. The framework is in Figure 4.1. We compare the network using a weighted feature(column 1 in Table 4.3) with the same network using an unweighted concatenated image feature(column 4 in Table 4.1). As we can see, the model using the weighted feature has more power than the one using the unweighted feature. 9 out of 12 categories get improved results. "Color" and "Other attributes" get around 9% accuracy increase.

To ensure that the improvement is not because of the added question type information but the attention mechanism using question type, we show the comparison of QTA with QT in Figure 4.4. With same text feature and image feature and approximately same number of parameters in the network, QTA is 3-5% better than QT.

We show the effect of QTA on image feature norms in Figure 4.5. By weighing the image features by question type, we find that our model relies more on Faster R-CNN features for "Absurd" question samples while it relies more on ResNet features for "Color" questions.

The best setting we get in concatenation model is using a weighted image feature concatenated with the output of the LSTM and Word2Vec feature(CATL-QTA^W). It gets 5% improvement compared to a similar generic model without QTA and also shows better performance than complicated deep network such as RAU and MCB-A in Table 4.3.

4.5.4 QTA in Pooling Models

To show how to combine QTA with more complicated feature integration operator, we propose MCB-QTA structure. Even though MCB-QTA in Table 4.3 doesn't win with simple accuracy, it shows great performance in many categories such as "Object Recognition" and "Counting". Accuracy in "Utility and Affordances" is improved by 6% compared to our

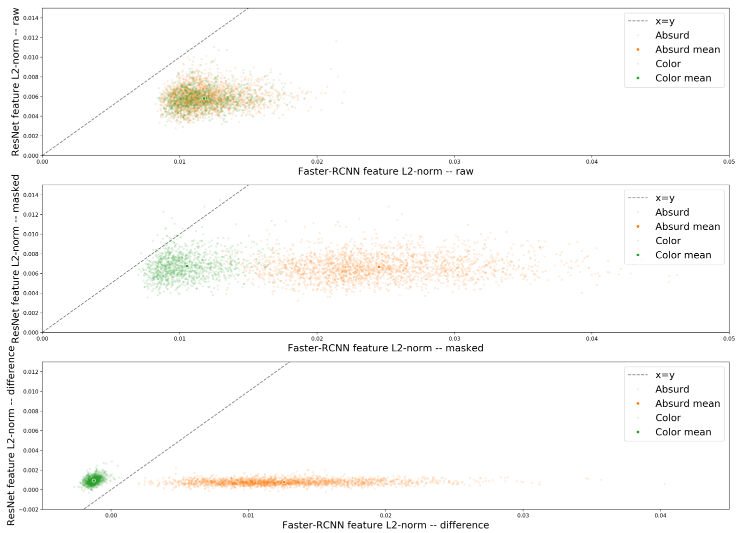


Figure 4.5: Effects of weighting by QTA. Top: raw feature norms, Middle: feature norms weighted by QTA, Bottom: differences of norms after weighting vs before weighting. For color questions, the feature norms shift towards ResNet features, while for absurd questions they shift towards Faster-RCNN features.

Table 4.3: Results of QTA models on TDIUC dataset compared to state-of-art models

Accuracy(%)	**CATL-QTA^W**	**MCB-QTA**	**MCB-A** [53]	**RAU** [53]
Scene Recognition	93.80	93.56	93.06	**93.96**
Sport Recognition	95.55	**95.70**	92.77	93.47
Color Attributes	60.16	59.82	**68.54**	66.86
Other Attributes	54.36	54.06	**56.72**	56.49
Activity Recognition	60.10	**60.55**	52.35	51.60
Positional Reasoning	34.71	34.00	**35.40**	35.26
Sub. Object Recognition	86.98	**87.00**	85.54	86.11
Absurd	**100.00**	100.00	84.82	96.08
Utility and Affordances	31.48	**37.04**	35.09	31.58
Object Presence	**94.55**	94.34	93.64	94.38
Counting	53.25	**53.99**	51.01	48.43
Sentiment Understanding	64.38	65.65	**66.25**	60.09
Overall (Arithmetic MPT)	69.11	**69.69**	67.90	67.81
Overall (Harmonic MPT)	60.08	**61.56**	60.47	59.00
Overall Accuracy	**85.03**	84.97	81.86	84.26

CATL-QTA model. It gets 8% improvement in “Activity recognition” compared to state-of-art model MCB-A and also gets the best Arithmetic and Harmonic MPT value.

4.5.5 Multi-task Analysis

In this part, we will discuss how we use QTA when we have questions without specific question types. It is quite easy to predict the question type from the question itself. We use a 2-layer LSTM followed by a classifier and the accuracy for test question type is 96% after 9 epochs. The problem is whether we can predict the question type while keeping the same performance for VQA task or not. As described in Figure 4.2, we use the predicted question type as input of the QTA network in a multi-task setting. We get 84.33% test simple accuracy for VQA task as shown in Table 4.7. When we compare it to MCB-A or RAU in Table 4.3, though accuracy gets a little affected for most of the categories, we still get 2% improvement in “Sports Recognition” and “Counting”.

We fine-tune our model on VQA v1 using a pre-trained multi-task model that was trained on TDIUC. VQA v1 doesn’t have question type information. We use the question type predictor in the multi-task model as the input of QTA. Our model’s performance is better than MCB in Table 4.4 with an approximately same number of parameters in the network.

4.5.6 Findings on TDIUC dataset

To further analyze the effects of the question type prediction part in this multi-task framework, we list the confusion matrix for the question type prediction results in Table 4.5. “Color” and “Absurd” question type predictions are most often bi-directionally confused. The reason for this is that among all absurd questions, more than 60% are questions start with “What color”. To avoid this bias, we remove all absurd questions and run our multi-

Table 4.4: Results of test-dev accuracy on VQA v1. Models are trained on the VQA v1 train split and tested on test-dev

	Accuracy(%)
Element-wise Sum [33]	56.50
Concatenation [33]	57.49
Concatenation + FC [33]	58.40
Concatenation + FC + FC [33]	57.10
Element-wise Product [33]	58.57
Element-wise Product + FC [33]	56.44
Element-wise Product + FC + FC [33]	57.88
MCB($2048 \times 2048 \rightarrow 16K$) [33]	59.83
CATL-QTA-M + FC	**60.32**

task model again. In this setting, our question type prediction did much better than before. Almost all categories get 99% accuracy as shown in Table 4.6. We also compare our QTA models' performance without absurd questions in Table 4.7. In CATL-QTA network, removing absurd questions doesn't help much because in test we feed in the true question type labels. But it is useful when we consider the multi-task model. From third and fourth columns, we see that without absurd questions, we get improved performance among all categories. This is because we remove the absurd questions that may mislead the network to predict "color" question type in the test.

Another concern we have is that since "Absurd" questions only have one unique answer which is absurd and cover 20% of the questions in TDIUC dataset, is our model better than others only because we feed 20% true answers to the model? The answer is no. From Table 4.7, our QTA model gets 17% improvement on simple accuracy compared to the baseline in [53] when we exclude absurd questions in training.

Table 4.5: Confusion matrix for test question types prediction in CATL-QTA-M using TDIUC dataset. 1. Other Attributes 2. Sentiment Understanding 3. Sports Recognition 4. Position Reasoning 5. Object Utilities/Affordances 6. Activity Recognition 7. Scene Classification 8. Color 9. Object Recognition 10.Object Presence 11.Counting 12. Absurd

Target	Predicted												Acc(%)
	1	2	3	4	5	6	7	8	9	10	11	12	95.66
1	77.76	0.00	0.89	3.20	0.00	0.08	0.42	1.15	0.12	0.00	0.00	16.38	
2	0.80	60.51	1.77	8.83	0.00	2.25	2.57	0.00	1.44	0.96	0.16	20.71	
3	0.31	0.00	73.08	0.37	0.00	0.17	0.00	0.03	0.02	0.00	0.01	26.01	
4	2.95	0.02	0.01	89.52	0.00	0.01	0.02	0.19	1.88	0.03	0.03	5.35	
5	12.50	0.63	3.12	45.62	0.00	0.00	3.12	0.00	11.25	0.00	0.00	23.75	
6	0.79	0.00	14.56	1.76	0.00	13.18	0.00	0.00	2.21	0.00	0.07	67.43	
7	0.04	0.00	0.04	0.40	0.00	0.01	99.40	0.02	0.00	0.00	0.06	0.03	
8	0.32	0.00	0.18	0.13	0.00	0.00	0.00	86.10	0.00	0.00	0.00	13.28	
9	0.01	0.00	0.00	0.31	0.00	0.00	0.00	0.00	98.96	0.01	0.00	0.71	
10	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	100.00	0.00	0.00	
11	0.00	0.00	0.00	0.01	0.00	0.00	0.02	0.00	0.02	0.05	99.90	0.00	
12	0.35	0.00	0.18	0.41	0.00	0.03	0.00	3.18	0.40	0.00	0.00	95.46	

Table 4.6: Confusion matrix for test question types prediction in CATL-QTA-M using TDIUC dataset without absurd questions. Numbers represent same categories as in Table 4.5

Target	Predicted												Acc(%)
	1	2	3	4	5	6	7	8	9	10	11	12	99.50
1	98.39	0.00	0.07	0.15	0.00	0.13	0.08	0.63	0.55	0.00	0.00	N/A	
2	0.16	84.03	3.67	0.00	0.00	3.35	5.59	0.00	0.48	0.00	2.72	N/A	
3	0.00	0.08	97.31	0.00	0.00	2.37	0.01	0.00	0.10	0.02	0.11	N/A	
4	1.01	0.00	0.00	98.07	0.00	0.01	0.00	0.51	0.41	0.00	0.00	N/A	
5	8.64	3.70	14.81	0.00	0.00	59.26	7.41	1.23	4.94	0.00	0.00	N/A	
6	0.45	0.15	31.42	0.00	0.00	67.39	0.04	0.04	0.45	0.00	0.07	N/A	
7	0.02	0.03	0.00	0.00	0.00	0.03	99.86	0.02	0.00	0.00	0.04	N/A	
8	0.06	0.00	0.00	0.13	0.00	0.04	0.07	99.70	0.00	0.00	0.00	N/A	
9	0.06	0.00	0.13	0.01	0.00	0.02	0.00	0.00	99.76	0.01	0.00	N/A	
10	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	100.00	0.00	N/A	
11	0.00	0.00	0.01	0.00	0.00	0.00	0.00	0.00	0.00	0.01	99.98	N/A	
12	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	

Table 4.7: Results of test accuracy when question type is hidden with/without absurd questions in training. We compare them with similar QTA models. * denotes training and testing without absurd questions

	CATL-QTA^W	**CATL**^{W*}	**CATL-QTA**^{W*}	**CATL-QTA-M**	**CATL-QTA-M***	**CAT1**^{1*} [53]
Scene Recognition	93.80	93.46	93.62	93.74	93.82	72.75
Sport Recognition	95.55	94.97	95.47	94.80	95.31	89.40
Color Attributes	60.16	57.84	58.63	57.62	59.73	50.52
Other Attributes	54.36	53.90	53.44	52.05	56.17	51.47
Activity Recognition	60.10	57.38	59.43	53.13	58.61	48.55
Positional Reasoning	34.71	33.98	34.63	33.90	34.70	27.73
Sub. Object Recognition	86.98	86.62	86.74	86.89	86.80	81.66
Absurd	100.00	N/A	N/A	98.57	N/A	N/A
Utility and Affordances	31.48	27.78	34.57	24.07	35.19	30.99
Object Presence	94.55	93.87	94.22	94.57	94.60	69.50
Counting	53.25	52.33	52.20	53.59	55.30	44.84
Sentiment Understanding	64.38	64.06	65.81	60.06	61.31	59.94
Overall (Arithmetic MPT)	69.11	65.11	66.25	66.92	66.88	57.03
Overall (Harmonic MPT)	60.08	55.89	58.51	55.77	58.82	50.30
Simple Accuracy	85.03	79.79	80.13	84.33	80.95	63.30

4.6 Conclusion

We propose a question type-guided visual attention (QTA) network. We show empirically that with the question type information, models can balance between bottom-up and top-down visual features and achieve state-of-the-art performance. Our results show that QTA systematically improves the performance by more than 5% across multiple question type categories such as "Activity Recognition", "Utility" and "Counting" on TDIUC dataset. We consider the case when we don't have question type for test and propose a multi-task model to overcome this limitation by adding question type prediction task in the VQA task. We get around 95% accuracy for the question type prediction while keeping the VQA task accuracy almost same as before.

Chapter 5

Extended BLAS Kernels for Tensor Contraction

Multilinear algebraic computations, are ubiquitous in multiple scientific domains such as machine learning and modern data science [7], quantum chemistry and physics [55], signal and image processing [37], chemometrics [16], and biochemistry [54]. The study of tensor computations has a long and diverse history, as early as in the work by Hitchcock [45]. The domains and references provided herein are by no means exhaustive but merely a small representative sample of the various flavors in which tensor computations are used in science. *Tensor contractions* play a central role in a variety of algorithms and applications. However, non-trivial performance bottlenecks in several application areas are encountered due to the high space and time complexities associated with tensor computations. In this chapter, motivated by the recent increased interest from machine learning and deep learning, we propose and study library-based communication-avoiding approaches for performing tensor contractions.

Conventional approaches for computing general tensor contractions rely on *matricization*, the logical or explicit restructuring of the data so that the computation can be performed with a sequence of Basic Linear Algebra Subroutine (Blas) library calls. The Blas routines provide efficient and portable implementations of linear algebra primitives, with many fast implementations existing across many architectures [18].

To this point, the GEneral Matrix Multiply (Gemm) primitive specified within the Blas library is possibly the most optimized and widely used routine in scientific computing. Noting that the basic theoretical computational and communication complexities of most tensor contractions is equivalent to that of Gemm, these computations should scale equally well. However, we find that existing tensor libraries such as the Tensor Toolbox and Cyclops Tensor Framework perform explicit data transposition to compute almost all tensor contractions and the cost of data restructuring often dominates the cost of the actual computation. Other approaches [63, 67] have previously proposed intrusive compiler and static analysis solutions, whereas we provide a much simpler library-based solution.

5.1 Summary of Results

We introduce a new Blas primitive, known as StridedBatchedGemm, that allows the majority of tensor contractions to be computed without any explicit memory motion. We begin by focusing on single-index contractions involving all the possible configurations of second-order and third-order tensors. We detail the so-called exceptional cases that cannot be evaluated with StridedBatchedGemm and demonstrate that an efficient solution exists with another small extension to the primitive. Through systematic benchmarking, we demonstrate that our approach can achieve 10x speedup on a K40c GPU and 2x speedup on dual-socket Haswell-EP CPUs, using Mkl and CuBlas respectively, for small and moderate tensor sizes. This is relevant in many machine learning applications such as deep learning, where tensor sizes tend to be small, but require numerous tensor contraction operations to be performed successively. We also demonstrate performance improvement using our approach in direct benchmarks to an application study: the Tucker decomposition. We show that using our kernels yields atleast an order of magnitude speedup as compared to state-of-the-art libraries.

Finally, the value of this approach and its applications are recognized by NVIDIA. The proposed interface exists in the CuBlas 8.0.

5.2 Related Work

Peise *et al* [81] extended results from Napoli *et al* [73] in mapping tensor contractions to sequences of Blas routines and modeling the performance of these mappings. In this work, they systematically enumerate and benchmark combinations of possible Blas kernels one could use to compute a given tensor contraction to conclude that the best performing algorithms involve the Gemm kernel. Some evaluation strategies are neglected to be considered, such as *flattening* or developing new, generic linear algebraic subroutines that could yield improved performance.

Lu *et al* [67] produce an optimizing compiler to determine the number and sequence of transpositions of a general tensor contraction so that the evaluation can be performed with a Gemm call. Li *et al* [63] also recognizes the cost of explicit copies and proposes evaluation strategies exactly comparable to the flattening and batching strategies addressed in this chapter. Their discussion of *loop modes* and *component modes* map to our discussion of *batch modes* and Gemm *modes*. However, Li *et al* do not discuss strategies beyond tensor-times-matrix multiply. Furthermore, they only consider *mode-n* tensor-times-matrix contractions of the form $Y_{i_1 \cdots i_{n-1} j \cdots i_N} = \sum_{i_n} X_{i_1 \cdots i_N} U_{j i_n}$, which avoids the more complicated cases in this chapter. Abdelfattah *et al* [3] presents a framework using batched Gemm for tensor contractions on GPUs. However, they focus on optimizing only limited number of tensor contraction kernels on extreme small size tensors. Other works in [1] [74] improve the tensor computation performance by doing loop reorganization and fusion.

The StridedBatchedGemm interface proposed in this chapter has previously been mentioned by Jhurani *et al* [51] as a low-overhead interface for multiple small matrices on NVIDIA GPUs. Jhurani proposes the same interface for CuBlas that we propose and focuses on implementation concerns. In this work, we treat StridedBatchedGemm as an available primitive, benchmark evaluation strategies that utilize it, and examine how it may be further extended for use in multi-linear algebra.

The Blas-like Library Instantiation Software (Blis) framework [99] offers Gemms which support non-unit strides in *both* the row and column dimensions, which are attractive solutions to some of the problems in this chapter. However, performance is expected to suffer due to decreases in cache line utilization, and SIMD opportunities.

Recent improvements in parallel and distributed computing systems have made complex tensor computation feasible. TensorFlow [2] can handle multi-linear algebra operations and it is primarily a data-flow and task-scheduling framework for machine learning.

5.3 Preliminaries

Notation We follow Einstein summation convention to represent tensor contractions. A general tensor contraction is written as

$$C_{\mathcal{C}} = \alpha\, A_{\mathcal{A}}\, B_{\mathcal{B}} + \beta\, C_{\mathcal{C}} \tag{5.1}$$

where $\mathcal{A}, \mathcal{B}, \mathcal{C}$ are ordered sequences of indices such that $\mathcal{C} \equiv (\mathcal{A} \cup \mathcal{B}) \setminus (\mathcal{A} \cap \mathcal{B})$. The indices in $\mathcal{A} \cap \mathcal{B}$ are called *contracted indices*. The indices in $\mathcal{C}$ are called *free indices*.

Conventional Tensor Contraction The conventional approach for tensor contraction is to *matricize* the tensors via transpositions and copies. Libraries such as Basic Tensor Algebra Subroutines (BTAS) [72], MATLAB Tensor Toolbox [13, 12], and Cyclops Tensor Framework [90] all perform some version of matricization, which is typically performed in four steps:

1. Consider a general tensor contraction of the form (5.1). Define the index sets $\mathcal{K}$, $\mathcal{I}$, $\mathcal{J}$ as

$$\mathcal{K} = \mathcal{A} \cap \mathcal{B}, \quad \mathcal{I} = \mathcal{A} \setminus (\mathcal{A} \cap \mathcal{B}), \quad \mathcal{J} = \mathcal{B} \setminus (\mathcal{A} \cap \mathcal{B})$$

2. Permute tensors A, B, and C into the form

$$C_{\mathcal{I}\mathcal{J}} = \alpha \, A_{\mathcal{I}\mathcal{K}} \, B_{\mathcal{K}\mathcal{J}} + \beta \, C_{\mathcal{I}\mathcal{J}} \tag{5.2}$$

3. Evaluate (5.2) using one of four BLAS kernels:

$$\begin{cases} \textsc{Dot} & |\mathcal{K}| = |\mathcal{A}| \text{ and } |\mathcal{K}| = |\mathcal{B}| \\ \textsc{Ger} & |\mathcal{K}| = 0 \\ \textsc{Gemv} & |\mathcal{K}| = |\mathcal{A}| \text{ xor } |\mathcal{K}| = |\mathcal{B}| \\ \textsc{Gemm} & \text{else} \end{cases}$$

4. Permute the result, $C_{\mathcal{I}\mathcal{J}}$, into the desired output, $C_{\mathcal{C}}$.

This approach to tensor contractions is completely general. It works for any two tensors of arbitrary order and any number of contraction indices. However, for even the simplest contractions, the cost of explicitly permuting the tensor data typically outweigh the cost of the computation to be performed. See Section 5.4.1 for examples.

An Important Practical Application In unsupervised learning, tensor decomposition [7] is gaining a lot of attention and is the crux of model estimation via the method of moments. A variety of problems such as topic model estimation, Gaussian mixtures model estimation, and social network learning can be provably, consistently and efficiently solved via the tensor decomposition techniques under certain mild assumptions.

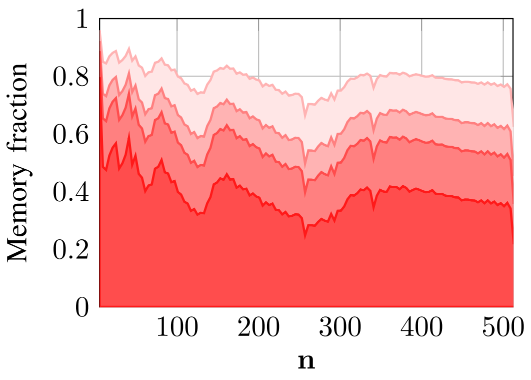
The basic building blocks of these algorithms involve tensor contractions. Two frequently used tensor decomposition methods are the CP decomposition [42] and the Tucker decomposition [98]. In [100], the authors use the Tucker decomposition to extract new representations of the face images despite different expressions or camera viewpoints. To illustrate the fundamental importance of tensor contractions, we will pick one of the most common tensor decomposition algorithms, namely the higher-order orthogonal iteration (HOOI) [62] for asymmetric Tucker decomposition, and use it as a case-study. In the Einstein notation, the factorization of a third-order tensor $T \in \mathbb{R}^{\mathbf{m}\times\mathbf{n}\times\mathbf{p}}$ is given by $T_{mnp} = G_{ijk}A_{mi}B_{nj}C_{pk}$, where $G \in \mathbb{R}^{\mathbf{i}\times\mathbf{j}\times\mathbf{k}}$ is the core tensor, $A \in \mathbb{R}^{\mathbf{m}\times\mathbf{i}}$, $B \in \mathbb{R}^{\mathbf{n}\times\mathbf{j}}$, $C \in \mathbb{R}^{\mathbf{p}\times\mathbf{k}}$. From Kolda et al [94], we summarize the algorithm for the third-order tensor case in Algorithm 3. Following their notation, $T_{(r)}$ denotes the mode-r unfolding of tensor T. For further technical details, we refer the reader to Kolda et al [94].

Algorithm 3 Tucker Decomposition Algorithm

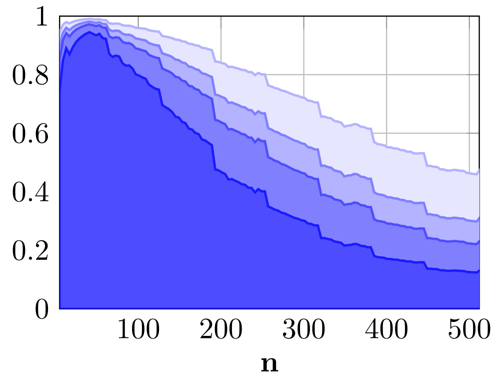
Require: Tensor $T \in \mathbb{R}^{\mathbf{m}\times\mathbf{n}\times\mathbf{p}}$, core tensor size $\mathbf{i}$, $\mathbf{j}$, $\mathbf{k}$, number of iterations $\mathcal{T}$.

Ensure: Factors $A^{\mathcal{T}}$, $B^{\mathcal{T}}$, $C^{\mathcal{T}}$ and core tensor G

```
1: Set $t = 0$;
2: Initialize $A^0 \leftarrow \mathbf{i}$ leading left singular vector of $T_{(1)}$
3:            $B^0 \leftarrow \mathbf{j}$ leading left singular vector of $T_{(2)}$
4:            $C^0 \leftarrow \mathbf{k}$ leading left singular vector of $T_{(3)}$
5: while $t < \mathcal{T}$ do
6:     $Y_{mjk} = T_{mnp} B^t_{nj} C^t_{pk}$
7:     $A^{t+1} \leftarrow \mathbf{i}$ leading left singular vector of $Y_{(1)}$.
8:     $Y_{ink} = T_{mnp} A^{t+1}_{mi} C^t_{pk}$
9:     $B^{t+1} \leftarrow \mathbf{j}$ leading left singular vector of $Y_{(2)}$.
10:    $Y_{ijp} = T_{mnp} B^{t+1}_{nj} A^{t+1}_{mi}$
11:    $C^{t+1} \leftarrow \mathbf{k}$ leading left singular vector of $Y_{(3)}$.
12: $G_{ijk} = T_{mnp} A^{\mathcal{T}}_{mi} B^{\mathcal{T}}_{nj} C^{\mathcal{T}}_{pk}$
```



(a) CPU



(b) GPU

Figure 5.1: The fraction of time spent in copies/transpositions when computing the contraction $C_{mnp} = A_{mk} B_{pkn}$ using the conventional approach.

5.4 Approach

In this section, we present library-based evaluation strategies for performing general tensor contractions in-place: without explicit copies and/or transpositions.

5.4.1 Motivating Observations

Case study 1 Consider $C_{mnp} = A_{mk}B_{nkp}$. The conventional approach presented in Section 5.3 results in an evaluation wherein one switches, by means of explicit copy operations, modes n and k in B to produce $C_{mnp} = A_{mk}B_{knp}$, which is now of the form (5.2) and can be evaluated directly with a GEMM. Alternatively, we observe that we may perform the computation without explicit copy by launching $\mathbf{p}$ individual GEMMs.

Case study 2 Consider $C_{mnp} = A_{km}B_{pkn}$. The conventional approach presented in Section 5.3 results in an evaluation wherein we may require more than one transposition. For concreteness, we analyzed how BTAS performs this contraction. We observed that BTAS uses four explicit transpositions that results in the following algorithm:

1. Permute A_{km} to A_{mk}.
2. Permute B_{pkn} to B_{kpn}.
3. Permute C_{mnp} to C_{mpn}.
4. Compute $C_{mpn} = \alpha A_{mk}B_{kpn} + \beta C_{mpn}$ with GEMM.
5. Permute C_{mpn} to C_{mnp}.

Similarly, in the MATLAB Tensor Toolbox, the main idea is to reshape all tensors to matrices. For instance, in Case 2.4 in Table 5.2, it reshapes A_{km} to A_{mk} and reshapes tensor B_{pkn} to matrix $B_{k(pn)}$ with the first dimension as $\mathbf{k}$ and the second dimension as $\mathbf{p} * \mathbf{n}$. Cyclops also uses index reordering methods for fully dense tensors. The reordering is avoided only in the more restrictive case of high-dimensional symmetric tensors.

We note that some of the steps in the above approach can certainly be avoided with an improved algorithm that still implements the conventional approach. For example, Step 1 can be avoided by using a GEMM that implicitly transposes the first matrix via a `CblasTrans` parameter or equivalent in Step 4. Another optimization would be to avoid Step 3 altogether when $\beta = 0$. Other approaches require even fewer transposition steps. Ultimately, observe that we may perform the computation without explicit copy by performing $\mathbf{p}$ individual GEMMS.

In Figure 5.1, we measure the cost of these explicit transpose operations in a representative tensor contraction on CPU and GPU. Lines are shown with 1, 2, 3, and 6 total transpositions performed on either the input or output. On the CPU we use MKL's `mkl_somatcopy` and `cblas_sgemm`, and on the GPU we use CUBLAS's `cublasSgeam` and `cublasSgemm` to perform each required matrix transposition and GEMM respectively. Note that transposition primitives are not specified in BLAS, but are vendor-specific BLAS-like extensions provided to perform common transpose operations. For this reason, these optimized functions are not typically used in tensor libraries with, instead, custom transposition implementations taking their place. These custom implementations are likely not as optimized as the vendor implementations.

As we can see from Figure 5.1, on the CPU, almost 40% of the time is used in copy and transpose, even when only a single mode transposition is performed. Clearly, with more transpose operations, the fraction is higher, requiring 60-80% of the total time. This correlates well with data presented in [63] where it is reported that Tensor Toolbox takes approximately 70% of the total time performing copies and transpositions in one algorithm. By avoiding these transpositions we may obtain 10x speedup on the GPU for small tensors with $\mathbf{n} \lesssim 100$, and more than 2x speedup on the CPU for almost all $\mathbf{n}$.

Although the fraction of time spent in transposition will asymptotically approach zero as $\mathbf{n}$ grows in both cases, the high bandwidth of the GPU allows the computation to dominate the communication much more quickly. Indeed, the reported maximum bandwidth of the K40c GPU is 288GB/sec and the dual-socket Xeon E5-2630 v3 CPU achieves 118GB/sec.

Additionally, that the gap between computational performance and communication performance continues to increase, so the cost of transposition is likely to increase in the future. Even now, especially for small tensor sizes, it is clear that the cost of performing explicit copies and transpositions is significant and should be avoided.

5.4.2 Extended Notations

We would like to express evaluation strategies for tensor contractions succinctly, so we introduce additional notation.

In this chapter, tensors are assumed to be stored in the *column-major* format. In other words, the i^{th} mode has a memory stride – termed "leading dimension" in BLAS – denoted $\texttt{ld<}i\texttt{>}$ with $\texttt{ld<0>} = 1$. Using this notation, A_{mnp} is stored as $A[m+n*\texttt{ld<1>}+p*\texttt{ld<2>}]$. Note that the common *packed-storage* case is obtained when, for all i, we have $\texttt{ld<}i\texttt{>} = \prod_{0\leq k<i} \texttt{dim<}k\texttt{>}$.

We now formalize three operations that are used in tensor contraction evaluations.

1. Batching: $[i]$ denotes that mode i is *batched*, A batched mode is considered *fixed*.
2. Flattening: (ij) denotes that modes i and j are *flattened*, i.e., , modes i and j are now considered together as a single mode. The combined mode $h = (ij)$ is considered *free*.
3. Transpose: $A_{mn}^{\top}$ denotes a matrix *transpose*. Transposes may only be applied to tensors with exactly two free modes.

The purpose of these notations is that they map directly to looped BLAS calls and the appropriate evaluation can often be read directly from the notated expression. Next, we review some rules that the above notation must follow in order to obtain a well-formed evaluation expression.

1. A batched mode $[i]$ cannot be the first mode of any matrix term. That is, $A_{[m]nk}$ is not allowed. Batching in the first mode would cause the $\mathbf{m}$ resulting logical $\mathbf{n} \times \mathbf{k}$ matrices

to be strided in both rows and columns and, therefore, cannot be used as a matrix in any BLAS routine.

2. A flattening (ij) requires that `ld<`j`>` $=$ `ld<`i`>` `dim<`i`>`. Unfortunately, the notation alone is therefore not sufficient to determine which modes may be flattened; it is contingent on the representation as well. In the common packed-storage case, however, this flattening condition is always true.
3. If a flattening operation occurs on the right side, it must occur on the left side with the same modes in the same order. For example, $C_{m(np)} = A_{mk}B_{k(pn)}$ is not allowed.
4. Standard transposition rules apply: $C^\top_{nm} = A_{mk}B_{kn}$ implies $C_{nm} = B^\top_{kn}A^\top_{mk}$. However, modes may not be swapped under transposition. For example, $A^\top_{mk}$ can not be replaced with A_{km}.

This notation allows us to quickly read off the intended extended BLAS evaluation expression for arbitrary tensor contractions. See Table 5.1 for examples.

Table 5.1: Example mapping between tensor contractions with batched and flattened modes in our notation and the corresponding BLAS expression evaluation. Note that the appropriate BLAS primitive, transposition, matrix pointer, and leading dimension parameters to GEMM can be read off directly from the notation

Contraction	BLAS Evaluation
$C_{m(np)} = A_{mk}B_{k(np)}$	GEMM ('N','N', **m**, **np**, **k**, 1, A, `lda<1>`, B, `ldb<1>`, 0, C, `ldc<1>`);
$C_{(mn)p} = B_{(mn)k}A^\top_{pk}$	GEMM ('N','T', **mn**, **p**, **k**, 1, B, `ldb<1>` $\cdot$ `ldb<2>`, A, `lda<1>`, 0, C, `ldc<1>` $\cdot$ `ldc<2>`);
$C_{m[n]p} = A_{mk}B_{k[n]p}$	for n in [0,**n**) GEMM ('N','N', **m**, **p**, **k**, 1, A, `lda<1>`, $B + n \cdot$ `ldb<1>`, `ldb<2>`, 0, $C + n \cdot$ `ldc<1>`, `ldc<2>`);
$C_{mn[p]} = B^\top_{k[p]m}A_{kn}$	for p in [0,**p**) GEMM ('T','N', **m**, **n**, **k**, 1, $B + p \cdot$ `ldb<1>`, `ldb<2>`, A, `lda<1>`, 0, $C + p \cdot$ `ldc<2>`, `ldc<1>`);
$C_{[n]p} = B_{pk}A_{k[n]}$	for n in [0,**n**) GEMV ('N', **p**, **k**, 1, B, `ldb<1>`, $A + n \cdot$ `lda<1>`, 1, 0, $C + n$, `ldc<1>`);

5.4.3 BatchedGemm

Instead of relying on explicit mode transpositions, Peise *et al* [80, 81] considered mapping tensor contractions to BLAS primitives directly – enumerating all possible BLAS primitives that could be used and their nesting within loops. Of course, the evaluation strategies that relied on level-3 BLAS primitives (GEMM) rather than level-2 primitives (GEMV, GER) were

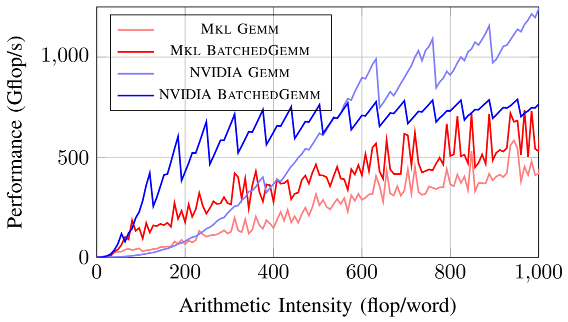


Figure 5.2: The arithmetic intensity of computing n Gemms of size $n \times n$.

much more efficient. This often resulted in the need for many small Gemms to be performed, which usually does not achieve ideal performance.

The need to compute many small Gemms has not gone unnoticed by the leading implementations of Blas. NVIDIA supplied the capability to multiply pairs of many small matrices in CuBlas v4.1 [CUDA Toolkit v4.1] via the function `cublasXgemmBatched`. Similarly, as of Mkl 11.3β, `cblas_Xgemm_batch` is available with a similar interface and is also specifically optimized for small matrix sizes.

In Figure 5.2, we plot the achieved performance on CPU and GPU of these BatchedGemm functions by evaluating $\mathbf{n}$ Gemms of size $\mathbf{n} \times \mathbf{n}$ using each strategy with Mkl 11.3.1 and CuBlas 7.5. All experiments are performed on a K40c GPU and 16 cores (32 threads) of a dual socket CPU. Note that there are much higher performance in both cases when $\mathbf{n}$ is small. When $\mathbf{n}$ is large, there is clearly room for optimization in `cublasSgemmBatched`.

Both of these interfaces are based on pointers to matrix pointers, which often require allocation and/or precomputation at the point-of-call. This makes them awkward to use in the context of tensor contractions where the strides between matrices are regular and the generality provided by these interfaces goes unused.

5.4.4 StridedBatchedGemm

Building on the BatchedGemm extensions to Blas, we propose StridedBatchedGemm (Listing 5.1) which offers a simplified interface for the constant-strided BatchedGemm and more optimizations opportunities for implementors. The interface and reference implementation of StridedBatchedGemm is provided in Listing 5.1. The `lda`, `ldb`, `ldc` parameters are the standard "leading dimension" parameters that appear in level-3 Blas primitives and denote to the stride between columns of the matrix. We refer to the new `loa`, `lob`, `loc` parameters as the "leading order" parameters and denote the stride between matrices of the batch.

There are a number of advantages to a StridedBatchedGemm primitive. First, StridedBatchedGemm is actually more restrictive than the BatchedGemm that has already appeared in Mkl and CuBlas, but we argue that a BatchedGemm with a constant stride between matrices is a common enough case to consider specializing for. By providing this interface, the common case with constant strides between matrices is not forced to perform allocations or precomputations as it currently must perform in order to use BatchedGemm. Additionally, these extra restrictions provide additional knowledge of the memory layout of the computation and offers additional optimizations opportunities in SIMDization, prefetching, and tiling. In other words, the "batch-loop" in StridedBatchedGemm now directly participates in the polyhedral computation as an affine for-loop. With the pointer-interface in BatchedGemm, the "batch-loop" cannot fully participate in a polyhedral model of the computation and is certainly not a candidate for vectorization or cache blocking.

In Table 5.2, we have enumerated all unique single-mode contractions between a second-order and third-order tensor using the notation from Section 5.4.2. All but 8 contractions can be computed with only a single call to StridedBatchedGemm.

Table 5.2: List of 36 possible single mode contraction operations between a second-order tensor and a third-order tensor and possible mappings to Level-3 Blas routines. Note that 8 cases may be performed with Gemm, 28 cases may be performed with Strided-BatchedGemm, and 8 cases remain exceptional

Case	Contraction	Kernel1	Kernel2	Kernel3
1.1	$A_{mk}B_{knp}$	$C_{m(np)} = A_{mk}B_{k(np)}$	$C_{mn[p]} = A_{mk}B_{kn[p]}$	$C_{m[n]p} = A_{mk}B_{k[n]p}$
1.2	$A_{mk}B_{kpn}$	$C_{mn[p]} = A_{mk}B_{k[p]n}$	$C_{m[n]p} = A_{mk}B_{kp[n]}$	
1.3	$A_{mk}B_{nkp}$	$C_{mn[p]} = A_{mk}B^\top_{nk[p]}$		
1.4	$A_{mk}B_{pkn}$	$C_{m[n]p} = A_{mk}B^\top_{pk[n]}$		
1.5	$A_{mk}B_{npk}$	$C_{m(np)} = A_{mk}B^\top_{(np)k}$	$C_{mn[p]} = A_{mk}B^\top_{n[p]k}$	
1.6	$A_{mk}B_{pnk}$	$C_{m[n]p} = A_{mk}B^\top_{p[n]k}$		
2.1	$A_{km}B_{knp}$	$C_{m(np)} = A^\top_{km}B_{k(np)}$	$C_{mn[p]} = A^\top_{km}B_{kn[p]}$	$C_{m[n]p} = A^\top_{km}B_{k[n]p}$
2.2	$A_{km}B_{kpn}$	$C_{mn[p]} = A^\top_{km}B_{k[p]n}$	$C_{m[n]p} = A^\top_{km}B_{kp[n]}$	
2.3	$A_{km}B_{nkp}$	$C_{mn[p]} = A^\top_{km}B^\top_{nk[p]}$		
2.4	$A_{km}B_{pkn}$	$C_{m[n]p} = A^\top_{km}B^\top_{pk[n]}$		
2.5	$A_{km}B_{npk}$	$C_{m(np)} = A^\top_{km}B^\top_{(np)k}$	$C_{mn[p]} = A^\top_{km}B^\top_{n[p]k}$	
2.6	$A_{km}B_{pnk}$	$C_{m[n]p} = A^\top_{km}B^\top_{p[n]k}$		
3.1	$A_{nk}B_{kmp}$	$C_{mn[p]} = B^\top_{km[p]}A^\top_{nk}$		
3.2	$A_{nk}B_{kpm}$	$C_{mn[p]} = B^\top_{k[p]m}A^\top_{nk}$		
3.3	$A_{nk}B_{mkp}$	$C_{mn[p]} = B_{mk[p]}A^\top_{nk}$		
3.4	$A_{nk}B_{pkm}$	$TRANS(A_{nk}B^\top_{pk[m]})$	$C_{[m][n]p} = B_{pk[m]}A_{[n]k}$	
3.5	$A_{nk}B_{mpk}$	$C_{mn[p]} = B_{m[p]k}A^\top_{nk}$		
3.6	$A_{nk}B_{pmk}$	$TRANS(A_{nk}B^\top_{p[m]k})$	$C_{[m][n]p} = B_{p[m]k}A_{[n]k}$	
4.1	$A_{kn}B_{kmp}$	$C_{mn[p]} = B^\top_{km[p]}A_{kn}$		
4.2	$A_{kn}B_{kpm}$	$C_{mn[p]} = B^\top_{k[p]m}A_{kn}$		
4.3	$A_{kn}B_{mkp}$	$C_{mn[p]} = B_{mk[p]}A_{kn}$		
4.4	$A_{kn}B_{pkm}$	$TRANS(A^\top_{kn}B^\top_{pk[m]})$	$C_{[m][n]p} = B_{pk[m]}A_{k[n]}$	
4.5	$A_{kn}B_{mpk}$	$C_{mn[p]} = B_{m[p]k}A_{kn}$		
4.6	$A_{kn}B_{pmk}$	$TRANS(A^\top_{kn}B^\top_{p[m]k})$	$C_{[m][n]p} = B_{p[m]k}A_{k[n]}$	
5.1	$A_{pk}B_{kmn}$	$C_{(mn)p} = B^\top_{k(mn)}A^\top_{pk}$	$C_{m[n]p} = B^\top_{km[n]}A^\top_{pk}$	
5.2	$A_{pk}B_{knm}$	$C_{m[n]p} = B^\top_{k[n]m}A^\top_{pk}$		
5.3	$A_{pk}B_{mkn}$	$C_{m[n]p} = B_{mk[n]}A^\top_{pk}$		
5.4	$A_{pk}B_{nkm}$	$TRANS(B_{nk[m]}A^\top_{pk})$	$C_{[m]n[p]} = B_{nk[m]}A_{[p]k}$	
5.5	$A_{pk}B_{mnk}$	$C_{(mn)p} = B_{(mn)k}A^\top_{pk}$	$C_{m[n]p} = B_{m[n]k}A^\top_{pk}$	
5.6	$A_{pk}B_{nmk}$	$TRANS(B_{n[m]k}A^\top_{pk})$	$C_{[m]n[p]} = B_{n[m]k}A_{[p]k}$	
6.1	$A_{kp}B_{kmn}$	$C_{(mn)p} = B^\top_{k(mn)}A_{kp}$	$C_{m[n]p} = B^\top_{km[n]}A_{kp}$	
6.2	$A_{kp}B_{knm}$	$C_{m[n]p} = B^\top_{k[n]m}A_{kp}$		
6.3	$A_{kp}B_{mkn}$	$C_{m[n]p} = B_{mk[n]}A_{kp}$		
6.4	$A_{kp}B_{nkm}$	$TRANS(B_{nk[m]}A_{kp})$	$C_{[m]n[p]} = B_{nk[m]}A_{k[p]}$	
6.5	$A_{kp}B_{mnk}$	$C_{(mn)p} = B_{(mn)k}A_{kp}$	$C_{m[n]p} = B_{m[n]k}A_{kp}$	
6.6	$A_{kp}B_{nmk}$	$TRANS(B_{n[m]k}A_{kp})$	$C_{[m]n[p]} = B_{n[m]k}A_{k[p]}$	

Listing 5.1: Interface and reference implementation of BLAS-like strided batched Gemm.

```
// C_p = alpha*opA(A_p)*opB(B_p) + beta*C_p
sb_gemm(op_type opA, op_type opB,
        int m, int n, int k,
        T alpha,
        const T* A, int lda, int loa,
        const T* B, int ldb, int lob,
        T beta,
        T* C, int ldc, int loc,
        int batch_size)
{
  // EXPOSITION ONLY
  for (int p = 0; p < batch_size; ++p)
    gemm(opA, opB,
         m, n, k,
         alpha,
         A + p*loa, lda,
         B + p*lob, ldb,
         beta,
         C + p*loc, ldc);
}
```

Listing 5.2: Nested batching.

```
for (int q = 0; q < Q; ++q)
  sb_gemm(OP_N, OP_T,
          M, N, K,
          1,
          A, lda<1>, lda<2>,
          B+q*ldb<2>, ldb<1>, 0,
          0,
          C+q*ldc<3>, ldc<1>, ldc<2>,
          P);
```

5.4.5 Exceptional Cases

The eight exceptional cases in Table 5.2 – Cases 3.4, 3.6, 4.4, 4.6, 5.4, 5.6, 6.4, and 6.6 – occur when batching forces the evaluation to either be a BatchedGemv or violate the no-first-mode rule.

This can be resolved by making an extension to the operation parameters allowed for BatchedGemm. Typically, the available operation parameters are "normal", "transpose", "conjugate", and "Hermitian". To account for the exceptional cases, "extended X" could be added to allow violations of the no-first-mode rule and consider all three modes involved in the batching simultaneously.

For example, Case 3.6 and 6.4 could then be written as $C_{mn[p]} = B_{[p]mk} \, A^\top_{nk}$ and $C_{m[n]p} = B^\top_{[n]km} \, A_{kp}$, and evaluated via

```
sb_gemm(OP_EX_N, OP_T,
        M, N, K,
        1,
        B, ldb<1>, ldb<2>,
        A, lda<1>, 0,
        0,
        C, ldc<1>, ldc<2>,
        P);
```

```
sb_gemm(OP_EX_T, OP_N,
        M, P, K,
        1,
        B, ldb<1>, ldb<2>,
        A, lda<1>, 0,
        0,
        C, ldc<2>, ldc<1>,
        N);
```

When the extended operation is passed, it is known that batching is in the first mode of the input which always has leading dimension 1. Thus, the leading order parameter to `sb_gemm` contains no information. Instead, leading dimensions of the other two modes in row-column order of the batched matrix are passed as the leading dimension and leading order parameters.

The implementation of a computation like this is expected to perform a "3D" tiling of B into cache in order to efficiently contract with the standard 2D cache tiling of A.

5.4.6 Generalization

In this section, we explain the generality of our approach and how it can be easily applied and extended to single-mode contractions involving tensors of arbitrary order.

Consider an arbitrary single-mode tensor contraction of the form (5.1). It is straightforward to see by simple counting that the number of unique contractions is $[(|\mathcal{A}|+|\mathcal{B}|-2)!]\cdot|\mathcal{A}|\cdot|\mathcal{B}|$. We note that Table 5.2 is obtained with $|\mathcal{A}|=2$ and $|\mathcal{B}|=3$. Of these contractions, all of them may be performed without explicit mode transpositions by nesting the BatchedGemm operations.

We observe that some single-mode contractions of two tensors of arbitrary order can be evaluated by batching on different modes with the BatchedGemm operations. For example, consider $C_{mn[p][q]} = A_{mk[p]}B_{nk[q]}$ wherein we can batch in either p and q. We prefer to choose the mode with the larger dimension for the BatchedGemm batching loop over the other (nested batching).

The nested-batching strategy in Listing 5.2 is general and extends to any two tensors of any order. Algorithms and heuristics for choosing the looped, batched, and Gemm-ed modes are provided in Section 5.5.4.

5.5 Results and Discussion

In this section, we benchmark varying evaluation strategies in order to define heuristics for computing general tensor contractions without copy or transposition. Additionally, we demonstrate the feasibility of the extended transpose parameter for exceptional case evaluations.

All performance measurements are performed on a heterogeneous CPU-GPU system with a dual-socket Intel Xeon E5-2630 v3 2.4GHz processor and an NVIDIA K40c GPU. Each CPU socket has 8 cores and 16 threads with an 8×256KB L2 cache and a 20MB L3 cache. The K40c has 2880 streaming cores distributed across 15 multiprocessors operating at 0.75GHz and a 1.5MB L2 cache.

All data used are randomized dense matrices. To eliminate noise from parallel competition of multi-sockets, all CPU results are generated from serial runs (one core, one thread).

5.5.1 Conventional Evaluation

We further motivate the use of StridedBatchedGemm evaluations by plotting the speedup of the conventional approach – transpositions until a single Gemm can be called – over a

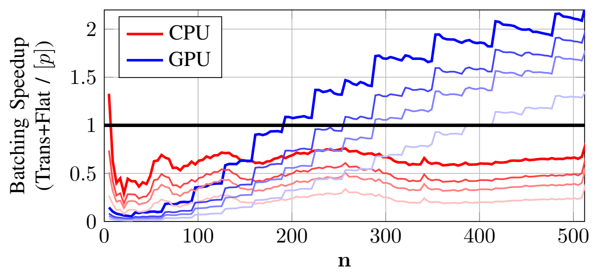


Figure 5.3: Performance ratio between the conventional approach with κ mode transpositions over a BatchedGemm in $[p]$ for Case 1.3. For color from deep to light, $\kappa = 1, 2, 3, 6$. Performance on CPU using Mkl's `mkl_somatcopy`, `cblas_sgemm`, and `cblas_sgemm_batch`. Performance on GPU using CuBlas's `cublasSgeam`, `cublasSgemm`, and our modified `cublasSgemmBatched`.

single StridedBatchedGemm call in evaluation of Case 1.3 from Table 5.2 for tensors of size $\mathbf{n} \times \mathbf{n} \times \mathbf{n}$. Figure 5.3 shows that StridedBatchedGemm is significantly faster than performing even a single transposition followed by a flattened Gemm, especially for small matrices. Here, a single transposition means $\mathbf{n}$ calls to `mkl_somatcopy` on CPU or `cublasSgeam` on GPU in order to fully exchange two modes. The dark lines include only a single transposition and the lighter lines include 2, 3, and 6 transpositions.

On CPU, the StridedBatchedGemm evaluation outperforms the conventional approach for all $\mathbf{n} < 512$. On GPU, the benefit from performing a single flattened Gemm eventually outweighs the cost of performing the transposition and for $\mathbf{n} \gtrsim 200$ the conventional approach achieves a speedup over the StridedBatchedGemm. This speaks to the highly optimized Gemm in CuBlas and that, perhaps, additional optimization gains from CuBlas's BatchedGemm may be available.

5.5.2 Extended BLAS Evaluation

In this section, we compare evaluation strategies given the extended Blas kernels. On GPU, the StridedBatchedGemm interface is provided by modifying `cublasSgemmBatched` from CuBlas 7.5. On CPU, the StridedBatchedGemm interface is implemented in serial with looped calls to `cblas_sgemm` from Mkl 11.2. Both implementations thereby avoid additional allocation and/or precomputation at the call site. The serial execution on CPU emphasizes the cache effects discussed in the following sections.

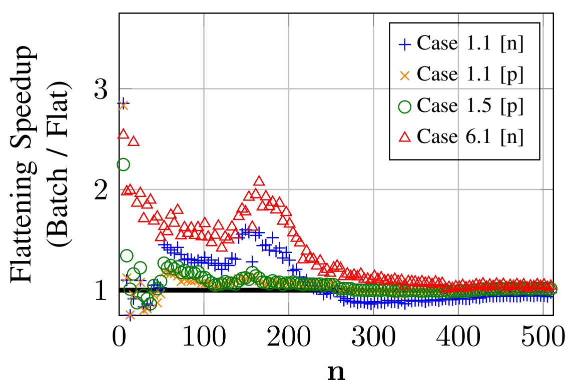
Flattening Cases 1.1, 1.5, and 6.1 can be evaluated without explicit transpositions with either a single flattened Gemm or a single BatchedGemm. We expect the flattened Gemm evaluation to outperform the BatchedGemm evaluation due to the optimization level of existing Gemms over that of the recently emerging BatchedGemm functions.

In Figure 5.4, we plot the speedup achieved by using a flattened Gemm evaluation over a StridedBatchedGemm evaluation. In Figure 5.4, the speedup is greater than one when FlattenedGEMM is faster than the StridedBatchedGemm. Clearly, most of the time, flattened Gemm is faster. Furthermore, we note the CuBlas implementation of StridedBatchedGemm is a great candidate for optimization as it appears to be significantly underperforming with respect to Gemm.

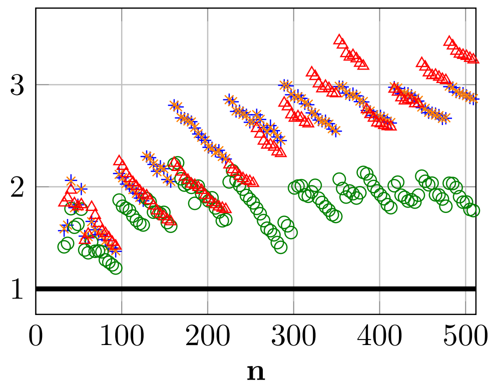
We also note the dependence of the performance on the shape of the flattened Gemm and the mode of the StridedBatchedGemm. On CPU, we find that the major determining factor in performance is the batching mode of the output. That is, the StridedBatchedGemm evaluation performs best when batched in the third mode of C – in Case 1.5 $[p]$ and 1.1 $[p]$. On GPU, the output batching mode makes no difference. It is unclear why the batched evaluation performs so well on Case 1.5 $[p]$.

Batching In this section, we attempt to quantify the performance gain by batching in the last mode versus an earlier mode and whether the input tensor or the output tensor should be prioritized for this optimization.

Case 1.1 and 2.1 can both be batched in the second ($[n]$) or third ($[p]$) mode. In Figure 5.5,



(a) CPU



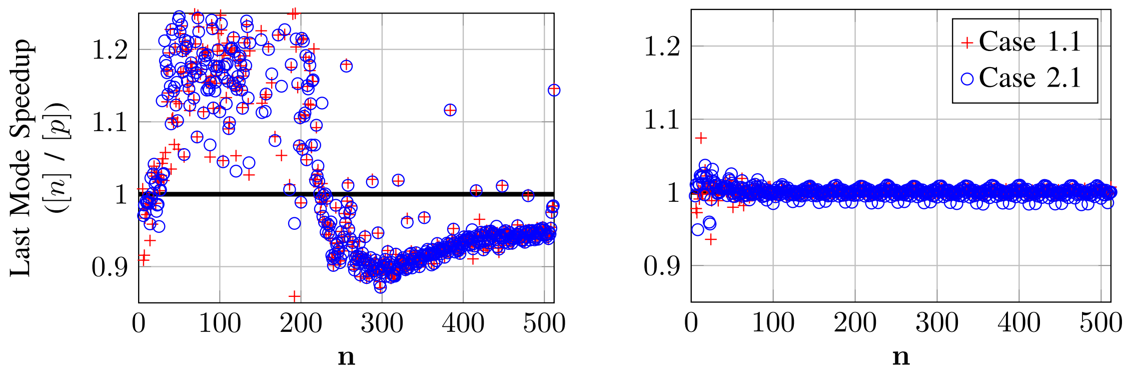
(b) GPU

Figure 5.4: Performance ratio for a BatchedGemm over a flattened Gemm in evaluation of Cases 1.1, 1.5, and 6.1.

we plot the speedup in performing the BatchedGemm in $[p]$ over performing it in $[n]$. When the size of the tensor is small, $\mathbf{n} \lesssim 256$, batching in the third mode is advantageous and can result in up to 1.25x speedup on CPU. When $\mathbf{n} \gtrsim 256$, it is approximately 1.1x faster to batch in the second mode rather than the third. We expect this is an effect of the 256KB L1 cache, which would house the contiguous B_{kn} submatrix for each p when $\mathbf{n} \lesssim 256$. Beyond that size both batching strategies will have forced cache misses within each Gemm, but by batching in the middle mode more data is shared between individual Gemms.

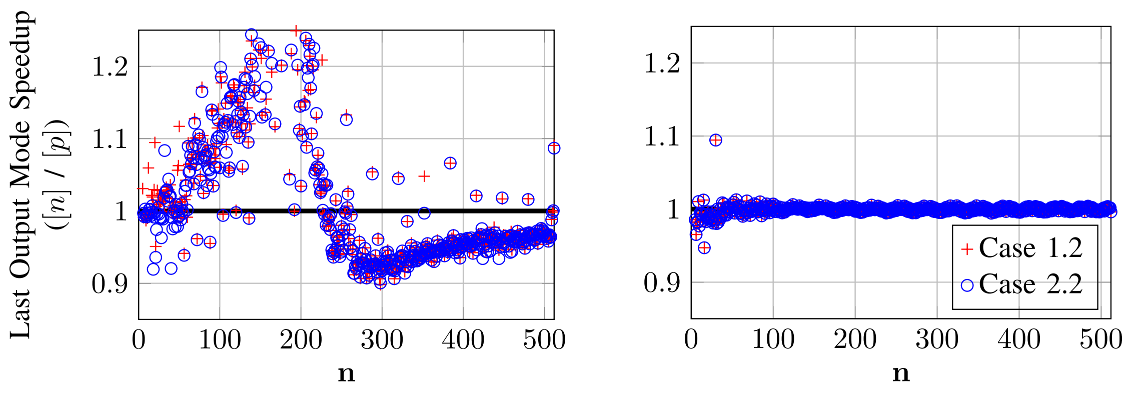
On GPU, we see no discernible preference in the choice of batching mode. The GPU has a much less sophisticated memory system with no prefetcher and the performance difference is primarily determined by the number of global memory transactions issued. When $\mathbf{n} \geq 32$, the coalescing width is reached so nearly the same number of transactions will be issued in each case – with small differences caused by alignment. We confirmed this by profiling the number of global memory reads and writes issued by each kernel and verifying that they correlate with the small differences in performance observed.

Additionally, we consider the mixed-mode batching evaluations to determine if the input or output array is the primary determination of batching performance. In Figure 5.6, we plot the speedup in performing StridedBatchedGemm in the last mode of the output



(a) CPU (b) GPU

Figure 5.5: Speedup obtained from batching in the last mode, $[p]$, rather than the middle mode, $[n]$, for Cases 1.1 and 2.1.



(a) CPU (b) GPU

Figure 5.6: Speedup obtained from batching in the last output mode, $[p]$, rather than the middle output mode, $[n]$, for Cases 1.2 and 2.2.

but the middle mode of the input, $[p]$, over performing it in the middle mode of the output and the last mode of the input, $[n]$, for Cases 1.2 and 2.2. The results are very similar to those of Figure 5.5 indicating that batching mode of the output tensor C is more important than the batching mode of the input tensor B on CPU. This is consistent with reference implementations of Gemm which accumulate results directly into the output matrix.

Exceptional Cases In this section, we demonstrate the feasibility of evaluation strategies for the exceptional cases.

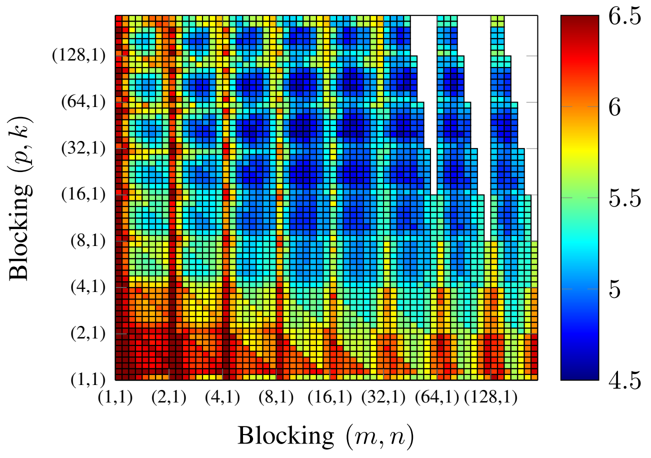


Figure 5.7: GPU tiling parameter profile from PPCG on K40c for Case 6.4. Performance values are $\log_{10}([\mu sec])$ and tests performed for $\mathbf{m} = \mathbf{n} = \mathbf{k} = \mathbf{p} = 256$. White indicates the run failed.

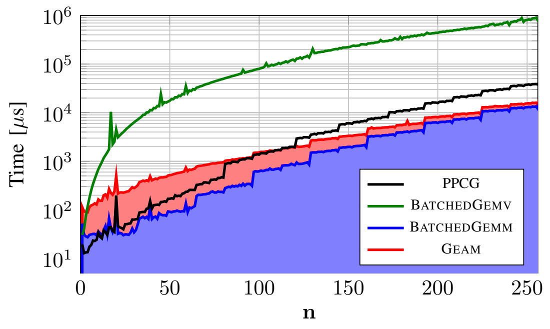


Figure 5.8: Benchmark of three evaluation strategies for Case 6.4: A BatchedGemv, a mode transposition followed by a BatchedGemm, and an extended transpose kernel generated by PPCG.

The Polyhedral Parallel Code Generator (PPCG) [101] is a source-to-source compiler capable of generating CUDA kernels from nested control loops in C. We use PPCG to generate a CUDA kernel for exceptional Case 6.4 and compare its performance against other evaluation strategies.

First, Case 6.4 has four nested loops and PPCG accepts a tiling parameter for each. We search the parameter space $(m, n, p, k) \in [1, 2, 4, 8, 16, 32, 64, 128]^4$ for the most efficient variant in Figure 5.7. The kernels were generated with $\alpha = 1$ and $\beta = 0$ statically known as generated versions with dynamic α, β had significant branching and divergent overhead, whereas we are primarily interested in the access patterns and tiling.

The tiling parameters that result in the highest performance are $(16, 4, 32, 4)$. Via inspection, we verify that the generated kernel is performing a 2D shared memory tiling for A, a "3D" shared memory tiling for B, and accumulating the C results in registers.

Using the $(16, 4, 32, 4)$ kernel, we benchmark against two possible evaluation strategies: (1) A BatchedGemv which requires no explicit transposition, and (2) A mode transposition in k and m followed by a BatchedGemm in $[n]$. In Figure 5.8, we show the execution time for each with the explicit transposition/Gemm stacked to show their relative proportion in the two-step evaluation. The PPCG kernel outperforms the explicit transposition/Gemm evaluation for small matrices and remains within a factor of 2-3x as $\mathbf{n}$ grows. We expect an expert implementation of the extended transpose parameter kernel would be able to close this gap and remain competitive with BatchedGemm for all $\mathbf{n}$.

5.5.3 Machine Learning Application

In this section, we present the benchmarking results for the application that we discussed in Section 5.3. For simulations on the CPU, we compare the performance on the Tucker decomposition using TensorToolbox, Btas, Cyclops and our StridedBatchedGemm. For simulations on the GPU, we don't have available GPU library to compare with, so we just evaluate our GPU implementation against StridedBatchedGemm. We fix the number of iterations as $\mathcal{T} = 200$, set the core tensor size as $\mathbf{i} = \mathbf{j} = \mathbf{k} = 10$, and set the dimensions as $\mathbf{m} = \mathbf{n} = \mathbf{p}$. From Figure 5.9, using our CPU StridedBatchedGemm, we obtain more than 10 times speedup compared to Cyclops/TensorToolbox and almost four orders of magnitude compared to Btas. Also, as expected, our GPU StridedBatchedGemm confers further speedup.

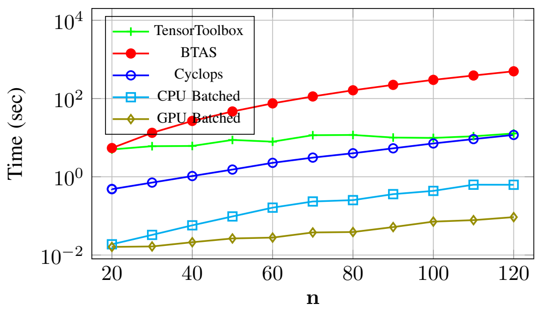


Figure 5.9: Performance on Tucker decompostion.

5.5.4 Evaluation Priorities

Rather than attempt to model the algorithm and machine as in [81, 73], we simply provide evaluation guidelines based on the data provided. These are a number of heuristics that may be important in constructing the most efficient evaluation strategy.

1. Flatten modes whenever possible. A single large GEMM is more efficient.
2. In the interest of performing the highest intensity computation within a BATCHEDGEMM, we recommend performing the largest GEMMs possible within a BATCHEDGEMM and batching in the mode with largest dimension.
3. Preferring to batch in the last mode versus earlier modes can depend on the input parameters and machine.

We summarize these evaluation guidelines with pseudocode for performing a single-index tensor contraction without copy or transposition in Algorithm 4.

5.6 Conclusion

Our experience reveals that the emergence of BATCHEDGEMM provides significant computational advantages for multi-linear algebraic computations. The primitive allows us to push a larger high intensity computations to vendor-provided implementations. Leading

Algorithm 4 Single-mode Tensor Contraction

```
1: **In:** Tensor $A_{\mathcal{A}}$, $\mathcal{A} = [a_1, \ldots, a_M]$,
2: **In:** Tensor $B_{\mathcal{B}}$, $\mathcal{B} = [b_1, \ldots, b_N]$, $\mathcal{A} \cap \mathcal{B} = \{k\}$.
3: **In, Out:** Tensor $C_{\mathcal{C}}$, $\mathcal{C} = [c_1, \ldots, c_{N+M-2}]$. WLOG, $c_1 \in \mathcal{A}$.
4: Common substrings in $\mathcal{A}$, $\mathcal{B}$ and/or $\mathcal{C}$ for flattening candidates.
5: Relabel flattened modes
6: Compute $\mathcal{P} = \{c_i \mid i \neq 1, c_i \not\equiv a_1, c_i \not\equiv b_1\}$
7: **if** $|\mathcal{C} \setminus \mathcal{P}| = |\{c_1\}| = 1.$ **then**
8:     [Case $C_{c_1 \cdots} = A_{k \cdots c_1 \cdots} B_{k \cdots}$]
9:     Let $c^* \in \mathcal{P} \setminus \mathcal{A}$ be index with max dimension
10:    Let $c^+ \in \mathcal{P} \setminus \{c_1, c^*\}$ be index with max dimension
11:    Nested in all $c_j \in P \setminus \{c^*, c^+\}$, BatchedGemm in $c_1, c^*, k, [c^+]$
12: **else if** $|\mathcal{C} \setminus \mathcal{P}| = |\{c_1, c_b\}| = 2$ **then**
13:    [Case $C_{c_1 \cdots c_b \cdots} = A_{k \cdots c_1 \cdots} B_{c_b \cdots k \cdots}$]
14:    Let $c^* \in \mathcal{P}$ be index with max dimension
15:    Nested in all $c_j \in P \setminus \{c^*\}$, BatchedGemm in $c_1, c_b, k, [c^*]$
16: **else if** $|\mathcal{C} \setminus \mathcal{P}| = |\{c_1, c_a\}| = 2$ **then**
17:    [Case $C_{c_1 \cdots c_a \cdots} = A_{c_a \cdots c_1 \cdots k \cdots} B_{k \cdots}$]
18:    Let $c^* \in \mathcal{P} \setminus \mathcal{A}$ be index with max dimension
19:    Nested in all $c_j \in P \setminus \{c^*\}$, Ex. BatchedGemm in $c_1, c^*, k, [c_a]$
20: **else if** $|\mathcal{C} \setminus \mathcal{P}| = |\{c_1, c_a, c_b\}| = 3$ **then**
21:    [Case $C_{c_1 \cdots c_a \cdots c_b \cdots} = A_{c_a \cdots c_1 \cdots k \cdots} B_{c_b \cdots k \cdots}$]
22:    Nested in all $c_j \in P$, Ex. BatchedGemm in $c_1, c_b, k, [c_a]$
```

implementations already provide BatchedGemm on highly parallel machines. To simplify their use and provide additional optimization opportunities, we propose StridedBatchedGemm and demonstrate its use for generalized tensor contractions. Calls to StridedBatchedGemm have significant opportunity to perform at or near the performance of Gemm and, by avoiding explicit transpositions or permutations of the data, accelerate these computations significantly.

Our improvement is most significant on small and moderate sized tensors. This is very important because in many applications, e.g. deep learning for training a recursive tensor network, we require evaluating a large number of tensor contractions of small sizes.

Although we focused on single-node performance, these evaluations may be used as building blocks for distributed memory implementations, which we intent to pursue as part of our future work. Further study into the optimized implementations, architecture-dependent implementations, and performance of the exceptional case kernels is warranted. More complicated contractions, such as multi-index contractions or sparse tensor algebra, also pose challenging problems.

Chapter 6

Conclusion and Outlook

6.1 Conclusion

In the dissertation, we discuss about how tensor is involved in machine learning and specifically how we can efficiently compute tensor operations in different machine learning applications. Tensor algebra, as an extension from vector and matrix algebra, provides more flexible data operations. These operations build up better model representations. On one hand, tensor product has been applied to multi-modality feature learning. On the other hand, the tensor decomposition analysis, which is one important way in learning hidden structure from large data, requires tensor contraction operations frequently.

We propose a dimensionality reduction method: higher-order count sketch. We show that this operation retains efficient tensor product and tensor contraction by directly performing the operations on the sketched components. We also demonstrate efficient tensor contraction primitives that maximal utilize the parallel scheme. We apply both methods in different machine learning tasks, such as video denoising and visual question answering.

6.2 Outlook

There are still so many unsolved problems in the machine learning field. Some of the topics that I am interested in are: how to represent large-scale neural networks as sequential tensor operations and how to apply tensor compression techniques to use less computation resources while preserving performance. More generally, I am curious about what is the optimal solution for different computations under specific conditions. For example, with limited memory and bandwidth , how can we integrate efficient tensor operations with the computing, learning and inference processes on edge devices. This relies on both software (algorithm) and hardware optimization.

Bibliography

[1] G. B. A. Allam, J. Ramanujam and P. Sadayappan. Memory minimization for tensor contractions using integer linear programming. *IPDPS'06*, 2006.

[2] M. Abadi et al. TensorFlow: Large-scale machine learning on heterogeneous systems. Software available on tensorflow.org.

[3] B. M. D. V. D. J. E. C. F. J. H. A. K. I. K. T. M. I. T. S. Abdelfattah, A. High-performance tensor contractions for gpus. *ICCS'16*.

[4] A. Agrawal, D. Batra, and D. Parikh. Analyzing the behavior of visual question answering models. *EMNLP 2016*, 2016.

[5] N. Alon, Y. Matias, and M. Szegedy. The space complexity of approximating the frequency moments. *Journal of Computer and System Sciences*, 58(1):137–147, February 1999.

[6] R. R. Amossen and R. Pagh. A new data layout for set intersection on gpus. In *Proceedings of the 2011 IEEE International Parallel & Distributed Processing Symposium*, IPDPS '11, pages 698–708, Washington, DC, USA, 2011. IEEE Computer Society.

[7] A. Anandkumar, R. Ge, D. Hsu, S. Kakade, and M. Telgarsky. Tensor decompositions for learning latent variable models. *JMRL*, 15(1):2773–2832, 2014.

[8] A. Anandkumar, R. Ge, D. Hsu, S. M. Kakade, and M. Telgarsky. Tensor decompositions for learning latent variable models. *J. of Machine Learning Research*, 15:2773–2832, 2014.

[9] A. Anandkumar, R. Ge, and M. Janzamin. Guaranteed non-orthogonal tensor decomposition via alternating rank-1 updates. *ArXiv 1402.5180*, 2014.

[10] P. Anderson, X. He, C. Buehler, D. Teney, M. Johnson, S. Gould, and L. Zhang. Bottom-up and top-down attention for image captioning and VQA. *arXiv:1707.07998*, 2017.

[11] S. Antol, A. Agrawal, J. Lu, M. Mitchell, D. Batra, C. L. Zitnick, and D. Parikh. VQA: Visual Question Answering. In *International Conference on Computer Vision (ICCV)*, 2015.

[12] B. W. Bader and T. G. Kolda. Algorithm 862: MATLAB tensor classes for fast algorithm prototyping. *ACM Trans. Math. Softw.*, 32(4):635–653, 2006.

[13] B. W. Bader, T. G. Kolda, et al. Matlab tensor toolbox version 2.6. Available online, `http://www.sandia.gov/~tgkolda/TensorToolbox/`, February 2015.

[14] B. Barak and A. Moitra. Tensor Prediction, Rademacher Complexity and Random 3-XOR. *ArXiv e-prints 1501.06521*, Jan. 2015.

[15] K. Bringmann and K. Panagiotou. Efficient sampling methods for discrete distributions. *Algorithmica*, 79(2):484–508, Oct 2017.

[16] R. Bro and H. A. Kiers. A new efficient method for determining the number of components in parafac models. *Journal of chemometrics*, 17(5):274–286, 2003.

[17] A. Z. Broder. On the resemblance and containment of documents. *IEEE:Compression and Complexity of Sequences: Proceedings, Positano, Amalfitan Coast, Salerno, Italy,*, 10:21–29, 1997.

[18] A. Buluç and J. R. Gilbert. The combinatorial blas:design, implementation, and applications. *IJHPCA'11.*

[19] C. F. Caiafa and A. Cichocki. Generalizing the column–row matrix decomposition to multi-way arrays. *Linear Algebra and its Applications*, 433:557–573, Sept 2010.

[20] E. J. Candès, X. Li, Y. Ma, and J. Wright. Robust principal component analysis? *Journal of the ACM (JACM)*, 58(3):11, 2011.

[21] D. Cartwright and B. Sturmfels. The number of eigenvalues of a tensor. *Linear algebra and its applications*, 438(2):942–952, 2013.

[22] V. Chandrasekaran, S. Sanghavi, P. A. Parrilo, and A. S. Willsky. Rank-sparsity incoherence for matrix decomposition. *SIAM Journal on Optimization*, 21(2):572–596, 2011.

[23] K.-C. Chang, K. Pearson, T. Zhang, et al. Perron-frobenius theorem for nonnegative tensors. *Commun. Math. Sci*, 6(2):507–520, 2008.

[24] M. Charikar, K. Chen, and M. Farach-Colton. Finding frequent items in data streams. In *Proceedings of ICALP'02*, pages 693–703, 2002.

[25] M. Charikar, K. Chen, and M. Farach-Colton. Finding frequent items in data streams. *In Proceedings of ICALP'02*, 2002.

[26] M. S. Charikar. Similarity estimation techniques from rounding algorithms. In *Proceedings of the Thiry-fourth Annual ACM Symposium on Theory of Computing*, STOC '02, pages 380–388, New York, NY, USA, 2002. ACM.

[27] T. Chen, M. Li, Y. Li, M. Lin, N. Wang, M. Wang, T. Xiao, B. Xu, C. Zhang, and Z. Zhang. Mxnet: A flexible and efficient machine learning library for heterogeneous distributed systems. *Neural Information Processing Systems, Workshop on Machine Learning Systems 2015*, 2015.

[28] T. Christiani, R. Pagh, and J. Sivertsen. Scalable and robust set similarity join. *The annual IEEE International Conference on Data Engineering*, 2018.

[29] G. Cormode and S. Muthukrishnan. What's hot and what's not: Tracking most frequent items dynamically. *ACM Trans. Database Syst.*, 30(1):249–278, Mar. 2005.

[30] E. D. Demaine, A. López-Ortiz, and J. I. Munro. Frequency estimation of internet packet streams with limited space. In *Proceedings of the 10th Annual European Symposium on Algorithms*, ESA '02, pages 348–360, London, UK, UK, 2002. Springer-Verlag.

[31] J. Donahue, Y. Jia, O. Vinyals, J. Hoffman, N. Zhang, E. Tzeng, and T. Darrell. Decaf: A deep convolutional activation feature for generic visual recognition. *arXiv:1310.1531*, 2013.

[32] C. Eckart and G. Young. The approximation of one matrix by another of lower rank. In *Psychometrika*. Springer-Verlag, 1936.

[33] A. Fukui, D. H. Park, D. Yang, A. Rohrbach, T. Darrell, and M. Rohrbach. Multimodal compact bilinear pooling for visual question answering and visual grounding. *EMNLP 2016*, 2016.

[34] S. Gandy, B. Recht, and I. Yamada. Tensor completion and low-n-rank tensor recovery via convex optimization. *Inverse Problems*, 27(2):025010, 2011.

[35] Y. Gao, O. Beijbom, N. Zhang, and T. Darrell. Compact bilinear pooling. *Computer Vision and Pattern Recognition (CVPR), 2016*, 2016.

[36] D. Goldfarb and Z. Qin. Robust low-rank tensor recovery: Models and algorithms. *SIAM Journal on Matrix Analysis and Applications*, 35(1):225–253, 2014.

[37] N. Goyal, S. Vempala, and Y. Xiao. Fourier pca and robust tensor decomposition. In *Proceedings of the 46th Annual ACM Symposium on Theory of Computing*, pages 584–593.

[38] Y. Goyal, T. Khot, D. Summers-Stay, D. Batra, and D. Parikh. Making the V in VQA matter: Elevating the role of image understanding in Visual Question Answering. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017.

[39] Q. Gu, H. Gui, and J. Han. Robust tensor decomposition with gross corruption. In Z. Ghahramani, M. Welling, C. Cortes, N. Lawrence, and K. Weinberger, editors, *Advances in Neural Information Processing Systems 27*, pages 1422–1430. Curran Associates, Inc., 2014.

[40] Q. Gu, H. Gui, and J. Han. Robust tensor decomposition with gross corruption. In *Advances in Neural Information Processing Systems*, pages 1422–1430, 2014.

[41] D. Gurari, Q. Li, A. J. Stangl, A. Guo, C. Lin, K. Grauman, J. Luo, and J. P. Bigham. Vizwiz grand challenge: Answering visual questions from blind people. *arXiv:1802.08218*, 2018.

[42] R. Harshman. Foundations of the parafac procedure: Models and conditions for an explanatory multi-model factor analysis. *UCLA Working Papers in Phonetics*, 16:1–84, 1970.

[43] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. *In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR).*

[44] C. J. Hillar and L.-H. Lim. Most tensor problems are np-hard. *Journal of the ACM (JACM)*, 60(6):45, 2013.

[45] F. L. Hitchcock. The expression of a tensor or a polyadic as a sum of products. *J. of Math.and Physics*, 6(1):164–189, 1927.

[46] S. Hochreiter and J. Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[47] D. Hsu, S. M. Kakade, and T. Zhang. Robust matrix decomposition with sparse corruptions. *Information Theory, IEEE Transactions on*, 57(11):7221–7234, 2011.

[48] B. Huang, C. Mu, D. Goldfarb, and J. Wright. Provable low-rank tensor recovery. *Preprint*, 2014.

[49] H. Huang and C. Ding. Robust tensor factorization using r_1 norm. In *Computer Vision and Pattern Recognition, 2008. CVPR 2008. IEEE Conference on*, pages 1–8. IEEE, 2008.

[50] C. Jhurani and P. Mullowney. A gemm interface and implementation on nvidia gpus for multiple small matrices. *J. of Parallel and Distributed Computing*, pages 133–140, 2015.

[51] C. Jhurani and P. Mullowney. A GEMM interface and implementation on NVIDIA GPUs for multiple small matrices. *J. of Parallel and Distributed Computing*, 75:133–140, 2015.

[52] J. Johnson, B. Hariharan, L. van der Maaten, L. Fei-Fei, C. L. Zitnick, and R. B. Girshick. CLEVR: A diagnostic dataset for compositional language and elementary visual reasoning. *arXiv:1612.06890*, 2016.

[53] K. Kafle and C. Kanan. An analysis of visual question answering algorithms. In *ICCV*, 2017.

[54] V. Kazeev, M. H. Khammash, M. Nip, and C. Schwab. *Direct solution of the chemical master equation using quantized tensor trains.* ETH-Zürich, 2013.

[55] V. Khoromskaia and B. N. Khoromskij. Tensor numerical methods in quantum chemistry: from hartree–fock to excitation energies. *Physical Chemistry Chemical Physics*, 2015.

[56] T. G. Kolda and J. R. Mayo. Shifted power method for computing tensor eigenpairs. *SIAM J. Matrix Analysis Applications*, 32(4):1095–1124, 2011.

[57] T. G. Kolda and J. Sun. Scalable tensor decompositions for multi-aspect data mining. *ICDM*, 2008.

[58] J. Kossaifi, Z. C. Lipton, A. Khanna, T. Furlanello, and A. Anandkumar. Tensor regression networks. 2017.

[59] N. Kreimer, A. Stanton, and M. D. Sacchi. Tensor completion based on nuclear norm minimization for 5d seismic data reconstruction. *Geophysics*, 78(6):V273–V284, 2013.

[60] R. Krishna, Y. Zhu, O. Groth, J. Johnson, K. Hata, J. Kravitz, S. Chen, Y. Kalantidis, L. Li, D. A. Shamma, M. S. Bernstein, and F. Li. Visual genome: Connecting language and vision using crowdsourced dense image annotations. *arXiv:1602.07332*, 2016.

[61] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 25*, pages 1097–1105. Curran Associates, Inc., 2012.

[62] L. Lathauwer, B. Moor, and J. Vandewalle. On the best rank-1 and rank-(r_1,r2,...rn) approximation of higher-order tensors. *SIAM J. Matrix Anal. Appl.*, 21:1324–1342, 2000.

[63] J. Li, C. Battaglino, L. Perros, J. Sun, et al. An input-adaptive and in-place approach to dense tensor-times-matrix multiply. In *SC'15*, pages 76:1–76:12.

[64] L. Li, W. Huang, I. Y.-H. Gu, and Q. Tian. Statistical modeling of complex backgrounds for foreground object detection. *Image Processing, IEEE Transactions on*, 13(11):1459–1472, 2004.

[65] T. Lin, M. Maire, S. J. Belongie, L. D. Bourdev, R. B. Girshick, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C. L. Zitnick. Microsoft COCO: common objects in context. *In ECCV*, 2014.

[66] J. Lu, J. Yang, D. Batra, and D. Parikh. Hierarchical question-image co-attention for visual question answering. *In NIPS*, 2016.

[67] Q. Lu, X. Gao, et al. Empirical performance model-driven data layout optimization and library call selection for tensor contraction expressions. *J. Parallel Distrib. Comput.*, 72(3):338–352, Mar 2012.

[68] Y. Ma, J. Li, X. Wu, C. Yan, J. Sun, and R. Vuduc. Optimizing sparse tensor times matrix on gpus. *Journal of Parallel and Distributed Computingg*, pages 99–109, 2019.

[69] M. W. Mahoney and P. Drineas. Cur matrix decompositions for improved data analysis. *Proceedings of the National Academy of Sciences*, 106(3):697–702, 2009.

[70] O. A. Malik and S. Becker. Low-rank tucker decomposition of large tensors using tensorsketch. *Neural Information Processing Systems*, 2018.

[71] T. Mikolov, I. Sutskever, K. Chen, G. Corrado, and J. Dean. Distributed representations of words and phrases and their compositionality. *Advances in neural information processing systems*, pages 3111–3119, 2013.

[72] N. Nakatani, B. Verstichel, G. Chan, J. Calvin, et al. Btas v0.0.1. Available online, `https://github.com/BTAS/BTAS`.

[73] E. D. Napoli, D. Fabregat-Traver, G. Quintana-Ortí, and P. Bientinesi. Towards an efficient use of the BLAS library for multilinear tensor contractions. *AMC*, 235:454 – 468, 2014.

[74] T. Nelson, A. Rivera, P. Balaprakash, et al. Generating efficient tensor contractions for gpus. *ICPP'15*.

[75] P. Netrapalli, U. Niranjan, S. Sanghavi, A. Anandkumar, and P. Jain. Non-convex robust pca. In *Advances in Neural Information Processing Systems*, pages 1107–1115, 2014.

[76] H. Noh and B. Han. Training recurrent answering units with joint loss minimization for vqa. *arXiv:1606.03647*, 2016.

[77] R. Pagh. Compressed matrix multiplication. *ITCS*, 2012.

[78] R. Pagh and F. F. Rodler. Cuckoo hashing. *Lecture Notes in Computer Science*, 2001.

[79] E. Papalexakis and K. Pelechrinis. thoops: A multi-aspect analytical framework for spatio-temporal basketball data. *ACM CIKM*, 2018.

[80] E. Peise and P. Bientinesi. Performance modeling for dense linear algebra. In *SC/12: High Performance Computing, Networking Storage and Analysis.*

[81] E. Peise, D. Fabregat-Traver, and P. Bientinesi. On the performance prediction of BLAS-based tensor contractions. In *PMBS'15.*

[82] N. Pham and R. Pagh. Fast and scalable polynomial kernels via explicit feature maps. *KDD*, 2013.

[83] S. S. A. A. P. J. Praneeth Netrapalli, U N Niranjan. Non-convex robust pca. *Conference on Neural Information Processing Systems*, 2014.

[84] S. Ragnarsson and C. F. Van Loan. Block tensors and symmetric embeddings. *Linear Algebra and Its Applications*, 438(2):853–874, 2013.

[85] S. Ren, K. He, R. B. Girshick, and J. Sun. Faster R-CNN: towards real-time object detection with region proposal networks. *arXiv:1506.01497*, 2015.

[86] P. Shah, N. Rao, and G. Tang. Sparse and low-rank tensor decomposition. *Advances in Neural Information Processing Systems*, 2015.

[87] Y. Shi, U. Niranjan, A. Anandkumar, and C. Cecka. Tensor contractions with extended blas kernels on cpu and gpu. *HiPC*, 2016.

[88] M. Simon, E. Rodner, Y. Gao, T. Darrell, and J. Denzler. Generalized orderless pooling performs implicit salient matching. 2017.

[89] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv:1409.1556*, 2014.

[90] E. Solomonik, D. Matthews, J. R. Hammond, and J. Demmel. Cyclops tensor framework: reducing communication and eliminating load imbalance in massively parallel contractions. In *Parallel & Distributed Processing*, pages 813–824, 2013.

[91] P. Springer, T. Su, and P. Bientinesi. Hptt: A high-performance tensor transposition c++ library. *Proceedings of the 4th ACM SIGPLAN International Workshop on Libraries, Languages, and Compilers for Array Programming*, 2017.

[92] F. Strub, H. de Vries, J. Mary, B. Piot, A. C. Courville, and O. Pietquin. End-to-end optimization of goal-driven and visually grounded dialogue systems. In *International Joint Conference on Artificial Intelligence (IJCAI)*, 2017.

[93] I. Sutskever, O. Vinyals, and Q. V. Le. Sequence to sequence learning with neural networks. *arXiv:1409.3215*, 2014.

[94] G. K. Tamara and W. B. Brett. Tensor decompositions and applications. *SIAM Review*, 51(3):455–500, Mar 2009.

[95] J. B. Tenenbaum and W. T. Freeman. Separating style and content with bilinear models. *Neural computation*, 12(6):1247–1283, 2000.

[96] D. Teney, P. Anderson, X. He, and A. van den Hengel. Tips and tricks for visual question answering: Learnings from the 2017 challenge. *arXiv:1708.02711*, 2017.

[97] R. Tomioka, K. Hayashi, and H. Kashima. Estimation of low-rank tensors via convex optimization. *arXiv preprint arXiv:1010.0789*, 2010.

[98] L. R. Tucker. Some mathematical notes on three-mode factor analysis. *Psychometrika*, 31(3):279–311, 1966.

[99] F. G. Van Zee and R. A. van de Geijn. Blis: A framework for rapidly instantiating blas functionality. *ACM Trans. Math. Softw.*, 41(3):14:1–14:33, June 2015.

[100] M. A. O. Vasilescu and D. Terzopoulos. Multilinear analysis of image ensamble: Tensorfaces. *ECCV'02*.

[101] S. Verdoolaege, J. C. Juega, and A. o. Cohen. Polyhedral parallel code generation for cuda. *ACM Trans. Archit. Code Optim.*, 9(4):54:1–54:23, Jan. 2013.

[102] Y. Wang, H.-Y. Tung, A. Smola, and A. Anandkumar. Fast and guaranteed tensor decomposition via sketching. *Proceedings of Advances in Neural Information Processing Systems (NIPS)*, 2015.

[103] K. Q. Weinberger and L. K. Saul. Distance metric learning for large margin nearest neighbor classification. *Journal of Machine Learning Research*, 10:207–244, 2009.

[104] Y. Wu, M. Schuster, Z. Chen, Q. V. Le, M. Norouzi, W. Macherey, M. Krikun, Y. Cao, Q. Gao, K. Macherey, J. Klingner, A. Shah, M. Johnson, X. Liu, Lukasz Kaiser, S. Gouws, Y. Kato, T. Kudo, H. Kazawa, K. Stevens, G. Kurian, N. Patil, W. Wang, C. Young, J. Smith, J. Riesa, A. Rudnick, O. Vinyals, G. Corrado, M. Hughes, and J. Dean. Google's neural machine translation system: Bridging the gap between human and machine translation. *arXiv:1609.08144*, 2016.

[105] H. Xu and K. Saenko. Ask, attend and answer: Exploring question-guided spatial attention for visual question answering. *European Conference on Computer Vision 2016*, 2016.

[106] K. Xu, J. Ba, R. Kiros, A. Courville, R. Salakhutdinov, R. Zemel, and Y. Bengio. Show, attend and tell: Neural image caption generation with visual attention. *International Conference on Machine Learning*, 2015.

[107] B. Yang, A. Zamzam, and N. D. Sidiropoulos. Parasketch: Parallel tensor factorization via sketching. *SIAM International Conference on Data Mining*, 2018.

[108] Z. Yang, X. He, J. Gao, L. Deng, and A. Smola. Stacked attention networks for image question answering. *arXiv:1511.02274*, 2015.

[109] R. Yu, S. Zheng, A. Anandkumar, and Y. Yue. Long-term forecasting using tensor-train rnns.

[110] P. Zhang, Y. Goyal, D. Summers-Stay, D. Batra, and D. Parikh. Yin and Yang: Balancing and answering binary visual questions. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.

Appendix A

Appendix for Tensor Robusr Principle Component Analysis

A.1 Bounds for block sparse tensors

One of the main bounds to control is the spectral norm of the sparse perturbation tensor S. The success of the power iterations and the improvement in accuracy of recovery over iterative steps of RTD requires this bound.

Lemma A.1 (Spectral norm bounds for block sparse tensors)**.** *Let $M \in \mathbb{R}^{n\times n\times n}$ satisfy the block sparsity assumption (S). Then*

$$\|M\|_2 = O(d^{1.5}\|M\|_\infty). \tag{A.1}$$

Proof. Let $\Psi \in \mathbb{R}^{n\times n\times n}$ be a tensor that encodes the sparsity of M i.e. $\Psi_{i,j,k} = 1$ iff $S^*_{i,j,k} \neq 0$

for all $i, j, k \in [n]$. We have that

$$\begin{aligned}
\|M\| &= \max_{u:\|u\|=1} \sum_{i,j,k} M_{i,j,k} u(i)u(j)u(k) \\
&= \max_{u:\|u\|=1} \sum_{i,j,k} M_{i,j,k} \Psi_{i,j,k} u(i)u(j)u(k) \\
&\leq \max_{u:\|u\|=1} \sum_{i,j,k} |M_{i,j,k} \Psi_{i,j,k} u(i)u(j)u(k)| \\
&\leq \|M\|_\infty \max_{u:\|u\|=1} \sum_{i,j,k} |\Psi_{i,j,k} u(i)u(j)u(k)| = \|M\|_\infty \|\Psi\|,
\end{aligned}$$

where the last inequality is from Perron Frobenius theorem for non-negative tensors [23]. Note that Ψ is non-negative by definition. Now we bound $\|\Psi\|$ on lines of [9, Lemma 4]. Recall that $\forall i \in [B]$, $j \in [n]$,

$$\Psi = \sum_{i=1}^{B} \psi_i \otimes \psi_i \otimes \psi_i, \quad \|\psi_i\|_0 \leq d, \, \psi_i(j) = 0 \text{ or } 1.$$

By definition $\|\psi_i\|_2 = \sqrt{d}$. Define normalized vectors $\widetilde{\psi}_i := \psi_i / \|\psi_i\|$. We have

$$\Psi = d^{1.5} \sum_{i=1}^{B} \widetilde{\psi}_i \otimes \widetilde{\psi}_i \otimes \widetilde{\psi}_i$$

Define matrix $\widetilde{\psi} := [\widetilde{\psi_1} | \widetilde{\psi_2}, \ldots \widetilde{\psi_B}]$. Note that $\widetilde{\psi}^\top \widetilde{\psi} \in \mathbb{R}^{B \times B}$ is a matrix with unit diagonal entries and absolute values of off-diagonal entries bounded by η, by assumption. From Gershgorin Disk Theorem, every subset of L columns in $\widetilde{\psi}$ has singular values within $1 \pm o(1)$, where $L < \frac{1}{\eta}$. Moreover, from Gershgorin Disk Theorem, $\|\widetilde{\psi}\| < \sqrt{1 + B\eta}$.

For any unit vector u, let S be the set of L indices that are largest in $\widetilde{\psi}^\top u$. By the argument above we know $\|(\widetilde{\psi}_S)^\top u\| \leq \|\widetilde{\psi}_S\| \|u\| \leq 1 + o(1)$. In particular, the smallest entry in $\widetilde{\psi}_S^\top u$ is at most $2/\sqrt{L}$. By construction of S this implies for all i not in S, $|\widetilde{\psi}_i^\top u|$ is at most $2/\sqrt{L}$.

Now we can write the ℓ_3 norm of $\widetilde{\psi}^\top u$ as

$$\begin{aligned}
\|\widetilde{\psi}^\top u\|_3^3 &= \sum_{i \in S} |\widetilde{\psi}_i^\top u|^3 + \sum_{i \notin S} |\widetilde{\psi}_i^\top u|^3 \\
&\le \sum_{i \in S} |\widetilde{\psi}_i^\top u|^2 + (2/\sqrt{L})^{3-2} \sum_{i \notin S} |\widetilde{\psi}_i^\top u|^2 \\
&\le 1 + 2\sqrt{\eta}\|\widetilde{\psi}\|^2 \le 1 + 2B\eta^{1.5}.
\end{aligned}$$

Here the first inequality uses that every entry outside S is small, and last inequality uses the bound argued on $\|(\widetilde{\psi}_S)^\top u\|$, the spectral norm bound is assumed on A_{S^c}. Since $B = O(\eta^{-1.5})$, we have the result. ■

Another important bound required is ∞-norm of certain contractions of the (normalized) sparse tensor and its powers, which we denote by M below. We use a loose bound based on spectral norm and we require $\|M\| < 1/\sqrt{n}$. However, this constraint will also be needed for the power iterations to succeed and is not an additional requirement. Thus, the loose bound below will suffice for our results to hold.

Lemma A.2 (Infinity norm bounds). *Let $M \in \mathbb{R}^{n \times n \times n}$ satisfy the block sparsity assumption (S). Let u, v satisfy the assumption $\|u\|_\infty, \|v\|_\infty \le \frac{\mu}{n^{1/2}}$. Then, we have*

1. $\|M(u, v, I)\|_\infty \le \frac{\kappa\mu}{n^{1/2}}\|M\|_\infty$, *where* $\kappa := \frac{Bd^2\mu}{\sqrt{n}}$.
2. $\|[M(u, v, I)]^p\|_\infty \le \kappa\mu\|M\|_\infty\|M\|^{p-1}$ *for* $p > 1$.
3. $\sum_{p \ge 1} \|[M(u, I, I)]^p v\|_\infty \le \frac{\kappa\mu}{\sqrt{n}}\|M\|_\infty \cdot \frac{\|M\|}{1 - \|M\|}$ *when* $\|M\| < 1/\sqrt{n}$.

Proof. We have from norm conversion

$$\begin{aligned}
\|M(u, v, I)\|_\infty &\le \|u\|_\infty \cdot \|v\|_\infty \max_j \|M(I, I, e_j)\|_1 && \text{(A.2)} \\
&\le \frac{\mu^2}{n} \cdot Bd^2\|M\|_\infty, && \text{(A.3)}
\end{aligned}$$

where ℓ_1 norm (i.e. sum of absolute values of entries) of a slice $M(I, I, e_j)$ is Bd^2, since the number of non-zero entries in one block in a slice is d^2.

Let $Z = M(u, I, I) \in \mathbb{R}^{n \times n}$. Now, $\|M(u, I, I)^p v\|_\infty = \|Z^p v\|_\infty = \|Z^{p-1} a\|_\infty$ where $a = Zv$. Now,

$$\|Z^{p-1}a\|_\infty = \max_j |e_j^T Z^{p-1} a| \le \|Z^{p-1}\|_2 \|a\|_2 \le \|Z\|_2^{p-1} \|a\|_2 \le \|M\|^{p-1} \sqrt{n} \|a\|_\infty \le \kappa\mu \|M\|_\infty \|M\|^{p-1}.$$

Hence, $\sum_{p \ge 1} \|[M(u, I, I)]^p v\|_\infty \le \kappa\mu \|M\|_\infty \cdot \frac{\|M\|_2}{1 - \|M\|_2}$. ■

A.2 Proof of Theorem 2.1

Lemma A.3. *Let L^*, S^* be symmetric and satisfy the assumptions of Theorem 2.1 and let $S^{(t)}$ be the t^{th} iterate of the l^{th} stage of Algorithm 1. Let $\sigma_1^*, \ldots, \sigma_r^*$ be the eigenvalues of L^*, such that $\sigma_1^* \ge \cdots \ge \sigma_r^* \ge 0$ and $\lambda_1, \cdots, \lambda_r$ be the eigenvalues of $T - S^{(t)}$ such that $\lambda_1 \ge \cdots \ge \lambda_r \ge 0$. Recall that $E^{(t)} := S^* - S^{(t)}$. Suppose further that*

1. $\|E^{(t)}\|_\infty \le \frac{8\mu^3 k}{n^{3/2}} \left(\sigma_{l+1}^* + \left(\frac{1}{2}\right)^{t-1} \sigma_l^* \right)$, *and*
2. $\operatorname{supp} E^{(t)} \subseteq \operatorname{supp} S^*$.

Then, for some constant $c \in [0, 1)$, we have

$$(1-c)\left(\sigma_{l+1}^* + \left(\frac{1}{2}\right)^t \sigma_l^*\right) \le \left(\lambda_{l+1} + \left(\frac{1}{2}\right)^t \lambda_l\right) \le (1+c)\left(\sigma_{l+1}^* + \left(\frac{1}{2}\right)^t \sigma_l^*\right). \quad \text{(A.4)}$$

Proof. Note that $T - S^{(t)} = L^* + E^{(t)}$. Now,

$$\left|\lambda_{l+1} - \sigma_{l+1}^*\right| \le 8 \left\|E^{(t)}\right\|_2 \le 8d^{3/2} \|E^{(t)}\|_\infty \le \frac{8\mu^3 r \gamma_t}{n^{3/2}} d^{3/2},$$

where $\gamma_t := \left(\sigma^*_{l+1} + \left(\frac{1}{2}\right)^{t-1} \sigma^*_l\right)$. That is, $\left|\lambda_{l+1} - \sigma^*_{l+1}\right| \leq 8\mu^3 r \left(\frac{d}{n}\right)^{3/2} \gamma_t$. Similarly, $|\lambda_l - \sigma^*_l| \leq 8\mu^3 r \left(\frac{d}{n}\right)^{3/2} \gamma_t$. So we have:

$$\begin{aligned}\left|\left(\lambda_{l+1} + \left(\frac{1}{2}\right)^t \lambda_l\right) - \left(\sigma^*_{l+1} + \left(\frac{1}{2}\right)^t \sigma^*_l\right)\right| &\leq 8\mu^3 r \left(\frac{d}{n}\right)^{3/2} \gamma_t \left(1 + \left(\frac{1}{2}\right)^t\right) \\ &\leq 16\mu^3 r \left(\frac{d}{n}\right)^{3/2} \gamma_t \\ &\leq c\left(\sigma^*_{l+1} + \left(\frac{1}{2}\right)^t \sigma^*_l\right),\end{aligned}$$

where the last inequality follows from the bound $d \leq \left(\frac{n}{c'\mu^3 k}\right)^{2/3}$ for some constant c'. ■

Lemma A.4. *Assume the notation of Lemma A.3. Also, let $L^{(t)}, S^{(t)}$ be the t^{th} iterates of r^{th} stage of Algorithm 1 and $L^{(t+1)}, S^{(t+1)}$ be the $(t+1)^{th}$ iterates of the same stage. Also, recall that $E^{(t)} := S^* - S^{(t)}$ and $E^{(t+1)} := S^* - S^{(t+1)}$.*

Suppose further that

1. $\|E^{(t)}\|_\infty \leq \frac{8\mu^3 r}{n^{3/2}}\left(\sigma^*_{l+1} + \left(\frac{1}{2}\right)^{t-1}\sigma^*_l\right)$, *and*
2. $\operatorname{supp} E^{(t)} \subseteq \operatorname{supp} S^*$.
3. $\|E^{(t)}\|_2 < \frac{C\sigma^*_l}{\sqrt{n}}$, *where $C < 1/2$ is a sufficiently small constant.*

Then, we have:

$$\|L^{(t+1)} - L^*\|_\infty \leq 2\frac{\mu^3 r}{n^{3/2}}\left(\sigma^*_{l+1} + \left(\frac{1}{2}\right)^t \sigma^*_l\right)$$

Proof. Let $L^{(t+1)} = \sum_{i=1}^l \lambda_i u_i^{(t+1)}$ be the eigen decomposition obtained using the tensor power method on $(T - S^{(t)})$ at the $(t+1)^{th}$ step of the l^{th} stage. Also, recall that $T - S^{(t)} = L^* + E^{(t)}$ where $L^* = \sum_{j=1}^r \sigma^*_j u_j^{\otimes 3}$. Define $E^{(t)} := S^* - S^{(t)}$. Define $E^i := E^{(t)}(u_i^{(t+1)}, I, I)$. Let $\|E^{(t)}\|_2 := \epsilon$.

Consider the eigenvalue equation $(T - S^{(t)})(u_i^{(t+1)}, u_i^{(t+1)}, I) = \lambda_i u_i^{(t+1)}$:

$$\begin{aligned}
L^*(u_i^{(t+1)}, u_i^{(t+1)}, I) + E^{(t)}(u_i^{(t+1)}, u_i^{(t+1)}, I) &= \lambda_i u_i^{(t+1)} \\
\sum_{j=1}^{r} \sigma_i^* \left\langle u_i^{(t+1)}, u_j \right\rangle^2 u_j + E^{(t)}(u_i^{(t+1)}, u_i^{(t+1)}, I) &= \lambda_i u_i^{(t+1)} \\
[\lambda_i I - E^{(t)}(u_i^{(t+1)}, I, I)] u_i^{(t+1)} &= \sum_{j=1}^{r} \sigma_i^* \left\langle u_i^{(t+1)}, u_j \right\rangle^2 u_j \\
u_i^{(t+1)} &= \left[I + \sum_{p \geq 1} \left(\frac{E^i}{\lambda_i} \right)^p \right] \sum_{j=1}^{r} \frac{\sigma_i^*}{\lambda_i} \left\langle u_i^{(t+1)}, u_j \right\rangle^2 u_j
\end{aligned}$$

Now,

$$\begin{aligned}
\|L^{(t+1)} - L^*\|_\infty &\leq \left\| \sum_{i \in [l]} \lambda_i (u_i^{(t+1)})^{\otimes 3} - \sum_{i \in [l]} \sigma_i^* {u_i}^{\otimes 3} \right\|_\infty + \left\| \sum_{i=l+1}^{r} \sigma_i^* {u_i}^{\otimes 3} \right\|_\infty \\
&\leq \sum_{i \in [l]} \left\| \lambda_i (u_i^{(t+1)})^{\otimes 3} - \sigma_i^* {u_i}^{\otimes 3} \right\|_\infty + \sum_{i=l+1}^{r} \|\sigma_i^* {u_i}^{\otimes 3}\|_\infty
\end{aligned}$$

For a fixed i, using $\lambda_i \leq \sigma_i^* + \epsilon$ [8] and using Lemma A.8, we obtain

$$\begin{aligned}
\left\| \lambda_i (u_i^{(t+1)})^{\otimes 3} - \sigma_i^* {u_i}^{\otimes 3} \right\|_\infty &\leq \left\| (\sigma_i^* + \epsilon)(u_i^{(t+1)})^{\otimes 3} - \sigma_i^* {u_i}^{\otimes 3} \right\|_\infty \\
&\leq \left\| \sigma_i^* (u_i^{(t+1)})^{\otimes 3} - \sigma_i^* {u_i}^{\otimes 3} \right\|_\infty + \epsilon \left\| (u_i^{(t+1)})^{\otimes 3} \right\|_\infty \\
&\leq \sigma_i^* \left\| (u_i^{(t+1)})^{\otimes 3} - {u_i}^{\otimes 3} \right\|_\infty + \epsilon \left\| (u_i^{(t+1)})^{\otimes 3} \right\|_\infty \\
&\leq \sigma_i^* [3\|u_i^{(t+1)} - u_i\|_\infty \|u_i\|_\infty^2 + 3\|u_i^{(t+1)} - u_i\|_\infty^2 \|u_i\|_\infty \\
&\quad + \|u_i^{(t+1)} - u_i\|_\infty^3] + \epsilon \|(u_i^{(t+1)})^{\otimes 3}\|_\infty \\
&\leq 7\sigma_i^* \|u_i^{(t+1)} - u_i\|_\infty \|u_i\|_\infty^2 + \epsilon \|(u_i^{(t+1)})\|_\infty^3
\end{aligned}$$

Now,

$$\begin{aligned}\left\|u_i^{(t+1)} - u_i\right\|_\infty &= \left\|(\sum_{j=1}^{r} \frac{\sigma_i^*}{\lambda_i} \left\langle u_i^{(t+1)}, u_j \right\rangle^2 u_j - u_i) + \sum_{j=1,p\geq 1}^{r} \frac{\sigma_i^*}{\lambda_i} \left\langle u_i^{(t+1)}, u_j \right\rangle^2 (E^i)^p u_j \right\|_\infty \\ &\leq \left\|(1 - \frac{\sigma_i^*}{\lambda_i} \left\langle u_i^{(t+1)}, u_i \right\rangle^2) u_i \right\|_\infty + \left\| \sum_{j\neq i} \frac{\sigma_i^*}{\lambda_i} \left\langle u_i^{(t+1)}, u_j \right\rangle^2 u_j \right\|_\infty \\ &+ \left\| \sum_{p\geq 1} \frac{\sigma_i^*}{\lambda_i} \left\langle u_i^{(t+1)}, u_i \right\rangle^2 \left(\frac{E^i}{\lambda_i}\right)^p u_i \right\|_\infty + \left\| \sum_{p,j\neq i} \frac{\sigma_i^*}{\lambda_i} \left\langle u_i^{(t+1)}, u_j \right\rangle^2 \left(\frac{E^i}{\lambda_i}\right)^p u_j \right\|_\infty\end{aligned}$$

For the first term, we have

$$\begin{aligned}\left\|(1 - \frac{\sigma_i^*}{\lambda_i} \left\langle u_i^{(t+1)}, u_i \right\rangle^2) u_i \right\|_\infty &\leq \left(1 - \frac{\sigma_i^*}{\sigma_i^* + \epsilon}\left(1 - \left(\frac{\epsilon}{\sigma_i^*}\right)^2\right)\right) \|u_i\|_\infty \\ &\leq \left(1 - \left(1 - \frac{\epsilon}{\sigma_i^*}\right)\right) \frac{\mu}{n^{1/2}} \\ &\leq \frac{\mu}{\sigma_i^* n^{1/2}} \epsilon \leq \frac{C\mu\sigma_l^*}{\sigma_i^* n}\end{aligned}$$

where we substitute for ϵ in the last step.

For the second term, we have

$$\left\| \sum_{j\neq i} \frac{\sigma_i^*}{\lambda_i} \left\langle u_i^{(t+1)}, u_j \right\rangle^2 u_j \right\|_\infty \leq \frac{\sigma_i^*}{\sigma_i^* - \epsilon} \left(\frac{\epsilon}{\sigma_i^*}\right)^2 \|u_i\|_\infty \leq 2\left(\frac{\epsilon}{\sigma_i^*}\right)^2 \frac{\mu}{n^{1/2}},$$

which is a lower order term.

Next,

$$\begin{aligned}\left\| \sum_{p\geq 1} \frac{\sigma_i^*}{\lambda_i} \left\langle u_i^{(t+1)}, u_i \right\rangle^2 \left(\frac{E^i}{\lambda_i}\right)^p u_i \right\|_\infty &\leq \left\| \sum_{p\geq 1} \frac{\sigma_i^*}{\lambda_i} \left(\frac{E^i}{\lambda_i}\right)^p u_i \right\|_\infty \leq \sum_{p\geq 1} \frac{\sigma_i^*}{\lambda_i} \left\| \left(\frac{E^i}{\lambda_i}\right)^p u_i \right\|_\infty \\ &\leq \frac{\sigma_i^*}{\lambda_i} \cdot \frac{\mu}{\sqrt{n}} \cdot \frac{\|E^{(t)}\|_2 \sqrt{n}/\lambda_i}{1 - \|E^{(t)}\|_2 \sqrt{n}/\lambda_i} \\ &\leq \frac{2}{(1-C)} \frac{\kappa_t \mu}{\lambda_i \sqrt{n}} \|E^{(t)}\|_\infty\end{aligned}$$

from Lemma A.2, and the assumption on spectral norm of $\|E^{(t)}\|_2$, where

$$\kappa_t := \frac{Bd^2\mu}{\sqrt{n}}.$$

For the remaining terms, we have

$$\begin{aligned}\left\|\sum_{p,j\neq i} \frac{\sigma_i^*}{\lambda_i} \left\langle u_i^{(t+1)}, u_j \right\rangle^2 \left(\frac{E^i}{\lambda_i}\right)^p u_j\right\|_\infty &\leq \sum_{j\neq i} \frac{\sigma_i^*}{\lambda_i} \left\langle u_i^{(t+1)}, u_j \right\rangle^2 \left\|\sum_{p\geq 1} \left(\frac{E^i}{\lambda_i}\right)^p u_1\right\|_\infty \\ &\leq \frac{\sigma_i^*}{\lambda_i} \left\|\sum_{p\geq 1} \left(\frac{E^i}{\lambda_i}\right)^p u_1\right\|_\infty \left(\frac{\epsilon}{\sigma_i^*}\right)^2,\end{aligned}$$

which is a lower order term.

Combining the above and recalling $\epsilon \ll \sigma_i^*$, $\forall i \in [l]$

$$\left\|u_i^{(t+1)} - u_i\right\|_\infty \leq \frac{8}{(1-C)} \frac{\kappa_t \mu}{\lambda_i \sqrt{n}} \|E^{(t)}\|_\infty.$$

Also, from Lemma 1

$$|\lambda_i - \sigma_i^*| \leq 8\|E^{(t)}\|_2 \leq 8\epsilon$$

Thus, from the above two equations, we obtain the bound for the parameters (eigenvectors and eigenvalues) of the low-rank tensor $\left\|u_i^{(t+1)} - u_i\right\|_\infty$ and $\|\lambda_i - \sigma_i^*\|_\infty$. We combine the individual parameter recovery bounds as:

$$\begin{aligned}\left\|\sum_{i\in[l]} \lambda_i (u_i^{(t+1)})^{\otimes 3} - \sum_{i\in[l]} \sigma_i^* u_i{}^{\otimes 3}\right\|_\infty &\leq r[7\sigma_i^* \|u_i^{(t+1)} - u_i\|_\infty \|u_i\|_\infty^2 + \epsilon \|(u_i^{(t+1)})\|_\infty^3] \\ &\leq \frac{224}{1-C} \frac{\kappa_t \mu^3 r}{n^{1.5}} \|E^{(t)}\|_\infty \qquad \text{(A.5)}\end{aligned}$$

and the other term

$$\sum_{i=l+1}^{r} \|\sigma_i^* u_i^{\otimes 3}\|_\infty \le \sigma_{l+1}^* \frac{r\mu^3}{n^{1.5}}.$$

Combining bound in (A.5) with the above, we have

$$\|L^{(t+1)} - L^*\|_\infty \le \frac{r\mu^3}{n^{1.5}} \left(\frac{224}{1-C} \kappa_t \|E^{(t)}\|_\infty + \sigma_{l+1}^* \right) < \frac{1}{4} \|E^{(t)}\|_\infty.$$

where the last inequality comes from the fact that $\frac{r\mu^3}{n^{1.5}}\sigma_{l+1}^* \le \frac{\|E^{(t)}\|_\infty}{8}$ and the assumption that $C < 1/2$, and we can choose B and d s.t.

$$\frac{448 r\mu^3}{n^{1.5}} \kappa_t < \frac{1}{8}.$$

This is possible from assumption (S). ■

The following lemma bounds the support of $E^{(t+1)}$ and $\|E^{(t+1)}\|_\infty$, using an assumption on $\|L^{(t+1)} - L^*\|_\infty$.

Lemma A.5. *Assume the notation of Lemma A.4. Suppose*

$$\|L^{(t+1)} - L^*\|_\infty \le 2\frac{\mu^3 r}{n^{3/2}} \left(\sigma_{l+1}^* + \left(\frac{1}{2}\right)^{t-1} \sigma_l^* \right).$$

Then, we have:

1. $\operatorname{supp} E^{(t+1)} \subseteq \operatorname{supp} S^*$.
2. $\|E^{(t+1)}\|_\infty \le 7\frac{\mu^3 r}{n^{3/2}} \left(\sigma_{l+1}^* + \left(\frac{1}{2}\right)^{t} \sigma_l^* \right)$, *and*

Proof. We first prove the first conclusion. Recall that,

$$S^{(t+1)} = H_\zeta(T - L^{(t+1)}) = H_\zeta(L^* - L^{(t+1)} + S^*),$$

where $\zeta = 4\frac{\mu^3 r}{n^{3/2}}\left(\lambda_{l+1} + \left(\frac{1}{2}\right)^t \lambda_l\right)$ is as defined in Algorithm 1 and $\lambda_1, \cdots, \lambda_n$ are the eigenvalues of $T - S^{(t)}$ such that $\lambda_1 \geq \cdots \geq \lambda_n$.

If $S^*_{abc} = 0$ then $E^{(t+1)}_{ijk} = \mathbf{1}_{\left|L^*_{abc} - L^{(t+1)}_{abc}\right| > \zeta} \cdot (L^*_{abc} - L^{(t+1)}_{abc})$. The first part of the lemma now follows by using the assumption that $\|L^{(t+1)} - L^*\|_\infty \leq 2\frac{\mu^3 r}{n^{3/2}}\left(\sigma^*_{l+1} + \left(\frac{1}{2}\right)^t \sigma^*_l\right) \overset{(\zeta_1)}{\leq} 4\frac{\mu^3 r}{n^{3/2}}\left(\lambda_{l+1} + \left(\frac{1}{2}\right)^t \lambda_l\right) = \zeta$, where (ζ_1) follows from Lemma A.3.

We now prove the second conclusion. We consider the following two cases:

1. $\left|T_{abc} - L^{(t+1)}_{abc}\right| > \zeta$: Here, $S^{(t+1)}_{abc} = S^*_{abc} + L^*_{abc} - L^{(t+1)}_{abc}$. Hence, $|S^{(t+1)}_{abc} - S^*_{abc}| \leq |L^*_{abc} - L^{(t+1)}_{abc}| \leq 2\frac{\mu^3 r}{n^{3/2}}\left(\sigma^*_{l+1} + \left(\frac{1}{2}\right)^t \sigma^*_l\right)$.
2. $\left|T_{abc} - L^{(t+1)}_{abc}\right| \leq \zeta$: In this case, $S^{(t+1)}_{abc} = 0$ and $\left|S^*_{abc} + L^*_{abc} - L^{(t+1)}_{abc}\right| \leq \zeta$. So we have, $\left|E^{(t+1)}_{abc}\right| = |S^*_{abc}| \leq \zeta + \left|L^*_{abc} - L^{(t+1)}_{abc}\right| \leq 7\frac{\mu^3 r}{n^{3/2}}\left(\sigma^*_{l+1} + \left(\frac{1}{2}\right)^t \sigma^*_l\right)$. The last inequality above follows from Lemma A.3.

This proves the lemma. ■

Theorem A.6. *Let L^*, S^* be symmetric and satisfy (L) and (S), and $\beta = 4\frac{\mu^3 r}{n^{3/2}}$. The outputs $\widehat{L}$ (and its parameters $\widehat{u}_i$ and $\widehat{\lambda}_i$) and $\widehat{S}$ of Algorithm 1 satisfy w.h.p.:*

$$\|\widehat{u}_i - u_i\|_\infty \leq \frac{\delta}{\mu^2 r n^{1/2} \sigma^*_{\min}}, \quad |\widehat{\lambda}_i - \sigma^*_i| \leq \delta, \quad \forall i \in [n],$$
$$\left\|\widehat{L} - L^*\right\|_F \leq \delta, \quad \|\widehat{S} - S^*\|_\infty \leq \frac{\delta}{n^{3/2}}, \text{ and } \quad \operatorname{supp}\widehat{S} \subseteq \operatorname{supp} S^*.$$

Proof. Recall that in the l^{th} stage, the update $L^{(t+1)}$ is given by: $L^{(t+1)} = P_l(T - S^{(t)})$ and $S^{(t+1)}$ is given by: $S^{(t+1)} = H_\zeta(T - L^{(t+1)})$. Also, recall that $E^{(t)} := S^* - S^{(t)}$ and $E^{(t+1)} := S^* - S^{(t+1)}$.

We prove the lemma by induction on both l and t. For the base case ($l = 1$ and $t = -1$), we first note that the first inequality on $\|L^{(0)} - L^*\|_\infty$ is trivially satisfied. Due to the thresholding step (step 3 in Algorithm 1) and the incoherence assumption on L^*, we have:

$$\|E^{(0)}\|_\infty \leq \frac{8\mu^3 r}{n^{3/2}} \left(\sigma_2^* + 2\sigma_1^*\right), \text{ and } \operatorname{supp} E^{(0)} \subseteq \operatorname{supp} S^*.$$

So the base case of induction is satisfied.

We first do the inductive step over t (for a fixed r). By inductive hypothesis we assume that: a) $\|E^{(t)}\|_\infty \leq \frac{8\mu^3 r}{n^{3/2}} \left(\sigma_{l+1}^* + \left(\frac{1}{2}\right)^{t-1} \sigma_l^*\right)$, b) $\operatorname{supp} E^{(t)} \subseteq \operatorname{supp} S^*$. Then by Lemma A.4, we have:

$$\|L^{(t+1)} - L^*\|_\infty \leq \frac{2\mu^3 r}{n^{3/2}} \left(\sigma_{l+1}^* + \left(\frac{1}{2}\right)^t \sigma_l^*\right).$$

Lemma A.5 now tells us that

1. $\|E^{(t+1)}\|_\infty \leq \frac{8\mu^3 r}{n^{3/2}} \left(\sigma_{l+1}^* + \left(\frac{1}{2}\right)^t \sigma_l^*\right)$, and
2. $\operatorname{supp} E^{(t+1)} \subseteq \operatorname{supp} S^*$.

This finishes the induction over t. Note that we show a stronger bound than necessary on $\|E^{(t+1)}\|_\infty$.

We now do the induction over l. Suppose the hypothesis holds for stage l. Let T denote the number of iterations in each stage. We first obtain a lower bound on T. Since

$$\left\|T - S^{(0)}\right\|_2 \geq \|L^*\|_2 - \left\|E^{(0)}\right\|_2 \geq \sigma_1^* - d^{3/2}\|E^{(0)}\|_\infty \geq \frac{3}{4}\sigma_1^*,$$

we see that $T \geq 10 \log\left(3\mu^3 r \sigma_1^*/\delta\right)$. So, at the end of stage r, we have:

1. $\|E^{(T)}\|_\infty \leq \frac{7\mu^3 r}{n^{3/2}} \left(\sigma_{l+1}^* + \left(\frac{1}{2}\right)^T \sigma_l^*\right) \leq \frac{7\mu^3 r \sigma_{l+1}^*}{n^{3/2}} + \frac{\delta}{10n}$, and

2. $\operatorname{supp} E^{(T)} \subseteq \operatorname{supp} S^*$.

Recall, $\left|\sigma_{r+1}\left(T - S^{(T)}\right) - \sigma^*_{r+1}\right| \leq \left\|E^{(T)}\right\|_2 \leq \frac{d}{n}\left(\mu^3 r \left|\sigma^*_{r+1}\right| + \delta\right)$. We will now consider two cases:

1. **Algorithm 1 terminates:** This means that $\beta\sigma_{r+1}\left(T - S^{(T)}\right) < \frac{\delta}{2n^{3/2}}$ which then implies that $\sigma^*_{r+1} < \frac{\delta}{6\mu^3 r}$. So we have:

$$\|\widehat{L} - L^*\|_\infty = \|L^{(T)} - L^*\|_\infty \leq \frac{2\mu^3 r}{n^{3/2}}\left(\sigma^*_{r+1} + \left(\frac{1}{2}\right)^T \sigma^*_r\right) \leq \frac{\delta}{5n^{3/2}}.$$

   This proves the statement about $\widehat{L}$ and its parameters (eigenvalues and eigenvectors). A similar argument proves the claim on $\|\widehat{S} - S^*\|_\infty$. The claim on $\operatorname{supp}\widehat{S}$ follows since $\operatorname{supp} E^{(T)} \subseteq \operatorname{supp} S^*$.

2. **Algorithm 1 continues to stage $(r+1)$:** This means that $\beta\sigma_{r+1}\left(L^{(T)}\right) \geq \frac{\delta}{2n^{3/2}}$ which then implies that $\sigma^*_{r+1} > \frac{\delta}{8\mu^3 r}$. So we have:

$$\begin{aligned}
\|E^{(T)}\|_\infty &\leq \frac{8\mu^3 r}{n^{3/2}}\left(\sigma^*_{r+1} + \left(\frac{1}{2}\right)^T \sigma^*_r\right) \\
&\leq \frac{8\mu^3 r}{n^{3/2}}\left(\sigma^*_{l+1} + \frac{\delta}{10\mu^3 r n^{3/2}}\right) \\
&\leq \frac{8\mu^3 r}{n^{3/2}}\left(\sigma^*_{l+1} + \frac{8\sigma^*_{l+1}}{10n}\right) \\
&\leq \frac{8\mu^3 r}{n^{3/2}}\left(\sigma^*_{l+2} + 2\sigma^*_{l+1}\right).
\end{aligned}$$

   Similarly for $\|L^{(T)} - L^*\|_\infty$.

This finishes the proof. ■

A.2.1 Short proof of Corollary 1

The state of art guarantees for robust matrix PCA requires that the overall sparsity along any row or column of the input matrix be $D = O(\frac{n}{r\mu^2})$ (when the input matrix is $\mathbb{R}^{n\times n}$).

Under (S), the total sparsity along any row or column of M_i is given by $D := dB$. Now, Theorem 2.1 holds when the sparsity condition in (2.2) is satisfied. That is, RTD succeeds when

$$D = O(d \cdot B) = O\left(\min\left(\frac{n^{4/3}}{r^{1/3}\mu^2}, \frac{n^{2/3}}{r^{2/3}\mu^2}(\frac{n}{r})^{1/3}\right)\right) = O\left(\frac{n}{r\mu^2}\right).$$

Hence, RTD can handle larger amount of corruption than the matrix methods and the gain becomes more significant for smaller η.

A.2.2 Some auxiliary lemmas

We recall Theorem 5.1 from [8]. Let $\epsilon = 8\|E^{(t)}\|_2$ where $E^{(t)} := S^* - S^{(t)}$.

Lemma A.7. *Let $L^{(t+1)} = \sum_{i=1}^k \lambda_i u_i^{(t+1)}$ be the eigen decomposition obtained using Algorithm 2 on $(T - S^{(t)})$. Then,*

1. *If $\|u_i^{(t+1)} - u_i\|_2 \leq \frac{\epsilon}{\sigma^*_{\min}}$, then $\operatorname{dist}(u_i^{(t+1)}, u_i) \leq \frac{\epsilon}{\sigma^*_{\min}}$.*
2. $\sum_{j\neq i} \left\langle u_i^{(t+1)}, u_j \right\rangle^2 \leq \left(\frac{\epsilon}{\sigma^*_{\min}}\right)^2$.
3. $\|u_i^{(t+1)}\|_\infty \leq \frac{\mu}{n^{1/2}} + \frac{\epsilon}{\sigma^*_{\min}}$.
4. $|\sigma_i^*| - \epsilon \leq |\lambda_i| \leq |\sigma_i^*| + \epsilon$.

Proof. 1. Let $z \perp u$ and $\|z\|_2 = 1$.

$$\begin{aligned}
u_i^{(t+1)} &= \left\langle u_i^{(t+1)}, u_i \right\rangle u_i + \operatorname{dist}(u_i^{(t+1)}, u_i) z \\
\|u_i^{(t+1)} - u_i\|_2^2 &= (\left\langle u_i^{(t+1)}, u_i \right\rangle - 1)^2 \|u_i\|_2^2 + \operatorname{dist}(u_i^{(t+1)}, u_i)\|z\|_2^2 + 0 \\
&\geq (\operatorname{dist}(u_i^{(t+1)}, u_i))^2
\end{aligned}$$

Then using Theorem 5.1 from [8], we obtain the result. Next, since $\langle u_i^{(t+1)}, u_i \rangle^2 + \operatorname{dist}(u_i^{(t+1)}, u_i)^2 = 1$, we have $\langle u_i^{(t+1)}, u_i \rangle^2 \geq 1 - \left(\frac{\epsilon}{\sigma^*_{\min}}\right)^2$.

2. Note that
$$u_i^{(t+1)} = \sum_{j=1}^{k} \left\langle u_i^{(t+1)}, u_j \right\rangle u_j + \operatorname{dist}(u_i^{(t+1)}, U) z$$
where $z \perp U$ such that $\|z\|_2 = 1$. Using $\|u_i^{(t+1)}\|_2 = 1$ and the Pythagoras theorem, we get
$$1 - \left\langle u_i^{(t+1)}, u_i \right\rangle^2 = \sum_{j \neq i} \left\langle u_i^{(t+1)}, u_j \right\rangle^2 + \operatorname{dist}(u_i^{(t+1)}, U)^2 .1 \geq \sum_{j \neq i} \left\langle u_i^{(t+1)}, u_j \right\rangle^2$$
Using part 1 of Lemma A.7, we get $\sum_{j \neq i} \left\langle u_i^{(t+1)}, u_j \right\rangle^2 \leq \left(\frac{\epsilon}{\sigma^*_{\min}}\right)^2$.

3. We have
$$\begin{aligned}
u_i^{(t+1)} &= \left\langle u_i^{(t+1)}, u_i \right\rangle u_i + \operatorname{dist}(u_i^{(t+1)}, u_i) z \\
\|u_i^{(t+1)}\|_\infty &\leq |\left\langle u_i^{(t+1)}, u_i \right\rangle| \|u_i\|_\infty + |\operatorname{dist}(u_i^{(t+1)}, u_i)| \|z\|_\infty \leq 1 . \frac{\mu}{n^{1/2}} + \frac{\epsilon}{\sigma^*_{\min}}
\end{aligned}$$

4. This follows from Theorem 5.1 from [8], i.e., $\forall i,\ ||\lambda_i| - |\sigma_i^*|| \leq \epsilon$.

■

Lemma A.8. *Let $a = b + \epsilon. \overrightarrow{1}$ where a, b are any 2 vectors and $\epsilon > 0$. Then, $\|a^{\otimes 3} - b^{\otimes 3}\|_\infty \leq \|a - b\|_\infty . \|b\|_\infty^2 + O(\epsilon^2)$.*

Proof. We have

$$\|a^{\otimes 3} - b^{\otimes 3}\|_\infty = \|(b + \epsilon \overrightarrow{1})^{\otimes 3} - b^{\otimes 3}\|_\infty$$

Let (i, j, k) be the maximum element. Therefore,

$$\begin{aligned}\|(b + \epsilon \overrightarrow{1})^{\otimes 3} - b^{\otimes 3}\|_\infty &= (b_i + \epsilon)(b_j + \epsilon)(b_k + \epsilon) - b_i b_j b_k \\ &= \epsilon(b_i b_j + b_j b_k + b_k b_i) + \epsilon^2(b_i + b_j + b_k) + \epsilon^3\end{aligned}$$

With $b_i \leq c \ \forall i$ for some $c > 0$ and $\epsilon = \|a - b\|_\infty$, we have $\|a^{\otimes 3} - b^{\otimes 3}\|_\infty \leq 3\epsilon c^2 + O(\epsilon^2)$ ■

A.3 Symmetric embedding of an asymmetric tensor

We use the symmetric embedding $sym(L)$ of a tensor L as defined in Section 2.3 of [84]. We focus on third order tensors which have low CP-rank. We have three properties to derive that is relevant to us:

1. *Symmetry:* From Lemma 2.2 of [84] we see that $sym(L)$ for any tensor is symmetric.
2. *CP-Rank:* From Equation 6.5 of [84] we see that CP-rank($sym(L)$) $\leq$ 6.CP-rank(L). Since this is a constant, we see that the symmetric embedding is also a low-rank tensor.
3. *Incoherece:* Theorem 4.7 of [84] says that if u_1, u_2 and u_3 are unit modal singular vectors of T, then the vector $\widetilde{u} = 3^{-1/2}[u_1; u_2; u_3]$ is a unit eigenvector of $sym(T)$. Without loss of generality, assume that T is of size $n_1 \times n_2 \times n_3$ with $n_1 \leq n_2 \leq n_3$. In this case, we have

$$\|\widetilde{u}\|_\infty \leq \frac{\mu}{(3n_1)^{1/2}} \tag{A.6}$$

and

$$\|\widetilde{u}\|_\infty \leq \frac{\widetilde{\mu}}{(n_1+n_2+n_3)^{1/2}} \tag{A.7}$$

for $\widetilde{\mu} = c\mu$ for some constant c to be calculated. Equating the right hand sides of Equations (A.6) and (A.7), we obtain $c = [(n_1+n_2+n_3)/(3n_1)]^{1/2}$. When $\Theta(n_1) = \Theta(n_2) = \Theta(n_3)$, we see that the eigenvectors $\widetilde{u}$ of $sym(T)$ as specified above have the incoherence-preserving property.

A.4 Proof of Theorem 2.2

Let $\widetilde{L}$ be a symmetric tensor which is a perturbed version of an orthogonal tensor L^*, $\widetilde{L} = L^* + E \in \mathbb{R}^{n\times n\times n}$, $\quad L^* = \sum_{i\in[r]} \sigma_i^* u_i^{\otimes 3}$, where $\sigma_1^* \geq \sigma_2^* \ldots \sigma_r^* > 0$ and $\{u_1, u_2, \ldots, u_r\}$ form an orthonormal basis.

The analysis proceeds iteratively. First, we prove convergence to eigenpair of $\widetilde{L}$, which is close to top eigenpair (σ_1^*, u_1) of L^*. We then argue that the same holds on the deflated tensor, when the perturbation E satisfies (2.8). from This finishes the proof of Theorem 2.2.

To prove convergence for the first stage, i.e. convergence to eigenpair of $\widetilde{L}$, which is close to top eigenpair (σ_1^*, u_1) of L^*, we analyze two phases of the shifted power iteration. In the first phase, we prove that with N_1 initializations and N_2 power iterations, we get close to true top eigenpair of L^*, i.e. (σ_1^*, u_1). After this, in the second phase, we prove convergence to an eigenpair of $\widetilde{L}$.

The proof of the second phase is outlined in the main text. Here, we now provide proof for the first phase.

A.4.1 Analysis of first phase of shifted power iteration

In this section, we prove that the output of shifted power method is close to original eigenpairs of the (unperturbed) orthogonal tensor, i.e. Theorem 2.2 holds, except for the property that the output corresponds to the eigenpairs of the perturbed tensor. We adapt the proof of tensor power iteration from [8] but here, since we consider the shifted power method, we need to modify it. We adopt the notation of [8] in this section.

Recall the update rule used in the shifted power method. Let $\theta_t = \sum_{i=1}^{k} \theta_{i,t} v_i \in \mathbb{R}^k$ be the unit vector at time t. Then

$$\theta_{t+1} = \sum_{i=1}^{k} \theta_{i,t+1} v_i := (\widetilde{T}(I, \theta_t, \theta_t) + \alpha\theta_t)/\|(\widetilde{T}(I, \theta_t, \theta_t) + \alpha\theta_t)\|.$$

In this subsection, we assume that $\widetilde{T}$ has the form

$$\widetilde{T} = \sum_{i=1}^{k} \widetilde{\lambda}_i v_i^{\otimes 3} + \widetilde{E} \tag{A.8}$$

where $\{v_1, v_2, \ldots, v_k\}$ is an orthonormal basis, and, without loss of generality,

$$\widetilde{\lambda}_1 |\theta_{1,t}| = \max_{i \in [k]} \widetilde{\lambda}_i |\theta_{i,t}| > 0.$$

Also, define

$$\widetilde{\lambda}_{\min} := \min\{\widetilde{\lambda}_i : i \in [k],\ \widetilde{\lambda}_i > 0\}, \quad \widetilde{\lambda}_{\max} := \max\{\widetilde{\lambda}_i : i \in [k]\}.$$

We assume the error $\widetilde{E}$ is a symmetric tensor such that, for some constant $p > 1$,

$$\|\widetilde{E}(I,u,u)\| \le \widetilde{\epsilon}, \quad \forall u \in S^{k-1}; \tag{A.9}$$

$$\|\widetilde{E}(I,u,u)\| \le \widetilde{\epsilon}/p, \quad \forall u \in S^{k-1} \text{ s.t. } (u^\top v_1)^2 \ge 1 - (3\widetilde{\epsilon}/\widetilde{\lambda}_1)^2. \tag{A.10}$$

In the next two propositions (Propositions A.4.1 and A.4.2) and Lemmas A.4.1, we analyze the power method iterations using $\widetilde{T}$ at some arbitrary iterate θ_t using only the property (A.9) of $\widetilde{E}$. But throughout, the quantity $\widetilde{\epsilon}$ can be replaced by $\widetilde{\epsilon}/p$ if θ_t satisfies $(\theta_t^\top v_1)^2 \ge 1 - (3\widetilde{\epsilon}/\widetilde{\lambda}_1)^2$ as per property (A.10).

Define

$$\begin{aligned} R_\tau &:= \left(\frac{\theta_{1,\tau}^2}{1-\theta_{1,\tau}^2}\right)^{1/2}, \qquad r_{i,\tau} := \frac{\widetilde{\lambda}_1 \theta_{1,\tau}}{\widetilde{\lambda}_i |\theta_{i,\tau}|}, \\ \gamma_\tau &:= 1 - \frac{1}{\min_{i\neq 1} |r_{i,\tau}|}, \qquad \delta_\tau := \frac{\widetilde{\epsilon}}{\widetilde{\lambda}_1 \theta_{1,\tau}^2}, \qquad \kappa := \frac{\widetilde{\lambda}_{\max}}{\widetilde{\lambda}_1} \end{aligned} \tag{A.11}$$

for $\tau \in \{t, t+1\}$.

Proposition A.4.1.

$$\min_{i\neq 1} |r_{i,t}| \ge \frac{R_t}{\kappa}, \qquad \gamma_t \ge 1 - \frac{\kappa}{R_t}, \qquad \theta_{1,t}^2 = \frac{R_t^2}{1+R_t^2}.$$

Proposition A.4.2.

$$r_{i,t+1} \ge r_{i,t}^2 \cdot \frac{1 - \delta_t + \frac{\alpha}{\widetilde{\lambda}_1 \theta_{1,t}}}{1 + \kappa \delta_t r_{i,t}^2 + \frac{\alpha}{\widetilde{\lambda}_i \theta_{i,t}}}, \quad i \in [k], \tag{A.12}$$

$$R_{t+1} \ge= R_t \cdot \frac{1 - \delta_t + \frac{\alpha}{\widetilde{\lambda}_1 |\theta_{1,t}|}}{1 - \gamma_t + \left(\delta_t + \frac{\alpha (1-\theta_{1,t})^{1/2}}{\widetilde{\lambda}_1 \theta_{1,t}^2}\right) R_t} \ge \frac{1 - \delta_t + \frac{\alpha}{\widetilde{\lambda}_1 |\theta_{1,t}|}}{\frac{\kappa}{R_t^2} + \delta_t + \frac{\alpha (1-\theta_{1,t})^{1/2}}{\widetilde{\lambda}_1 \theta_{1,t}^2}}. \tag{A.13}$$

Proof. Let $\check{\theta}_{t+1} := \widetilde{T}(I, \theta_t, \theta_t) + \alpha \theta_t$, so $\theta_{t+1} = \check{\theta}_{t+1} / \|\check{\theta}_{t+1}\|$. Since $\check{\theta}_{i,t+1} = \widetilde{T}(v_i, \theta_t, \theta_t) =$

$T(v_i, \theta_t, \theta_t) + \alpha\theta_t + E(v_i, \theta_t, \theta_t)$, we have

$$\check{\theta}_{i,t+1} = \widetilde{\lambda}_i \theta_{i,t}^2 + E(v_i, \theta_t, \theta_t) + \alpha\theta_t^\top v_i, \quad i \in [k].$$

By definition, we have $\theta_{i,t} = \theta_t^\top v_i$. Using the triangle inequality and the fact $\|E(v_i, \theta_t, \theta_t)\| \leq \widetilde{\epsilon}$, we have

$$\check{\theta}_{i,t+1} \geq \widetilde{\lambda}_i \theta_{i,t}^2 - \widetilde{\epsilon} + \alpha\theta_{i,t} \geq |\theta_{i,t}| \cdot \Big(\widetilde{\lambda}_i |\theta_{i,t}| - \widetilde{\epsilon}/|\theta_{i,t}| + \alpha\Big) \tag{A.14}$$

and

$$|\check{\theta}_{i,t+1}| \leq |\widetilde{\lambda}_i \theta_{i,t}^2| + \widetilde{\epsilon} + \alpha\theta_{i,t} \leq |\theta_{i,t}| \cdot \Big(\widetilde{\lambda}_i |\theta_{i,t}| + \widetilde{\epsilon}/|\theta_{i,t}| + \alpha\Big) \tag{A.15}$$

for all $i \in [k]$. Combining (A.14) and (A.15) gives

$$\begin{aligned}
r_{i,t+1} &= \frac{\widetilde{\lambda}_1 \theta_{1,t+1}}{\widetilde{\lambda}_i |\theta_{i,t+1}|} = \frac{\widetilde{\lambda}_1 \check{\theta}_{1,t+1}}{\widetilde{\lambda}_i |\check{\theta}_{i,t+1}|} \\
&\geq r_{i,t}^2 \cdot \frac{1 - \delta_t + \frac{\alpha}{\widetilde{\lambda}_1 \theta_{1,t}}}{1 + \frac{\widetilde{\epsilon}}{\widetilde{\lambda}_i \theta_{i,t}^2} + \frac{\alpha}{\widetilde{\lambda}_i \theta_{i,t}}} = r_{i,t}^2 \cdot \frac{1 - \delta_t + \frac{\alpha}{\widetilde{\lambda}_1 \theta_{1,t}}}{1 + (\widetilde{\lambda}_i/\widetilde{\lambda}_1)\delta_t r_{i,t}^2 + \frac{\alpha}{\widetilde{\lambda}_i \theta_{i,t}}} \\
&\geq r_{i,t}^2 \cdot \frac{1 - \delta_t + \frac{\alpha}{\widetilde{\lambda}_1 \theta_{1,t}}}{1 + \kappa\delta_t r_{i,t}^2 + \frac{\alpha}{\widetilde{\lambda}_i \theta_{i,t}}}.
\end{aligned}$$

Moreover, by the triangle inequality and Hölder's inequality,

$$\begin{aligned}
\left(\sum_{i=2}^n [\check{\theta}_{i,t+1}]^2\right)^{1/2} &= \left(\sum_{i=2}^n \Big(\widetilde{\lambda}_i \theta_{i,t}^2 + E(v_i, \theta_t, \theta_t) + \alpha\theta_{i,t}\Big)^2\right)^{1/2} \\
&\leq \left(\sum_{i=2}^n \widetilde{\lambda}_i^2 \theta_{i,t}^4\right)^{1/2} + \left(\sum_{i=2}^n E(v_i, \theta_t, \theta_t)^2\right)^{1/2} + \left(\sum_{i=2}^k \alpha^2 \theta_{i,t}^2\right)^{1/2} \\
&\leq \max_{i \neq 1} \widetilde{\lambda}_i |\theta_{i,t}| \left(\sum_{i=2}^n \theta_{i,t}^2\right)^{1/2} + \widetilde{\epsilon} + \left(\alpha^2 \sum_{i=2}^k \theta_{i,t}^2\right)^{1/2} \\
&= (1 - \theta_{1,t}^2)^{1/2} \cdot \Big(\max_{i \neq 1} \widetilde{\lambda}_i |\theta_{i,t}| + \widetilde{\epsilon}/(1 - \theta_{1,t}^2)^{1/2} + \alpha\Big).
\end{aligned} \tag{A.16}$$

Combining (A.14) and (A.16) gives

$$\frac{|\theta_{1,t+1}|}{(1-\theta_{1,t+1}^2)^{1/2}} = \frac{|\check{\theta}_{1,t+1}|}{\left(\sum_{i=2}^n [\check{\theta}_{i,t+1}]^2\right)^{1/2}} \geq \frac{|\theta_{1,t}|}{(1-\theta_{1,t}^2)^{1/2}} \cdot \frac{\widetilde{\lambda}_1|\theta_{1,t}| - \widetilde{\epsilon}/|\theta_{1,t}| + \alpha}{\max_{i\neq 1} \widetilde{\lambda}_i|\theta_{i,t}| + \widetilde{\epsilon}/(1-\theta_{1,t}^2)^{1/2} + \alpha}.$$

In terms of R_{t+1}, R_t, γ_t, and δ_t, this reads

$$\begin{aligned} R_{t+1} &\geq \frac{1-\delta_t + \frac{\alpha}{\widetilde{\lambda}_1|\theta_{1,t}|}}{(1-\gamma_t)\left(\frac{1-\theta_{1,t}^2}{\theta_{1,t}^2}\right)^{1/2} + \delta_t + \frac{\alpha(1-\theta_{1,t})^{1/2}}{\widetilde{\lambda}_1\theta_{1,t}^2}} = R_t \cdot \frac{1-\delta_t + \frac{\alpha}{\widetilde{\lambda}_1|\theta_{1,t}|}}{1-\gamma_t + \left(\delta_t + \frac{\alpha(1-\theta_{1,t}^2)^{1/2}}{\widetilde{\lambda}_1\theta_{1,t}^2}\right) R_t} \\ &= \frac{1-\delta_t + \frac{\alpha}{\widetilde{\lambda}_1|\theta_{1,t}|}}{\frac{1-\gamma_t}{R_t} + \left(\delta_t + \frac{\alpha(1-\theta_{1,t}^2)^{1/2}}{\widetilde{\lambda}_1\theta_{1,t}^2}\right)} \geq \frac{1-\delta_t + \frac{\alpha}{\widetilde{\lambda}_1|\theta_{1,t}|}}{\frac{\kappa}{R_t^2} + \delta_t + \frac{\alpha(1-\theta_{1,t}^2)^{1/2}}{\widetilde{\lambda}_1\theta_{1,t}^2}} \end{aligned}$$

where the last inequality follows from Proposition A.4.1. ■

Lemma A.4.1. *Fix any $\rho > 1$. Assume*

$$0 \leq \delta_t < \min\left\{\frac{1}{2(1+2\kappa\rho^2)}, \frac{1-1/\rho}{1+\kappa\rho}\right\}$$

and $\gamma_t > 2(1+2\kappa\rho^2)\delta_t$.

1. *If $r_{i,t}^2 \leq 2\rho^2$, then $r_{i,t+1} \geq |r_{i,t}|\left(1+\frac{\gamma_t}{2}\right)$.*

Proof. By (A.12) from Proposition A.4.2,

$$r_{i,t+1} \geq r_{i,t}^2 \cdot \frac{1-\delta_t + \frac{\alpha}{\widetilde{\lambda}_1\theta_{1,t}}}{1+\kappa\delta_t r_{i,t}^2 + \frac{\alpha}{\widetilde{\lambda}_i\theta_{i,t}}} \geq |r_{i,t}| \cdot \frac{1}{1-\gamma_t} \cdot \frac{1-\delta_t + \frac{\alpha}{\widetilde{\lambda}_1\theta_{1,t}}}{1+2\kappa\rho^2\delta_t + \frac{\alpha}{\widetilde{\lambda}_i\theta_{i,t}}} \geq |r_{i,t}|\left(1+\frac{\gamma_t}{2}\right)$$

where the last inequality is seen as follows: Let

$$\xi = 2.\frac{1-\delta_t + \frac{\alpha}{\widetilde{\lambda}_1\theta_{1,t}}}{1+2\kappa\rho^2\delta_t + \frac{\alpha}{\widetilde{\lambda}_i\theta_{i,t}}}$$

Then, we have $\gamma_t^2 + \gamma_t - 2 + \xi \geq 0$. The positive root is $\frac{-1+(9-4\xi)^{1/2}}{2}$. Since $\gamma_t \geq 0$, we have

$(9-4\xi)^{1/2} \geq 1$, so we assume $\xi \leq 2$ for the inequality to hold, i.e., $\frac{\alpha}{\widetilde{\lambda}_1 \theta_{1,t}} - \frac{\alpha}{\widetilde{\lambda}_i \theta_{i,t}} \leq (1+2\kappa\rho^2)\delta_t$.
■

The rest of the proof is along the similar lines of [8], except that we use SVD initialization instead of random initialization. The proof of SVD initialization is given in [9].

Appendix B

Appendix for Higher-order Count Sketch

B.1 List of some algorithms mentioned in the chapter

B.1.1 Count sketch

Algorithm 5 Count Sketch

```
1: procedure CS($x, c$)                                   ▷ $x \in \mathbb{R}^n$
2:     $s \in Maps([n] \Rightarrow \{-1, +1\})$
3:     $h \in Maps([n] \Rightarrow \{0, \cdots c\})$
4:     for i:1 to n do
5:         $y[h[i]] += s[i]x[i]$
6:     return $y$
7: procedure CS-Decompress($y$)
8:     for i:1 to n do
9:         $\widehat{x}[i] = s[i]y[h[i]]$
10:    return $\widehat{x}$
```

B.1.2 Higher-order count sketch

Algorithm 6 Higher-order Count Sketch

```
1: procedure HCS($T, M_{list}$)                                   ▷ $T \in \mathbb{R}^{n_1 \times \cdots \times n_N}$
2:                                   ▷ $M_{list}$ contains sketching parameters: $m_1 \dots m_N$
3:     Generate hash functions $s_1, \cdots s_N$, $h_1, \cdots h_N$ given $M_{list}$
4:     Compute hash matrices $\mathcal{S}$, $H_1, \cdots H_N$
5:     return $(\mathcal{S} \circ T)(H_1, \cdots, H_N)$
6: procedure HCS-DECOMPRESS(HCS($T$))
7:     return $\mathcal{S} \circ \text{HCS}(T)(H_1^T, \cdots, H_N^T)$
```

B.1.3 Approximate Kronecker product

Algorithm 7 Compress/Decompress Kronecker Product

```
1: procedure COMPRESS-KP($A, B, m_1, m_2$)          ▷ $A \in \mathbb{R}^{n_1 \times n_2}, B \in \mathbb{R}^{n_3 \times n_4}$
2:     for X in [A,B] do
3:         $X^{HCS}$ = HCS($X, [m_1, m_2]$)
4:     FFT2($A^{HCS}$),FFT2($B^{HCS}$)
5:     $P$=IFFT2($A^{HCS}$ ∘ $B^{HCS}$)
6:     return $(P)$
7:
8: procedure DECOMPRESS-KP($P$)
9:     $C$ = zeros($n_1 n_3, n_2 n_4$)
10:    for w,q,o,g:=1 to $n_1, n_2, n_3, n_4$ do
11:        k = $(h_{A1}[w] + h_{B1}[o])$ mod $m_1$
12:        l = $(h_{A2}[q] + h_{B2}[g])$ mod $m_2$
13:        tmp = $s_{A1}[w] s_{A2}[q] s_{B1}[o] s_{B2}[g] P[k, l]$
14:        i = $n_3(w-1) + o$
15:        j = $n_4(q-1) + g$
16:        $C_{ij}$ = tmp
17:    return $(C)$
```

B.1.4 Approximate Matrix product

Algorithm 8 Compress/Decompress Matrix Product

```
1: procedure COMPRESS-MP($A, B, m_1, m_2, m_3$)        ▷ $A \in \mathbb{R}^{n_1 \times k}, B \in \mathbb{R}^{k \times n_2}$
2:     $A^{HCS}$ = HCS($A, [m_1, m_2]$)  ▷ Choose hash matrix along k mode be identity matrix
3:     $B^{HCS}$ = HCS($B, [m_2, m_3]$)
4:     $P$=$A^{HCS}$  $B^{HCS}$
5:     return ($P$)
6:
7: procedure DECOMPRESS-MP($P$)
8:     $C$ = zeros($n_1, n_2$)
9:     for i,j:=1 to $n_1, n_2$ do
10:        k = $h_{A1}[i]$
11:        l = $h_{B2}[j]$
12:        $C_{ij}$ = $s_{A1}[i]s_{B2}[j]P[k,l]$
13:    return ($C$)
```

B.2 Proofs of some technical theorems/lemmas

B.2.1 Analysis of CS and HCS approximation error

Theorem B.1 ([24]). *Given a vector $u \in \mathbb{R}^n$, **CS** hashing functions s and h with sketching dimension c, for any i^*, the recovery function $\widehat{u}_{i^*} = s(i^*)CS(u)(h(i^*))$ computes an unbiased estimator for u_{i^*} with variance bounded by $||u||_2^2/c$.*

Proof of Theorem B.1. For $i \in \{1, 2, \cdots n\}$, let K_i be the indicator variable for the event

$h(i) = h(i^*)$. We can write $\widehat{u}_{i^*}$ as

$$\widehat{u}_{i^*} = s(i^*) \sum_i K_i s(i) u_i \tag{B.1}$$

Observe that $K_i = 1$, if $i = i^*$, $E(s(i^*)s(i)) = 0$, for all $i \neq i^*$, and $E(s(i^*)^2) = 1$, we have

$$\begin{aligned} E(\widehat{u}_{i^*}) &= E(s(i^*)K_{i^*}s(i^*)u_{i^*}) + E(s(i^*)\sum_{i \neq i^*} K_i s(i) u_i \\ &= u_i \end{aligned} \tag{B.2}$$

To bound the variance, we rewrite the recovery function as

$$\widehat{u}_{i^*} = s(i^*)K_{i^*}s(i^*)u_{i^*} + s(i^*) \sum_{i \neq i^*} K_i s(i) u_i \tag{B.3}$$

To simplify notation, we assign X as the first term, Y as the second term. $\text{Var}(X) = 0$, and $COV(X, Y) = 0$ since $s(i)$ for $i \in \{1, 2, \cdots n\}$ are 2-wise independent. Thus,

$$\text{Var}(X + Y) = \sum_{i \neq i^*} \text{Var}(K_i s(i^*) s(i) u_i) \tag{B.4}$$

$E(K_i s(i^*) s(i) u_i) = 0$ for $i \neq i^*$. Consequently,

$$\text{Var}(K_i s(i^*) s(i) u_i) = E((K_i s(i^*) s(i) u_i)^2) = E(K_i^2) u_i^2 = u_i^2/c \tag{B.5}$$

The last equality uses that $E(K_i^2) = E(K_i) = 1/c$, for all $i \neq i^*$. Summing over all terms, we have $\text{Var}(\widehat{u}_{i^*}) \leq ||u||_2^2/c$. ■

Proof of Theorem 3.1. For simplicity, we assume $u \in \mathbb{R}^d$ is reshaped into a second-order tensor $A \in \mathbb{R}^{n_1 \times n_2}$ in the following proof. But the analysis can be extended to reshaping u into any order tensor.

For $i \in \{1, 2, \cdots n_1\}$, $j \in \{1, 2, \cdots n_2\}$, let K_{ij} be the indicator variable for the event $h_1(i) =$

$h_1(i^*)$ and $h_2(j) = h_2(j^*)$. We can write $\widehat{A}_{i^*j^*}$ as

$$\widehat{A}_{i^*j^*} = s_1(i^*)s_2(j^*)\sum_{ij} K_{ij}s_1(i)s_2(j)A_{ij} \tag{B.6}$$

Notice that $A = reshape(u)$, we know the index mapping: $A_{i^*j^*} = u_{t^*}$, where $t^* = n_2 i^* + j^*$. Observe that $K_{ij} = 1$, if $i = i^*$, $j = j^*$. $E(s_1(i^*)s_1(i)) = 0$, $E(s_2(j^*)s_2(j)) = 0$, for all $i \neq i^*$, $j \neq j^*$, and $E(s_1(i^*)^2) = 1$, $E(s_2(j^*)^2) = 1$, we have

$$\begin{aligned} E(\widehat{A}_{i^*j^*}) &= E(s_1^2(i^*)s_2^2(j^*)K_{i^*j^*}A_{i^*j^*} + E(s_1(i^*)s_2(j^*)\sum_{i\neq i^* or j\neq j^*} K_{ij}s_1(i)s_2(j)A_{ij}) \\ &= A_{i^*j^*} \end{aligned} \tag{B.7}$$

To bound the variance, we rewrite the recovery function as

$$\widehat{A}_{i^*j^*} = s_1^2(i^*)s_2^2(j^*)K_{i^*j^*}A_{i^*j^*} + s_1(i^*)s_2(j^*)\sum_{i\neq i^* or j\neq j^*} K_{ij}s_1(i)s_2(j)A_{ij} \tag{B.8}$$

To simplify notation, we assign X as the first term, Y as the second term. $\mathrm{Var}(X) = 0$, and $COV(X,Y) = 0$ since $s_1(i)$ and $s_2(j)$ for $i \in \{1, 2, \cdots n_1\}$, $j \in \{1, 2, \cdots n_2\}$ are both 2-wise independent. Thus,

$$\mathrm{Var}(X+Y) = \mathrm{Var}(X)+\mathrm{Var}(Y)-2\,\mathrm{Cov}(X,Y) = \sum_{i\neq i^* or j\neq j^*} \mathrm{Var}(K_{ij}s_1(i^*)s_2(j^*)s_1(i)s_2(j)A_{ij}) \tag{B.9}$$

$E(K_{ij}s_1(i^*)s_2(j^*)s_1(i)s_2(j)A_{ij}) = 0$ for $i \neq i^*$ or $j \neq j^*$. Therefore, Equation B.9 becomes:

$$\begin{aligned} \sum_{i\neq i^* or j\neq j^*} E((K_{ij}s_1(i^*)s_2(j^*)s_1(i)s_2(j)A_{ij})^2) &= \sum_{i\neq i^* or j\neq j^*} E(K_{ij}^2)A_{ij}^2 \\ &= \sum_{i\neq i^*, j\neq j^*} \frac{A_{ij}^2}{m_1 m_2} + \sum_{i\neq i^*, j=j^*} \frac{A_{ij}^2}{m_1} + \sum_{i=i^*, j\neq j^*} \frac{A_{ij}^2}{m_2} \end{aligned} \tag{B.10}$$

This is because $E(K_{ij}^2) = E(K_{ij}) = 1/(m_1 m_2)$, for all $i \neq i^*$, $j \neq j^*$. $E(K_{ij}^2) = E(K_{ij}) = 1/(m_1)$, for all $i \neq i^*$, $j = j^*$. $E(K_{ij}^2) = E(K_{ij}) = 1/(m_2)$, for all $i = i^*$, $j \neq j^*$.

If any fiber of A has extreme large data value, or $\max(\|A_i\|_2) \approx \|u\|_2$, where A_i is any row or column of A, we can omit the first term, $\mathrm{Var}(\widehat{A}_{i^*j^*}) \leq \|u\|_F^2 / (min(m_1, m_2))$. Otherwise, if $\|u\|_2 \gg \max(\|A_i\|_2)$, we can omit the second and third terms and $\mathrm{Var}(\widehat{A}_{i^*j^*}) = \Omega(\|u\|_2^2 / (m_1 m_2))$. ■

B.2.2 HCS of the Kronecker product

For simplicity, we show proof for Kronecker product here. But this can be extended to general tensor product.

Lemma B.2. *Given two matrices $A \in \mathbb{R}^{n\times n}$, $B \in \mathbb{R}^{n\times n}$,*

$$\begin{aligned} HCS(A \otimes B) &= HCS(A) * HCS(B) \\ &= IFFT2(FFT2(HCS(A)) \circ FFT2(HCS(B))) \end{aligned} \tag{B.11}$$

Proof of Lemma B.2. The Kronecker product defines $(A \otimes B)_{n_3(p-1)+h \; n_4(q-1)+g} = A_{pq} B_{hg}$. Thus:

$$\begin{aligned} &\sum_{pqhg} (A \otimes B)_{ab} s_1(p) s_2(q) s_3(h) s_4(g) w^{t_1 h_a + t_2 h_b} \\ &= \sum_{pqhg} A_{pq} B_{hg} s_1(p) s_2(q) s_3(h) s_4(g) w^{t_1 h_a + t_2 h_b} \\ &= \sum_{pq} A_{pq} s_1(p) s_2(q) w^{t_1 h_1(p) + t_2 h_2(q)} \sum_{hg} B_{hg} s_3(h) s_4(g) w^{t_1 h_3(h) + t_2 h_4(g)} \\ &= FFT2(HCS(A)) \circ FFT2(HCS(B)) \end{aligned} \tag{B.12}$$

where $a = n_3(p-1) + h$, $b = n_4(q-1) + g$, $h_a = h_1(p) + h_3(h)$, $h_b = h_2(q) + h_4(g)$.

Assign $i = n_3(p-1) + h$, $j = n_4(q-1) + g$, $s_5(i) = s_1(p) s_3(h)$, $s_6(j) = s_1(q) s_3(g)$, $h_5(i) =$

$h_1(p) + h_3(h)$ and $h_6(i) = h_2(q) + h_4(g)$, we have

$$\begin{aligned}
&\sum_{pqhg} (A \otimes B)_{ab} s_1(p) s_2(q) s_3(h) s_4(g) w^{t_1 h_a + t_2 h_b} \\
&= \sum_{ij} (A \otimes B)_{ij} s_5(i) s_6(j) w^{t_1 h_5(i) + t_2 h_6(j)} \\
&= FFT2(HCS(A \otimes B)) \\
&= FFT2(HCS(A)) \circ FFT2(MS(B)) \qquad \text{(B.13)}
\end{aligned}$$

Consequently, we have $HCS(A \otimes B) = IFFT2(FFT2(HCS(A)) \circ FFT2(HCS(B)))$. The recovery map is

$$\widehat{A \otimes B}_{n_3(p-1)+h \;\; n_4(q-1)+g} = s_1(p) s_2(q) s_3(h) s_4(g) HCS(A \otimes B)_{(h_1(p)+h_3(h)) \bmod m_1 \;\; (h_2(q)+h_4(g)) \bmod m_2} \qquad \text{(B.14)}$$

for $p \in [n_1]$, $q \in [n_2]$, $h \in [n_3]$, $g \in [n_4]$. ■

B.2.3 HCS of the matrix product

Higher-order tensor contraction can be seen as a matrix product by grouping all free indices and contraction indices separately. We show the proof for Lemma 3.3 in matrix case.

Lemma B.3. *Given two matrices $A \in \mathbb{R}^{n \times n}$, $B \in \mathbb{R}^{n \times n}$,* HCS$(A) = H_1(s_1 \otimes s_2 \circ A)H_2^T$*,* HCS$(B) = H_2(s_2 \otimes s_3 \circ B)H_3^T$*, then*

$$\text{HCS}(AB) = \text{HCS}(A)\text{HCS}(B) \qquad \text{(B.15)}$$

if $H_2^T H_2 = I$.

Proof of Lemma B.3. The compact HCS representations for A and B are HCS$(A) = H_1(s_1 \otimes s_2 \circ A)H_2^T$, HCS$(B) = H_2(s_2 \otimes s_3 \circ B)H_3^T$ as described in Section 3.4. Here $H_1 \in \mathbb{R}^{m_1 \times n_1}$, $H_2 \in \mathbb{R}^{m_2 \times r}$, $H_3 \in \mathbb{R}^{m_3 \times n_2}$, $s_1 \in \mathbb{R}^{n_1}$, $s_2 \in \mathbb{R}^r$ and $s_3 \in \mathbb{R}^{n_2}$. Assume HCS$(AB) = H_4(s_4 \otimes s_5 \circ AB)H_5^T$.

If H_2 is orthogonal, or $H_2^T H_2 = I$,

$$\begin{aligned} \text{HCS}(A)\text{HCS}(B) &= H_1(s_1 \otimes s_2 \circ A)H_2^T H_2(s_2 \otimes s_3 \circ B)H_3^T \\ &= H_1(s_1 \otimes s_2 \circ A)(s_2 \otimes s_3 \circ B)H_3^T \\ &= H_1(s_1 \otimes s_3 \circ AB)H_3^T \end{aligned} \tag{B.16}$$

By setting $H_4 = H_1$, $H_5 = H_3$, $s_4 = s_1$ and $s_5 = s_3$, we have HCS$(AB) = $ HCS(A)HCS(B). ■

B.2.4 Analysis of Kronecker product approximation error

Theorem B.4 (CS recovery analysis for Kronecker product). *Suppose $\widehat{C}$ is the recovered tensor for $C = A \otimes B$ after applying CS on $A \otimes B$ with sketching dimension c. We suppose the estimation takes d independent sketches of $A \otimes B$ and then report the median of the d estimates. If $d = \Omega(\log(1/\delta))$, $c = \Omega(\frac{\|C\|_F^2}{\epsilon^2})$, then with probability $\geq 1 - \delta$ there is $|\widehat{C}_{ij} - C_{ij}| \leq \epsilon$.*

Proof. CS$(C) = $ CS$(vec(A) \otimes vec(B))$. Given Theorem B.1, we have $E(\widehat{C}) = C = vec(A) \otimes vec(B)$, $\text{Var}(\widehat{C}_{ij}) \leq \|vec(A) \otimes vec(B)\|_2^2 / c = \|C\|_F^2 / c$. From Chebychev's inequality, if we run this sketch d times, where $d = \Omega(\log(1/\delta))$, we can get the desired error bond with probability at least $1 - \delta$. ■

Theorem B.5 (HCS recovery analysis for Kronecker product). *Suppose $\widehat{C}$ is the* *recovered tensor for $C = A \otimes B$ after applying HCS on $A \otimes B$ with sketching dimension m along each mode. We suppose the estimation takes d independent sketches of $A \otimes B$ and then report the median of the d estimates. If $d = \Omega(\log(1/\delta))$, $m^2 = \Omega(\frac{\|C\|_F^2}{\epsilon^2})$, then with probability $\geq 1 - \delta$ there is $|\widehat{C}_{ij} - C_{ij}| \leq \epsilon$.*

Proof. We have shown in Lemma B.2 that $\text{HCS}(C) = \text{HCS}(A) * \text{HCS}(B)$. Given Theorem 3.1, we have $E(\widehat{C}) = C = A \otimes B$, $\text{Var}(\widehat{C}_{ij}) \leq \|C\|_F^2 / m^2$ (We assume C is well-distributed). From Chebychev's inequality, if we run this sketch d times, where $d = \Omega(\log(1/\delta))$, we can get the desired error bond with probability at least $1 - \delta$. ■

B.2.5 Analysis of matrix product approximation error

Theorem B.6 (CS recovery analysis for matrix product). *Suppose $\widehat{C}$ is the recovered tensor for $C = AB$ after applying CS on AB with sketching dimension c. We suppose the estimation takes d independent sketches of AB and then report the median of the d estimates. If $d = \Omega(\log(1/\delta))$, $c = \Omega(\frac{\|C\|_F^2}{\epsilon^2})$, then with probability $\geq 1 - \delta$ there is $|\widehat{C}_{ij} - C_{ij}| \leq \epsilon$.*

Proof. $\text{CS}(C) = \sum_{i=1}^r \text{CS}(A_i \otimes B_i)$. Thus,

$$E(\widehat{C}) = \sum_{k=1}^{r} E(CS(A_k \otimes B_k)) = \sum_{k=1}^{r} A_k \otimes B_k = C \tag{B.17}$$

$$\begin{aligned}
\operatorname{Var}(\widehat{C}_{ij}) &= \sum_{k=1}^{r} \operatorname{Var}((\widehat{A}_{ik}\widehat{B}_{kj})) \\
&= \sum_{k=1}^{r} E^2(\widehat{A}_{ik}) \operatorname{Var}(\widehat{B}_{kj}) + E^2(\widehat{B}_{kj}) \operatorname{Var}(\widehat{A}_{ik}) + \operatorname{Var}(\widehat{A}_{ik}) \operatorname{Var}(\widehat{B}_{kj}) \\
&\leq \sum_{k=1}^{r} A_{ik} \left\|B_k\right\|_2^2 / c + B_{kj} \left\|A_k\right\|_2^2 / c + \left\|A_k\right\|_2^2 \left\|B_k\right\|_2^2 / c^2 \qquad \text{(B.18)} \\
&\leq \sum_{k=1}^{r} \left\|A_k\right\|_2^2 \left\|B_k\right\|_2^2 (\frac{1}{c} + \frac{1}{c^2}) \\
&\leq 3 \left\|AB\right\|_F^2 / c
\end{aligned}$$

From Chebychev's inequality, if we run this sketch d times, where $d = \Omega(\log(1/\delta))$, we can get the desired error bond with probability at least $1-\delta$. ■

Theorem B.7 (**HCS recovery analysis for matrix product**). *Suppose $\widehat{C}$ is the recovered tensor for $C = AB$ after applying HCS on AB with sketching dimension m along each mode. We suppose the estimation takes d independent sketches of AB and then report the median of the d estimates. If $d = \Omega(\log(1/\delta))$, $m^2 = \Omega(\frac{\|C\|_F^2}{\epsilon^2})$, then with probability $\geq 1-\delta$ there is $|\widehat{C}_{ij} - C_{ij}| \leq \epsilon$.*

Proof. We have shown in Section 3.5.2 that $\text{HCS}(AB) = \text{HCS}(A)\text{HCS}(B)$. Given Theorem 3.1, we have $E(\text{HCS}(AB)) = AB$, $\operatorname{Var}(\widehat{AB}_{ij}) \leq \left\|AB\right\|_F^2 / m^2 = \left\|C\right\|_F^2 / m^2$. From Chebychev's inequality, if we run this sketch d times, where $d = \Omega(\log(1/\delta))$, we can get the desired error bond with probability at least $1-\delta$. ■