Speeding Up Optimizations via Data Structures: Faster Search, Sample and Maintenance

Lichen Zhang

CMU-CS-22-107

May 11, 2022

School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213

Thesis Committee:
Gary Miller, Chair
Anil Ada
Zhao Song, Adobe Research

Submitted in partial fulfillment of the requirements for the degree of Master of Computer Science.

Keywords: Data Structure, Discrete Optimization, Continuous Optimization

To my family and loved ones.

Abstract

In this thesis, we present novel techniques to solve various fundamental discrete and continuous optimization problems, with the deployment of highly-efficient data structures.

- **Sparsification:** We obtain fast deterministic and randomized algorithms for spectral sparsification and its variants. We give a deterministic algorithm for constructing linear-sized spectral sparsifier in time $O(d^{\omega+1})$ where $\omega \approx 2.37$ is the exponent of matrix multiplication, breaking the $\Omega(d^4)$ barrier for all prior deterministic methods.
- **Non-Convex Optimization:** We present the first algorithm to train deep and over-parametrized neural networks in truly sub-quadratic time. It has a training cost of $O(m^{2.25-\alpha})$, where m is the width of the network and $\alpha \approx 0.31$ is the dual exponent of matrix multiplication.

The main theme of these major improvements is the novel adaption of data structures in different iterative processes. We show that for different optimization problems, we can frame their iterations as solving certain data structure problems. We design different data structures that are efficient and adaptively robust to realize such speedup.

Acknowledgments

First and foremost, I would like to thank my mentor and advisor, Professor Gary Miller. Since I first took his class 3 years ago, Gary has been an important source of inspiration, wisdom and support. He has given many very helpful insights to the problems I tried to solve and solved, and provided guidance on how to approach research as a whole. I'll be forever thankful for working with Gary as my entry to the grand scene of theoretical computer science research.

Next, I would like to thank Zhao Song, who I have been working with for almost two years. Zhao is the most hardworking and diligent person I have ever known, and his expertise in almost all fields in theoretical computer science has always amazed me and inspired me to work harder. He also provided me many invaluable guidance on how to conduct research and pursue academic goals. I'm very fortunate to be working with him, and growing from our collaborations.

I would like to thank Professor Pravesh Kothari, who I have been his teaching assistants for both his "Undergraduate Complexity Theory" and "The Computational Lens". Though our research interests are not the closet, Pravesh has always been a great mentor for me on important life decisions and how to approach the applications to graduate school as a whole. I'm grateful for his very helpful advice on how to be a better researcher.

I would like to thank Professor Anil Ada for being my committee member, and I really enjoyed being a teaching assistant for his class "The Computational Lens". Anil's passion on teaching always intrigues me and inspires me to structure a new way to interact with students and help them learning.

I would also like to thank all my coauthors throughout the year. This thesis would not be what it is without them. For the joint work that is reflected in this thesis, thank you to Zhao Song, Danyang Zhuo, Ruizhe Zhang, Zhaozhuo Xu, Yuanyuan Yang and Lianke Qin. Thank you to Gary Miller, Pravesh Kothari, Jonathan Kelner and Hengjie Zhang for very helpful discussions for many of my problems.

Finally, I would like to thank my family and my girlfriend Yuke Lin for their boundless support and love. Without their guidance, patience and encouragement I would not be able to chase my dream and complete the works included in this thesis.

Contents

List of Tables

Chapter 1

Introduction

Data structures are perhaps the most important and widely-studied objects in computer science. In fact, every aspect and subfield of computer science heavily replies on data structures. Data structures also play critical roles in advancing theoretical computer science, and many recent breakthroughs in developing fast algorithms come from the smart deployment of data structures. In this thesis, we show how to improve the running time of different discrete and continuous optimization processes using efficient data structures.

Specifically, we study the following important theoretical problems and give an overview of their history and improvements.

Linear-Sized Spectral Sparsification and its Varaints. Given a matrix $V \in \mathbb{R}^{m\times d}$ with $m \gg d$, the goal is to compute a matrix $\widetilde{V} \in \mathbb{R}^{s\times d}$ with $s \ll m$ whose rows are a subset of re-scaled rows of V, with the guarantee that

$$(1-\varepsilon)V^\top V \preceq \widetilde{V}^\top \widetilde{V} \preceq (1+\varepsilon)V^\top V,$$

this is the well-known spectral sparsifier problem. Using leverage score sampling, [80] shows that it suffices to have $s = \Theta(\varepsilon^{-2}d\log d)$. The seminal work by Batson, Spielman and Srivastava [12] further improves this s to $\Theta(\varepsilon^{-2}d)$, which is essentially optimal. However, for the most common setting in which $m = d^2$, their algorithm takes $\Omega(d^5)$ time. [90] improves the deterministic running time to $O(d^4)$ by introducing a deterministic procedure to generate a sparsifier of size $\varepsilon^{-2}d\log d$, then run the [12] algorithm. To further speed up the construction, [3, 53] makes use of a new variant of potential functions, and [54] finally settles down this problem by providing a randomized algorithm that runs in time $\widetilde{O}(\varepsilon^{-2}m)$ for graphs and $O(\varepsilon^{-2}\mathrm{nnz}(V^\top V))$ for general matrices.

While efficient, most of these recent developments highly rely the use of randomness and it is unclear how to derandomized their methods and preserving the efficiency in the meantime. On the other hand, it is important to develop fast deterministic algorithms for spectral sparsifier, since it is much easier to turn a static algorithm to handle updates to the matrix, and obtaining a nearly linear time deterministic algorithm will also lead to breakthroughs in computing deterministic minimum weighted cut problem.

Besides linear-sized spectral sparsifier, the potential function in [12] has wide range of applications in other problems, such as restricted invertibility [81], one-sided vector packing [84],

experimental design [7, 51] and Fourier signal interpolation [77].

Training Over-parametrized Neural Networks. Machine learning and particularly deep learning is one of the most popular topics in modern computer science. Despite of its empirical success, it has always been a mystery from a theoretical perspective, that why deep learning works so well. Recently, via the power of over-parametrization, a long line of works [5, 6, 31, 32, 38, 41, 52, 56, 71, 75] show that popular first order methods such as gradient descent and stochastic gradient descent converge on such architecture of networks. Second order methods provide even faster convergence rate for two-layer over-parametrized ReLU networks [17, 18, 89]. However, such methods typically suffer from slow training time, due to the over-parametrized nature of the network, which means the network width $m = \mathrm{poly}(n)$ where n is the number of data points. In fact, for deep networks, it seems one has to pay $\Omega(m^2)$ per training iteration, since the weight matrix itself has size $m \times m$.

For training shallow ReLU networks, [75] has shown how to obtain a sublinear training time per iteration using some high dimensional geometric search data structure. However, the main reason they can obtain sublinear running time is due to the size of the weight matrix is only $m \times d$ for $d \ll m$ being the data dimension. Moreover, their algorithms suffer from exponential initialization time due to the data structures they use. For second order methods, [17] uses sketching for preconditioning to realize a nearly linear time algorithm for shallow ReLU networks. However, when adapting their methods for deep networks, the running time cannot bypass $\Omega(m^2)$.

Linear Programming and Robust Interior Point Method. Linear programmings have been studied for nearly a century. One of the first and most popular LP algorithm is the simplex algorithm [28]. Despite it works well in practical small size problems, the simplex algorithm is known to be an exponential time algorithm in the worst case of Klee-Minty cube [50]. The first polynomial time algorithm for solving LP is the ellipsoid method [49]. Although this algorithm runs in polynomial time theoretically, but in practice this algorithm runs much slower than the simplex algorithm. The interior point type of methods [47] have both polynomial running time in theory and fast and stable performance in practice. In the case of $d = \Omega(n)$, the time complexity of Karmarkar's algorithm is $O^*(n^{3.5})$. In the work [65, 82], the time complexity was further improved to $O^*(n^3)$. In 1989, Vaidya further proposed an algorithm that takes a running time of $O^*(n^{2.5})$ [83]. This result hasn't been improved for three decades until recent work due to Cohen, Lee and Song [27], which gives an algorithm that runs in time $O^*(n^\omega + n^{2.5-\alpha/2} + n^{2+1/6})$, where ω denotes the exponent of matrix multiplication [8], and α denotes the dual exponent of matrix multiplication [33]. Currently $\omega \approx 2.373$ and $\alpha \approx 0.31$. By using a more sophisticated analysis and data structure, Jiang, Song, Weinsten and Zhang further improve this running time to $O^*(n^\omega + n^{2.5-\alpha/2} + n^{2+1/18})$ [42]. Besides the square LP case $d = \Omega(n)$, there are also a line of work focusing on studying the flat LP case $d \ll n$, for instance [25].

The breakthrough result by Cohen, Lee and Song [27] can be roughly described as follows: given a projection matrix $P := \sqrt{W}A^\top (AWA^\top)^{-1} A\sqrt{W}$ for $A \in \mathbb{R}^{d\times n}$ be a fixed constraint matrix and a diagonal matrix $W \in \mathbb{R}^{n\times n}$ with non-negative entries, they provide an algorithm that maintains P in a lazy fashion when W undergoes ℓ_2 multiplicative change. Specifically, by utilizing fast rectangular matrix multiplication, they get a per iteration cost of $O^*(n^{\omega-0.5} +$

$n^{2-\alpha/2}$). Further, they need to multiply P with a (possibly dense) vector $h \in \mathbb{R}^n$. To speed up this process, they use an importance sampling method to sparsify the vector h, so that it only has $\sqrt{n}$ nonzero entries in expectation. To facilitate the analysis, they introduce a novel framework called stochastic central path, which gives comparable convergence rate as classical central path.

Following their seminal work, many alternative approaches have been proposed to "sparsify" the vector h or reduce its dimension directly. In [55] and [72], they use randomized sketching techniques to reduce the size of either the projection matrix or the vector directly, and achieve similar result of [27].

1.1 Data Structures: Tasks and Results

Data structures are integral to speed up iterative processes. Many optimization algorithms can be formulated as solving certain data structure tasks. In this section, we start with a list of tasks, and give various data structures for solving these tasks.

1.1.1 Data Structure Tasks

Many of the tasks relate to inner product — perhaps, search, sample and estimate inner product is one of the most important components in the series problems we study.

References	Sa/Se/Es	Main Task	Special Task	Theorem	Chapter
[12, 78]	Search	Task 1.1.1	Task 1.1.4	Theorem 1.2.1	Chapter 3
[75]	Search	Task 1.1.1	Task 1.1.4	Theorem 1.2.6	Chapter 1
[53, 54]	Sample	Task 1.1.2	Task 1.1.5	Theorem 3.1.9	Chapter 3
[27, 72]	Estimate	Task 1.1.3	Task 1.1.9	Theorem 2.6.12	Chapter 2
[55]	Estimate	Task 1.1.3	Task 1.1.9	Theorem 2.6.10	Chapter 2

Table 1.1: Optimization algorithms and their corresponding data structure tasks: inner product search/sample/estimate. We use **Sa** to denote Sample, **Se** to denote Search and **Es** to denote Estimate.

General Inner Product Tasks We give a list of general tasks related to inner product. Roughly, it can be categorized into inner product search, which, suppose the input dataset satisfies some constraints on inner product, then give a query matrix, we search a vector that satisfies the desired property.

Inner product sample is similar to search, but instead, one wants to sample from a distribution defined by the inner product of vectors in the dataset and the query.

Another task is to estimate inner product. Given a query vector, we are asked to (approximately) estimate the inner product between query with one or more vectors in the dataset.

Task 1.1.1 (General Inner Product Search)**.** Let $X = \{x_1, \ldots, x_m\} \subset \mathbb{R}^d$ be a collection of vectors. Given a query matrix $Q \in \mathbb{R}^{d \times d}$ with some promised condition $g(X, Q)$, we need to output a vector x_i such that $f(i, X, Q)$ holds.

Inner product search has wide range of applications, we will see later that constructing deterministic spectral sparsification [12] relies on it. Vector packing [84] and experimental design [7] can also be formulated as inner product search. Interesting enough, we also see its applications in training over-parametrized networks [75].

Task 1.1.2 (General Inner Product Sample)**.** Let $X = \{x_1, \ldots, x_m\} \subset \mathbb{R}^d$ be a collection of vectors. Given a query matrix $Q \in \mathbb{R}^{d \times d}$ with some promised condition $g(X, Q)$, we need to sample a vector x_i with probability $f(i, X, Q)$.

Sampling from the distribution defined by the inner product is core to many problems, such as randomized linear-sized spectral sparsifier [53, 54], which is essential to combat the potential presence of the noise in the model [20]. We believe this task also captures sampling from symmetric Determinantal Point Process (DPP) and Non-symmetric Determinantal Point Process (NDPP) distributions.

Task 1.1.3 (General Inner Product Estimation)**.** Let $X = \{x_1, \ldots, x_m\} \subset \mathbb{R}^d$ be a collection of vectors. Let $\varepsilon \in (0, 1)$ be a parameter. Given a query vector $q \in \mathbb{R}^d$ with some promised condition $g(X, q)$, we need to output some estimates est_i such that $f(i, X, q, \varepsilon)$ holds.

At first glance, this task is rather abstract. However, note that the matrix-vector product can be interpreted as estimating all the inner products: given $V \in \mathbb{R}^{m \times d}$ and $q \in \mathbb{R}^d$, estimating all inner products can be implemented trivially as computing Vq in time $O(md)$. So this task can be alternatively viewed as computing the matrix-vector product approximately and efficiently.

Note that this is a more general task than the prior two: once we estimated all inner products, we can just scan through all results to implement the search or sample.

Due to its generality, this task facilitates solving linear programs fast [27, 72] and empirical risk minimization [55].

Special Instances We localize the discussion to specific tasks corresponding to the general inner product tasks we defined above.

Task 1.1.4 (Special case of Task 1.1.1, Non-negative Inner Product Search)**.** Let $X = \{x_1, \ldots, x_m\} \subset \mathbb{R}^d$ be a collection of vectors. Given a query matrix $Q \in \mathbb{R}^{d \times d}$ with $g(X, Q)$ being the predicate $\sum_{i \in [m]} x_i^\top Q x_i \geq 0$, we need to output a vector x_i such that $f(i, X, Q)$ is the predicate that $x_i^\top Q x_i \geq 0$.

We will show the above task directly captures the iterative process of deterministic linear-sized spectral sparsifier [12].

Task 1.1.5 (Special case of Task 1.1.2, Inner Product Sample)**.** Let $X = \{x_1, \ldots, x_m\} \subset \mathbb{R}^d$ be a collection of vectors. Given a positive semidefinite matrix $Q \in \mathbb{R}^{d \times d}$ and $g(X, Q)$ is the predicate that Q is PSD, we need to sample a vector x_i with probability $f(i, X, Q) = \frac{x_i^\top Q x_i}{\sum_{i \in [m]} x_i^\top Q x_i}$.

Sampling based on a distribution defined by the inner product is key to randomized linear-sized spectral sparsifier [53, 54].

Task 1.1.6 (Special case of Task 1.1.1, Minimum Inner Product Search)**.** Let $X = \{x_1, \ldots, x_m\} \subset \mathbb{R}^d$. Given a query matrix $Q \in \mathbb{R}^{d \times d}$ with $g(X, Q)$ being that $\forall i \in [m], x_i^\top Q x_i \geq 0$, we need to find an x_i with $f(i, X, Q) := \arg\min_{x \in X} x^\top Q x$.

Given a set of vectors, the goal is to find the one that has the minimum inner product with the query vector. Using certain inner product-preserving transformation [59], this problem is dual to

the furthest neighbor search problem, where one wants to find the vector that has the maximum ℓ_2 distance with the query vector. This task has classical data structures to realize [39], but as we will show, it is nontrivial to augment classical data structures for iterative process. Surprisingly enough, we also show estimating all inner products is a good approach.

Minimum inner product search captures the iterative process for vector packing [84] and experimental design [7]. In general, when one tries to use a one-sided barrier potential function, it becomes useful.

Task 1.1.7 (Special case of Task 1.1.1, Inner Product Search with Prior Knowledge)**.** Let $X = \{x_1, \ldots, x_n\} \subset \mathbb{R}^d$ and $W = \{w_1, \ldots, w_m\} \subset \mathbb{R}^d$. For any $Q = x_i$ for $i \in [m]$, we need to find all w_j's with $f(j, W, Q)$ being $\langle w_j, Q \rangle \geq \tau$ for some threshold $\tau \geq 0$.

This task is related to the training of neural network under ReLU or similar activations. After applying ReLU, only a subset of neurons is nonzero, and the training consists of returning the set of nonzero neurons efficiently [75]. The inner product $\langle w_j, x_i \rangle$ can be viewed as multiplying the weight with the data point, which is key in neural network inference.

Task 1.1.8 (Special case of Task 1.1.2, Inner Product Sample with Prior Knowledge)**.** Let $X = \{x_1, \ldots, x_m\} \subset \mathbb{R}^d$ and $\mathsf{Q} = \{Q_1, \ldots, Q_n\} \subset \mathbb{R}^{d \times d}$ with $g(X, Q_i)$ being Q_i is positive semidefinite for $i \in [n]$, at query time, we are given a fixed Q_j and we need to sample x_i with probability $f(i, X, Q_j) = \frac{x_i^\top Q_j x_i}{\sum_{i \in [m]} x_i^\top Q_j x_i}$.

For sampling from DPP or NDPP distributions, which are defined on the determinant of certain sub-matrix, one can generate samples one at a time. By using rank-1 determinant formula, this can also be formulated as an inner product sample task. In contrast to other applications, the query quantity comes from a sequence we know in advance: it is a linear combination of the input vectors. Hence, one can use this knowledge to build up faster data structures.

Task 1.1.9 (Special case of Task 1.1.3, Approximate Inner Product Estimation)**.** Let $X = \{x_1, \ldots, x_m\} \subset \mathbb{R}^d$ and $\varepsilon \in (0, 1)$. Given a query vector $q \in \mathbb{R}^d$, we need to output all estimates est_i for all $i \in [m]$ with $f(i, X, q, \varepsilon)$ being $\mathrm{est}_i = \langle x_i, q \rangle \pm \varepsilon \|x_i\|_2 \|q\|_2$.

This is perhaps the most important inner product task, since it captures estimating the matrix-vector product approximately. All prior tasks can be reduced to solve this task, and itself is crucial for recent breakthroughs in linear programming [27, 72] and empirical risk minimization [55].

Matrix Maintenance Tasks We discuss two tasks related to maintain certain matrices. Many optimization algorithms require maintain matrices under some (slow) changes, or maintain some low rank structure of the change.

References	LR/Proj	Task	Theorem	Chapter
[76]	LR	Task 1.1.10	Theorem 1.2.4	Chapter 4
[27, 55, 72]	Proj	Task 1.1.11	Theorem 2.6.10 and 2.6.12	Chapter 2

Table 1.2: Optimization algorithms and their corresponding data structure tasks: low rank or projection maintenance. We use **LR** to denote Low Rank and **Proj** to denote Projection

Task 1.1.10 (Low Rank Maintenance)**.** Let $A \in \mathbb{R}^{d \times d}$ and $U, V \in \mathbb{R}^{d \times k}$ for some $k < d$. The goal is to maintain the matrix $A + UV^\top$ so that for any vector $x \in \mathbb{R}^d$, the matrix-vector product

query $(A + UV^\top)x$ can be answered fast.

The above task is inspired by training deep, large neural networks [76], in which the gradient of the weight matrix has a natural low rank structure, and the network inference requires to compute the matrix-vector product with a (sparse) vector.

Task 1.1.11 (Projection Maintenance)**.** Let $W \in \mathbb{R}^{n \times n}$ be a diagonal matrix with non-negative entries, $A \in \mathbb{R}^{d \times n}$ with $d \leq n$ and rank d. The goal is to maintain the projection matrix $P = \sqrt{W}A^\top (AWA^\top)^{-1} A\sqrt{W}$ under the change to diagonal matrix W, so that for any vector $h \in \mathbb{R}^n$, the matrix-vector product query Ph can be answered fast.

Projection maintenance is fundamental for efficiently implementing the central path method, which is the crux for interior point method (IPM). Problems such as linear programming [27, 72] and empirical risk minimization crucially rely on fast IPM implementations.

1.1.2 Data Structure Results

In this section, we give our main data structures, addressing tasks we propose in the previous section.

Data Structures	Special Task	Theorem	Chapter
Matrix Search Tree	Task 1.1.4	Theorem 1.1.12	Chapter 2.2.1
Matrix Sample Tree	Task 1.1.5	Theorem 1.1.12	Chapter 2.2.1
Vector Search Tree	Task 1.1.4	Theorem 1.1.13	Chapter 2.2.2
Vector Sample Tree	Task 1.1.5	Theorem 1.1.13	Chapter 2.2.2
AFN+TensorSparse	Task 1.1.6	Theorem 1.1.14	Chapter 2.3
AIPE	Task 1.1.6	Theorem 1.1.15	Chapter 2.4
Correlation DTree	Task 1.1.7	Theorem 1.1.16	Chapter 2.2.3
Correlation WTree	Task 1.1.7	Theorem 1.1.17	Chapter 2.2.3
Coordinate-wise Embedding	Task 1.1.9	Theorem 1.1.20	Chapter 2.6.2
Low Rank Maintenance	Task 1.1.10	Theorem 1.1.18	Chapter 2.5
Projection Maintenance	Task 1.1.11	Theorem 2.6.10 and 2.6.12	Chapter 2.6

Table 1.3: Our data structures that fulfill different tasks.

Matrix and Vector Search & Sample Tree. Our first data structures are surprisingly simple yet highly effective trees that solves Task 1.1.4 and 1.1.5. They have good initialization and query time that can be balanced based on the structure of the input. We state their guarantees here.

Theorem 1.1.12 (Matrix Search & Sample Tree, informal version of Theorem 2.2.1)**.** *There exists a data structure with the following procedures:*

- INIT($\{M_1, M_2, \cdots, M_m\} \subseteq \mathbb{R}^{d \times d}$). *It takes a sequence of matrices $M_1, M_2, \cdots, M_m$ as input, and preprocesses in time $O(\sum_{i=1}^m \mathrm{nnz}(M_i))$.*
- QUERYPOSITIVESEARCH[1]$(A \in \mathbb{R}^{d \times d})$. *Given a matrix A with the promise that $\sum_{i=1}^m \langle M_i, A\rangle > 0$, it returns an index i such that $\langle M_i, A\rangle > 0$ in time $O(d^2 \log m)$.*

[1]Throughout this thesis, we use the phrase "positive search" and "nonnegative search" interchangeably.

- QUERYSAMPLE($A \in \mathbb{R}^{d \times d}$). *Given a positive semidefinite matrix $A \in \mathbb{R}^{d \times d}$, it samples an index i with probability $\frac{\langle M_i, A \rangle}{\sum_{i=1}^m \langle M_i, A \rangle}$ in time $O(d^2 \log m)$.*

The above data structure has very fast query time, since reading the input matrix A will take $O(d^2)$ time for a dense A, and when the preprocessed dataset $\{M_1, \ldots, M_m\}$ are rather sparse, it also has fast initialization time.

Theorem 1.1.13 (Vector Search & Sample Tree, informal version of Theorem 2.2.4)**.** *There exists a data structure with the following procedures:*

- INIT($\{v_1, v_2, \cdots, v_m\} \subseteq \mathbb{R}^d$). *It takes a sequence of vectors $v_1, v_2, \cdots, v_m$ as input, and preprocesses in time $O(md^{\omega-1})$ and in space $O(md)$.*
- QUERYSAMPLE($A \in \mathbb{R}^{d \times d}$). *Given a positive semidefinite matrix $A \in \mathbb{R}^{d \times d}$, it samples an index i with probability $\frac{\langle M_i, A \rangle}{\sum_{i=1}^n \langle M_i, A \rangle}$ in time $O(d^2 \log m + d^\omega)$.*
- QUERYPOSITIVESEARCH($A \in \mathbb{R}^{d \times d}$). *Given a matrix A with the promise that $\sum_{i=1}^m \langle M_i, A \rangle > 0$, it returns an index i such that $\langle M_i, A \rangle > 0$ in time $O(d^2 \log m + d^\omega)$.*

When the dataset is given as a set of vectors, the above data structure gives a faster initialization time at the expense of a worse query time. Unlike traditional data structures, this is acceptable — as we will see later, the optimization task we consider has d^ω barrier per iteration due to computing some full rank updated inverse. Hence, the slower query time actually gives us leverage to design algorithm that has overall good running time.

We remark the two data structures do not have any internal randomness, this means they are automatically robust against adaptive queries. Using these simple data structures to reduce iteration cost resembles a similarity to that of [48].

Approximate Furthest Neighbor and Sparse Embedding for Tensors. To solve Task 1.1.6, i.e., finding the vector with the minimum inner product, it is natural to consider its dual problem, the furthest neighbor search. By using a reduction that normalizes vector while preserving inner product, we solve Task 1.1.6 using the AFN data structure of [39]. However, there are two caveats: 1). the data structure is not robust against adaptive adversary, 2). the data structure is typically used with Johnson-Lindenstrauss transform, which itself is not robust.

We address these two problems individually. For the first one, we use a quantization method to make it robust. For the second one, we design a new type of sparse embedding matrix for tensors, and augment it to be robust. Our result can be viewed as a practice to equip classical data structures with the arsenals for iterative process.

Theorem 1.1.14 (Informal version of Theorem 2.3.24)**.** *Let $c, \tau, \varepsilon, \delta \in (0, 1)$ and $X = \{x_1, \ldots, x_m\} \subset \mathbb{R}^d$.*

Let $k = \widetilde{O}(d^2)$ be the number of independent TensorSparse *matrices. Then there exists a randomized data structure with success probability at least $1 - \delta$ even against an adaptive sequence of queries, such that given a query matrix $Q \in \mathbb{R}^{d \times d}$ with $\min_{x \in X} x^\top Q x \leq \tau$, the data structure outputs a vector $\widehat{x} \in X$ such that $\widehat{x}^\top Q \widehat{x} \leq \tau/c + o(1)$. Moreover if $c \in (\tau, \frac{400\tau}{(1-\varepsilon)^2\tau + 2\varepsilon + 399})$, the data structure has the following runtime behaviors:*

- *It preprocesses X in time $\widetilde{O}((m^{1.01} + \mathrm{nnz}(X))d^2)$.*
- *Given a query matrix Q, it outputs a $\widehat{x} \in X$ with the promised guarantee in time $\widetilde{O}(m^{0.01} + d^2)$.*

- *It inserts or deletes a point into the dataset in time $\widetilde{O}(m^{0.01}d^2)$.*

Approximate Inner Product Estimation. An alternative approach for Task 1.1.6 is to simply estimate all the inner products approximately. Note data structure with such properties can also be used for Task 1.1.9, but the AIPE data structure uses many independent JL matrices (the number is proportional to the dimension), hence when iteration is small (which is common in optimization), it does not have the best performance.

To produce such data structure, we again use the duality between inner product and norm, and adapt the adaptive distance estimation data structures [22, 23].

Theorem 1.1.15 (Informal version of Theorem 2.4.6)**.** *Let $c, \tau, \delta \in (0, 1)$. Given $X = \{x_1, \ldots, x_m\} \subset \mathbb{R}^d$ and $Q \in \mathbb{R}^{d \times d}$ with $\min_{x \in X} x^\top Q x \leq \tau$, the data structure outputs a vector $\widehat{x} \in X$ such that $\widehat{x}^\top Q x \leq \tau / c$. Moreover, if $c \in (\tau, \frac{1.01\tau}{\tau + 0.99})$, the data structure has the following runtime behaviors:*

- *It preprocesses X in time $\widetilde{O}(md^2)$.*
- *Given a query matrix Q, it outputs a $\widehat{x} \in X$ with the promised guarantee in time $\widetilde{O}(m+d^2)$.*
- *It inserts or deltes a point into the dataset in time $\widetilde{O}(d^2)$.*

Correlation Trees. Task 1.1.7 gives us two datasets X and W, then we need to preprocess one of them, using the other as query dataset. One of the datasets needs to be constantly updated. This task is closely related to training over-parametrized neural networks, in which one needs to find the neurons being fired up, which means $\langle w_j, x_i \rangle \geq \tau$ for some τ.

The data structures we design is called the *correlation tree*, basically, we preprocess either X or W, storing the maximum inner product at each node. During query, we only search the subtree with the value at node at least as large as the threshold. Hence, we only touch the subtree that contains fired-up neurons.

Theorem 1.1.16 (Correlation DTree, informal version of Theorem 2.2.9)**.** *There exists a data structure with the following procedures:*

- INIT($\{w_1, w_2, \cdots, w_m\} \subset \mathbb{R}^d, \{x_1, x_2, \cdots, x_n\} \subset \mathbb{R}^d, n \in \mathbb{N}, m \in \mathbb{N}, d \in \mathbb{N}$). *Given a series of weights $w_1, w_2, \cdots, w_m$ and datas $x_1, x_2, \cdots, x_n$ in d-dimensional space, it preprocesses in time $O(nmd)$*
- UPDATE($z \in \mathbb{R}^d, r \in [m]$). *Given a weight z and index r, it updates weight w_r with z in time $O(n \cdot (d + \log m))$*
- QUERY($i \in [n], \tau \in \mathbb{R}$). *Given an index i indicating data point x_i and a threshold τ, it finds all index $r \in [m]$ such that $\langle w_r, x_i \rangle > \tau$ in time $O(|\widetilde{S}(\tau)| \cdot \log m)$, where $\widetilde{S}(\tau) := \{r : \langle w_r, x_i \rangle > \tau\}$*

The second data structure preprocesses all data points and supports queries in terms of weights:

Theorem 1.1.17 (Correlation WTree, informal version of Theorem 2.2.13)**.** *There exists a data structure with the following procedures:*

- INIT($\{w_1, w_2, \cdots, w_m\} \subset \mathbb{R}^d, \{x_1, x_2, \cdots, x_n\} \subset \mathbb{R}^d, n \in \mathbb{N}, m \in \mathbb{N}, d \in \mathbb{N}$). *Given a series of weights $w_1, w_2, \cdots, w_m$ and datas $x_1, x_2, \cdots, x_n$ in d-dimensional space, it preprocesses in time $O(nmd)$*

- UPDATE($z \in \mathbb{R}^d, r \in [m]$). *Given a weight z and index r, it updates weight w_r with z in time $O(nd)$*
- QUERY($r \in [m], \tau \in \mathbb{R}$). *Given an index r indicating weight w_r and a threshold τ, it finds all index $i \in [n]$ such that $\langle w_r, x_i \rangle > \tau$ in time $O(|S(\tau)| \cdot \log m)$, where $S(\tau) := \{i : \langle w_r, x_i \rangle > \tau\}$*

Low Rank Maintenance via Lazy Update. Task 1.1.10 describes the following simple problem: given a matrix $A \in \mathbb{R}^{d \times d}$, at each iteration, it receives an update $U_k V_k^\top$ with $U_k, V_k \in \mathbb{R}^{d \times k}$ for some $k \ll d$, we need to maintain $A + U_k V_k^\top$ so that matrix-vector product can be computed fast. Suppose one has computed Ah for some $h \in \mathbb{R}^d$, computing $U_k V_k^\top h$ only takes $O(kd)$ time. However, when the number of updates becomes large, this k grows larger and larger, and we might need to restart. The following data structure implements this idea.

Theorem 1.1.18 (Informal version of Lemma 2.5.2)**.** *There exists a deterministic data structure such that, given an initial matrix $A \in \mathbb{R}^{d \times d}$, has the following guarantees:*

- *It preprocesses A in time $O(d^2)$.*
- *Given an update $U_k, V_k \in \mathbb{R}^{d \times k}$, it updates the representation $A + \Delta A + U_k V_k^\top$ in amortized time $O(kd^{2-\alpha+o(1)})$, where $\alpha \approx 0.31$ is the dual matrix multiplication exponent [33].*
- *Given a vector $h \in \mathbb{R}^d$, it computes the product $(\Delta A)h$ in time $O(d \cdot (\mathrm{nnz}(h) + r))$, where r is the rank of ΔA when it is queried.*

Projection Maintenance via Inverse Maintenance and Coordinate-wise Embedding. Projection maintenance in its original form is not obvious how to achieve it efficient, due it its rather complicated structure. We simplify it via a reduction to inverse maintenance, this reduction is general enough to take into the account of the sketching matrices to speed up the construction.

The construction itself uses Schur complement, basically, for proper matrices A, B, C, D, we have

$$\begin{bmatrix} A & B \\ C & D \end{bmatrix}^{-1} = \begin{bmatrix} A^{-1} + A^{-1}B(D - CA^{-1}B)^{-1}CA^{-1} & -A^{-1}B(D - CA^{-1}B)^{-1} \\ -(D - CA^{-1}B)^{-1}CA^{-1} & (D - CA^{-1}B)^{-1} \end{bmatrix}.$$

Assume all corresponding parts that require inversion is invertible. By properly putting the matrices into correct location, it is not hard to see that

$$\begin{bmatrix} W^{-1} & A^\top & \sqrt{W}^{-1} & 0 \\ A & 0 & 0 & 0 \\ 0 & 0 & -I & 0 \\ (\sqrt{W}^{-1})^\top & 0 & 0 & -I \end{bmatrix}^{-1} \begin{bmatrix} 0_n \\ 0_d \\ -h \\ -h \end{bmatrix} = \begin{bmatrix} \star \\ \star \\ \star \\ \sqrt{W}A^\top (AUA^\top)^{-1} A \sqrt{W} h \end{bmatrix}$$

For a more detailed discussion, we refer readers to Section 2.6.

As observed in [27], even if the projection matrix P is given us for free, computing Ph might take $O(n^2)$ assuming no structures on P and h, so they use a sample scheme to sparsify h so that it only has $\sqrt{n}$ nonzero entries. [55] uses sketch to reduce the size of projection matrix, more specifically, they maintain a matrix RP where $R \in \mathbb{R}^{\sqrt{n}} \times n$. [72] maintains $PR^\top$ and Rh and

they categorize a family of sketching matrices that can support their desired accuracy into the so-called *coordinate-wise embedding property*.

Definition 1.1.19 ((α, β, δ)-coordinate wise embedding)**.** We say a randomized matrix $R \in \mathbb{R}^{b \times n}$ satisfying (α, β, δ)-coordinate wise embedding if

$$
\begin{aligned}
&1.\ \mathbb{E}_{R\sim\Pi}[g^\top R^\top R h] = g^\top h, \\
&2.\ \mathbb{E}_{R\sim\Pi}[(g^\top R^\top R h)^2] \leq (g^\top h)^2 + \frac{\alpha}{b}\|g\|_2^2\|h\|_2^2, \\
&3.\ \Pr_{R\sim\Pi}\left[|g^\top R^\top R h - g^\top h| \geq \frac{\beta}{\sqrt{b}}\|g\|_2\|h\|_2\right] \leq \delta.
\end{aligned}
$$

Coordinate-wise embedding solves Task 1.1.9, it is more light-weighted, optimization friendly and compatible to many updates to the rows of matrix P, compare to that of AIPE. Both [55] and [72] can be directly captured by coordinate-wise embedding. Our result shows that the sampling technique in [27] is also a coordinate-wise embedding:

Theorem 1.1.20 (Informal version of Lemma 2.6.16)**.** *Given a vector $h \in \mathbb{R}^n$, let D be a diagonal sampling matrix defined as*

$$
D_{i,i} = \begin{cases} \frac{1}{p_i}, & \text{with probability } p_i := b \cdot \left(\frac{h_i^2}{\|h\|_2^2} + \frac{1}{n}\right); \\ 0, & \text{otherwise.} \end{cases}
$$

For any $g \in \mathbb{R}^n$, we have

- $\mathbb{E}_D[g^\top D h] = g^\top h$.
- $\mathbb{E}_D[(g^\top D h)^2] = (g^\top h)^2 + \frac{1}{b}\|g\|_2^2\|h\|_2^2$.
- $\Pr_D[|g^\top D h - g^\top h| \geq \frac{\log(1/\delta)}{\sqrt{b}}\|g\|_2\|h\|_2] \leq \delta$.

With this result, we achieve a final unification for [27, 55, 72], or more specifically, a class of algorithms for robust central path method with running time[2]

$$
O^*(n^\omega + n^{2+1/6})
$$

Both [27] and [72] can be viewed as sketching on the right, since one can decompose the sampling matrix in [27] into $R^\top R$ with $R \in \mathbb{R}^{\sqrt{n}\times n}$. The major difference is [27] adapts a data-dependent approach, since the sampling matrix is designed *after* observing the vector h. [72] is an oblivious approach, in the sense that the sketching matrices are chosen beforehand. Clearly, the former has some advantages, e.g., has the optimal dependence on β ($\log(1/\delta)$). But the parameters for oblivious sketching are also enough to achieve comparable results.

[55] is an instance of sketching on the left, i.e., maintains RP, and in the query time, first compute RPh then $R^\top RPh$. In contrast for the other two papers, it approximately maintains the product between I and Ph instead of P and h. This makes the central path infeasible, and one needs to restart the algorithm after a number of iterations. Their sketching technique is also oblivious.

[2]We use O^* to hide $\mathrm{poly}\log(n)$ and $n^{o(1)}$.

1.2 Faster Optimization Algorithms via Data Structures

In this section, we give an overview of our main results in speeding up optimization algorithms using the bag of data structures we developed.

Deterministic Linear-Sized Spectral Sparsifier. We show that the linear-sized spectral sparsifier problem solved by Batson, Spielman and Srivastava [12] is an iterative process in which each iteration, it solves Task 1.1.4. Hence, we use data structures we developed to speed up this process. Suppose the input is given as a list of vectors $X = \{x_1, \ldots, x_m\} \subset \mathbb{R}^d$.

References	Time for Sparse Instances	Time for Dense Instances
Batson, Spielman and Srivastava [12]	$\varepsilon^{-2} d(\sum_{i\in[m]} \mathrm{nnz}^2(x_i) + d^\omega)$	$\varepsilon^{-2} d(md^2 + d^\omega)$
Zouzias [90]	$\varepsilon^{-4} d^4 + \varepsilon^{-2} md^2$	$\varepsilon^{-4} d^4 + \varepsilon^{-2} md^2$
Theorem 1.2.1	$\sum_{i\in[m]} \mathrm{nnz}^2(x_i) + \varepsilon^{-2} d^{\omega+1}$	$md^{\omega-1} + \varepsilon^{-2} d^{\omega+1}$

Table 1.4: Main results for deterministic linear-sized spectral sparsifier.

Theorem 1.2.1 (Informal version of Theorem 3.1.8)**.** *Let $\{x_1, \ldots, x_m\} \subset \mathbb{R}^d$. Suppose we have $\sum_{i=1}^m x_i x_i^\top = I$, then there exists a deterministic algorithm that computes a set of non-negative weights $\{s_i\}_{i=1}^m$ with $|\{s_i : s_i \neq 0\}| = \Theta(\varepsilon^{-2} d)$ in time*

$$\min\{\sum_{i=1}^m \mathrm{nnz}^2(x_i), md^{\omega-1}\} + \varepsilon^{-2} d^{\omega+1}.$$

For the most standard setting that $m = d^2$, we improve from the $O(d^4)$ algorithm of [90] to $O(d^{\omega+1})$. Perhaps the most surprising aspect of our result is it models the algorithm of [12] as Task 1.1.4, which can also be adapted to improve the algorithm of [3], the sampling process of [53] when input is given as rank-1 matrices.

One-Sided Vector Packing and Experimental Design. The potential function studied in [12] has other applications, including restricted invertibility [81] and one-sided vector packing [84]. The iterative algorithm in [84] can be modeled as solving Task 1.1.6 per iteration. One interesting perspective of the one-sided vector packing task is the number of vectors one needs to pick can be small or large. We give different algorithms for these two cases. When the number of vectors is small, we use the AIPE data structure, and when the number is large, we use the AFN+TensorSparse data structure.

Theorem 1.2.2 (Informal of Theorem 3.2.10)**.** *Let $\tau, c \in (0,1)$ and $N \in \mathbb{N}_+$, if $X := \{x_1, \ldots, x_m\}$ is a finite sequence of vectors in $\mathbb{R}^d$ satisfying $\|x_i\|_2 = \frac{1}{\sqrt{N}}, \forall i \in [m]$ and $\sum_{i=1}^m x_i x_i^\top = I$. Then for any $n < m$, there exists a randomized algorithm (succeed with high probability) that takes time $\mathcal{T}$ to find a set S ($|S| = n$) such that*

$$\left\| \sum_{i\in S} x_i x_i^\top \right\| \leq \frac{1}{c} \cdot (\frac{n}{m} + O(\frac{1}{\sqrt{N}})).$$

Further, we have that,

- *if* $c \in (\tau, \frac{400\tau}{399+\tau})$, *then* $\mathcal{T} = \widetilde{O}((m^{1.01} + \mathrm{nnz}(X)) \cdot d^2 + n \cdot (m^{0.01} d^2 + d^\omega))$.
- *if* $c \in (\tau, \frac{1.01\tau}{0.01+\tau})$, *then* $\mathcal{T} = \widetilde{O}(md^2 + n \cdot (m + d^\omega))$.

References	Running Time
[84]	$n \cdot (md^2 + d^\omega)$
Theorem 1.2.2	$(m^{1.01} + \mathrm{nnz}(V)) \cdot d^2 + n \cdot (m^{0.01} d^2 + d^\omega)$
Theorem 1.2.2	$md^2 + n \cdot (m + d^\omega)$

Table 1.5: Main results for one-sided vector packing problem.

A key rounding algorithm for the experimental design problem [7] can also be viewed as solving Task 1.1.6, we include our result here.

Theorem 1.2.3 (Informal version of Theorem 3.3.16)**.** *Let* $\pi \in [0,1]^m$ *with* $\|\pi\|_1 \leq n$ *and* $\sum_{i=1}^m \pi_i x_i x_i^\top = I_d$*. Let* $\gamma \geq 3$ *and* $\varepsilon \in (0, \frac{1}{\gamma}]$*. Then, there exists a subset* $S \subset [m]$ *with* $|S| \leq n$ *such that*

$$\lambda_{\min}(\sum_{i \in S} x_i x_i^\top) \geq 1 - \gamma \cdot \varepsilon.$$

Let $\tau \in (0,1)$ *and* $c \in (\frac{1}{\gamma - 1}, 1)$*. If* $n \geq \frac{6d/\varepsilon^2}{\gamma - 1 - 1/c}$*, then there exists a randomized algorithm (success with high probability) that takes time* $\mathcal{T}$ *to find such* S*. Furthermore,*

- *If* $c \in (\tau, \frac{1.01\tau}{0.01+\tau})$, *then* $\mathcal{T} = \widetilde{O}(nd^2 + \varepsilon^{-1} n \cdot (n + d^\omega + (m-n)d^2))$;
- *If* $c \in (\tau, \frac{400\tau}{399+\tau})$, *then* $\mathcal{T} = \widetilde{O}((n^{1.01} + \mathrm{nnz}(X))d^2 + \varepsilon^{-1} n \cdot (n^{0.01} d^2 + d^\omega + (m-n)d^2))$.

References	Running Time
[7] vanilla	$\varepsilon^{-1} mnd^2$
[7] warm restart	mnd^2
Theorem 1.2.3	$nd^2 + \varepsilon^{-1} n(n + d^\omega + (m-n)d^2)$
Theorem 1.2.3	$(n^{1.01} + \mathrm{nnz}(X))d^2 + \varepsilon^{-1} n((n^{0.01} + z)d^2 + d^\omega + (m-n)d^2)$

Table 1.6: Main results for experimental design via regret minimization.

In the table, we use z to denote $\max_{i \in [m]} \mathrm{nnz}(x_i)$.

Training Deep, Over-parametrized Networks. Let $X = \{x_1, \ldots, x_n\} \subset \mathbb{R}^d$ be the dataset, the goal is to train a neural network with L layers and the network width $m = \mathrm{poly}(n, d, L)$. Note that the weight matrices for layer 2 to $L-1$ are of size $m \times m$. From an algorithmic perspective, the major challenge is to implement a training algorithm that has per iteration cost $o(m^2)$, since in this scenario, $m \geq n^4$, so reduce the dependence on m is crucial.

We obtain an algorithm that has training cost per iteration $\widetilde{O}(nm^{2-\alpha})$ where $\alpha \approx 0.31$. Under current best over-parametrization setting in which $m \geq n^4$, our algorithm has running time $\widetilde{O}(m^{1.94})$, which is the first *truly* subquadratic algorithm that realizes such training cost.

Theorem 1.2.4 (Runtime, informal version of Theorem 4.3.1)**.** *There exists a randomized algorithm that trains a multi-layer neural network of width m with the cost per training iteration being*

$$\widetilde{O}(nm^{2-\alpha}).$$

Moreover, under current over-parametrization setting in which $m \geq n^4$, the cost per training iteration being

$$\widetilde{O}(m^{1.94}).$$

Our algorithm also has a good convergence rate. Let λ_L be the smallest eigenvalue of the induced Neural Tangent Kernel of the deep, over-parametrized network and $y \in \mathbb{R}^n$ being the label. Moreover, let $f_t \in \mathbb{R}^n$ denote the prediction of the our trained neural network at time t, then

Theorem 1.2.5 (Convergence, informal version of Theorem 4.7.14)**.** *Suppose the width of the neural network satisfies $m \geq \mathrm{poly}(n, L, \lambda_L)$, then there exists an algorithm such that, over the randomness of initialization of the network and the algorithm, with probability at least $1 - e^{-\Omega(\log^2 n)}$, we have*

$$\|f_{t+1} - y\|_2 \leq \frac{1}{3}\|f_t - y\|_2.$$

Improving Initialization of [75]. One of the most important tasks for training over-parametrized networks is to quickly lookup the neurons that have been fired-up. This is Task 1.1.7, and we'll use the Correlation DTree and Correlation WTree to give two different runtime guarantees.

Consider the network architecture being a 2-layer ReLU network with m hidden units, the last layer is un-trained uniform Rademacher, and the training loss of the squared ℓ_2 loss. The ReLU we use is the shifted ReLU, with shifting parameter $b = \sqrt{0.4 \log n}$.

In [75], they obtain different running times for initializing weights or data points. Here, we show that by using DTree or WTree, the running time is essentially identical, asymptotically.

Theorem 1.2.6. *Given n data points in d-dimensional space, running gradient descent using Correlation DTree (Theorem 1.1.16) or Correlation WTree (Theorem 1.1.17) has initialization time $O(mnd)$. For each iteration, the expected training cost is $\widetilde{O}(m^{4/5}n^2d)$.*

We note that comparing to the per iteration cost of [75], which is $\widetilde{O}(m^{4/5}nd)$, we lose an extra n factor, but our initialization time is much more tractable in contrast to their exponential initialization time. We believe this result can find more applications for training deep networks, in which one observes an even larger discrepancy between m and n.

1.3 Discussion: Optimization-Friendly Data Structures

Unlike traditional design of data structures, the data structures we develop are specifically geared towards their deployment in an iterative process. We highlight some of the major differences:

- The balance of initialization, and per iteration query, update time. Traditional data structures typically have very fast query and update time, at the expense of worse initialization time, with the most prominent case being high dimensional geometric search data structures [1, 13]. If our goal is to use them in an iterative process, we have to balance the initialization and per iteration cost, so that the initialization is not too slow, and the query/update is not too fast so that the initialization time dominates. This makes certain data structures that are not perfect in both worlds very valuable, since they achieve a good balance due to the number of iterations of the optimization algorithm.
- Robust against adaptive queries. To improve the efficiency, many data structures maintain some internal randomness and provide Monte Carlo guarantees. However, most such success guarantees are against a sequence that is *oblivious* towards the internal randomness of the data structure. In an iterative process, this is usually *not* the case, since one might use the query result of the data structure from last iteration as the input to the data structure. For example, locality senstive hashing-based data structures [30, 40] are typically used in conjunction with Johnson-Lindenstrauss transforms [43], which assumes the query and update sequence is oblivious towards the JL and the LSH data structures, renders them useless in an iterative process, in their original form.

  To address this issue, many techniques have been developed, including:

  1. Using deterministic data structures [78] or data structures that generate fresh randomness for new queries which is perhaps the most prevalent adaptation of data structures in iterative process.
  2. Using an ε-net argument and union bound over all points on the net [69, 70]. Specifically, one can quantize the unit Euclidean ball into lattices spanned by so-called net points. One can show that $\exp(d)$ many net points suffice to cover the unit Euclidean ball in d-dimension. Then, one can union bound over all net points and the query can be performed on these points instead. This provides a "for all" guarantee for success probability, instead of for a fixed set.
  3. Effectively detecting the failure of the data structure via so-called flip number [34]. In many streaming problems, one typically maintain a linear sketch to approximate some statistics in small space, such linear sketch is randomized, and will leak the randomness of itself as the it receives more and more updates. However, it does not leak the randomness immediately after receiving one update, rather gradually performs worse and worse. Flip number captures the moment that the linear sketch does not function well. One can then use a fresh sketch to restart the computation.
  4. Injecting random noise into the algorithm to make it differentially private against the internal randomness of the data structure [79]. Differential privacy can be viewed as a framework to smooth the algorithm, in the sense that if only one row of the input database has changed, the function output should still be close. By adding carefully-crafted noises, one makes the algorithm "smooth" with respect to the randomness of the data structure. In this way, even though the query will leak the randomness, the adaptive adversary cannot differentiate between the internal randomness and the noise added.

5. For specific task such as finding the large/heavy coordinates of a vector, one can use sparse recovery technique to approximately find location of these coordinates first, then compute the values for those heavy coordinates exactly. In the standard sparse recovery problem [21, 35, 63], there are two steps: one is approximately finding the heavy indices ("locate frequency"). The other is approximately evaluating the frequencies ("set query"). In classical sparse recovery, we are not allowed to verify the heavy indices exactly and therefore have to run the set query step [62]. However, in optimization task, one can evaluate the heavy indices exactly and hence remove the effect of randomness. The reason is we can conditioned on the correctness of all locate frequency algorithm are correct at the beginning for the deterministic optimization algorithm.

1.4 Open Problems

In this thesis, we study data structures and optimizations, two central topics in computer science, to better support each other. We design data structures that are friendly to optimizations, and we formalize optimization problems in terms of data structure tasks.

Note that Task 1.1.9 is to estimate all inner products approximately, so it can serve as a more general case for Task 1.1.6. Naturally, all data structures that support Task 1.1.9 also can be used to solve Task 1.1.6. Hence, we have 3 data structures for this task: coordinate-wise embedding (CE), AIPE and AFN. CE is perhaps one of the most flexible and iteration-sensitive approach, when number of iterations is small ($\ll d$), it provides the best performance due to the number of sketches it uses depends on the number of iterations. When number of iterations is comparable to or slightly larger than the small dimension d ($\sim d$), AIPE gives the fastest algorithm, since it uses roughly d sketches. However, both of these approaches require linear scan over all m data points, so when number of iterations is very large, say comparable to the large dimension m ($\sim m$), AFN is more useful since it has the best query complexity. For many optimization tasks, number of iterations is rather small, therefore CE is more valuable for most applications. It is interesting to discover and study the natural optimization problems that have many iterations, so that AIPE and AFN can be utilized.

There are several open problems in this thesis. The first is to obtain a nearly linear time deterministic sparsification algorithm for graph and general matrix $V \in \mathbb{R}^{m \times d}$. Solving this problem deterministically has major implications in dynamic algorithms and computing minimum cut of a graph, and even might lend insights to the dynamic graph sparsification problem. We achieve the current best running time $O(d^{\omega+1})$, it will be an interesting to further improve this running time to $O(d^{\omega})$ or even $O(d^2)$, which is nearly linear for dense graph.

Also, we state Task 1.1.8 but do not provide a data structure and its corresponding optimization problem to it. This task is actually related to the symmetric Determinantal Point Process (DPP) and Non-symmetric Determinantal Point Process (NDPP). We leave exploring data structures to speed up this task as an open problem.

Improving the running time of linear programming to $O^*(n^2)$ is always an important open problem in computer science. Currently, even though we have $\omega = 2$, the best running time is $O^*(n^{2+1/18})$. We believe to obtain a breakthrough result, it is essential to simplify the current best

algorithms for LP. We show that a coordinate-wise embedding-based dimensionality reduction tool is key to this speedup, and it will be interesting to see how this property manifests in state-of-the-art [42] and potentially lead to other future improvements.

Finally, note that we solve a one-sided Kadison-Singer problem posed by Weaver [84] in this thesis. The ultimate goal is to solve the two-sided Kadison-Singer problem in quasi-polynomial time. Marcus, Spielman and Srivastava [58] shows that given a set of vectors $\{v_1, \ldots, v_m\} \subset \mathbb{R}^d$ with $\sum_{i=1}^m v_i v_i^\top = I$ and $\|v_i\|_2^2 \leq \alpha$, it is possible to find a subset S with $\frac{1}{2} - c \cdot \sqrt{\alpha} \leq \sum_{i \in S} \langle v_i, u \rangle^2 \leq \frac{1}{2} + c \cdot \sqrt{\alpha}$, which resolves the Kadison-Singer conjecture [44]. However, their algorithm requires exhaustive search over all possible subsets, which takes $O(2^m)$ time. Instead, we ask the following data structure problem: given m vectors in dimension d, does there exist a data structure that can preprocess these m vectors and answer the query on the set of vectors $\sum_{i=1}^m c_i v_i$ where $c_i \in \{\pm 1\}$? Intuitively, this means that we are given m points, and we wish to design data structure that answers query in a restricted span of these m points and can be dynamically updated, in quasi-polynomial time? We believe such data structure is key to solve Kadison-Singer problem in quasi-polynomial time, improving from the state-of-the-art which approximates the largest root of polynomial in $2^{\widetilde{O}(m^{1/3})}$ time [9]. Note that when $d = \mathrm{poly}(\log m)$, we can aim at a preprocessing time of $m^{\mathrm{poly}(d)} = 2^{\mathrm{poly}(\log m)}$ which could already be significant progress on algorithmic Kadison-Singer problem. This is an open problem posed in a 2021 April manuscript by Song, Xu and Zhang [74][3].

1.5 Preliminaries and Thesis Structure

We give a preliminary overview of the notations that will be used across this thesis. We also discuss the structure of this thesis and provide a general roadmap.

1.5.1 Notations

General Notations. For a positive integer n, we use $[n]$ to denote the set $\{1, 2, \cdots, n\}$. We use $\mathbb{E}[\cdot]$ for expectation and $\Pr[\cdot]$ for probability. We use $\mathcal{N}(\mu, \sigma^2)$ for the Gaussian distribution with mean μ and variance σ^2. We use $\widetilde{O}(f(n))$ for $O(f(n) \cdot \mathrm{poly}\log(f(n)))$. For a matrix A or a vector x, we use $\mathrm{nnz}(A)$ and $\mathrm{nnz}(x)$ to denote the number of nonzero entries of A and x respectively.

Vectors. For any vector x, we use $\|x\|_2$ to denote its ℓ_2 norm, $\|x\|_0$ denote the number of nonzero entries x. Note that $\|\cdot\|_0$ is a semi-norm since it satisfies triangle inequality. Let $\langle \cdot, \cdot \rangle$ be the inner product between two vectors defined as $\langle x, y \rangle = x^\top y$. We use $x \otimes y = \mathrm{vec}(xy^\top)$ for the tensor product between two conforming vectors x and y. For a vector $x \in \mathbb{R}^n$, we use $\mathrm{diag}(x) \in \mathbb{R}^{n \times n}$ to denote a diagonal matrix with diagonal being x.

Matrices. For any matrix $A \in \mathbb{R}^{m \times m}$, We use $\|A\|$ for the spectral norm of a real symmetric matrix A, i.e., $\|A\| = \max_{i \in [m]} \{|\lambda_i(A)|\}$ where $\lambda_i(A)$ is the i-th eigenvalue. We use $\|A\|_F$ for

[3] [74] is an early version of [78], most of the results in [74] have been improved in [78]. We state the open problem only in [74] 2021 manuscript version.

the Frobenius norm of A. We use $\|A\|_0$ to denote the number of nonzero entries of A.

For any $A \in \mathbb{R}^{n \times m}$, we define $A^\top \in \mathbb{R}^{m \times n}$ to be the transpose of A. We use I_m for the identity matrix of size $m \times m$. For a square and full rank matrix A, we use A^{-1} to denote its inverse.

Given a real square matrix A, we use $\lambda_{\max}(A)$ and $\lambda_{\min}(A)$ for its largest and smallest eigenvalues respectively. Given a real matrix A, we use $\sigma_{\max}(A)$ and $\sigma_{\min}(A)$ for its largest and smallest singular values respectively. We use $\mathrm{tr}[A]$ to denote the trace of matrix A.

We use $\langle \cdot, \cdot \rangle$ to denote the inner product between two conforming matrices A and B, defined as $\langle A, B \rangle = \mathrm{tr}[A^\top B]$.

We say a symmetric matrix $A \in \mathbb{R}^{n \times n}$ is positive semidefinite (PSD, denoted as $A \succeq 0$) if for any vector $x \in \mathbb{R}^n$, $x^\top A x \geq 0$. We say a symmetric matrix $A \in \mathbb{R}^{n \times n}$ is positive definite (PD, denoted as $A \succ 0$) if for any vector $x \in \mathbb{R}^n$, $x^\top A x > 0$.

For two symmetric matrices A and B with conforming sizes, we say $A \preceq B$ if and only if $A - B \preceq 0$.

For a real positive semidefinite matrix A, we define its square root $A^{1/2}$ to be the unique positive semidefinite matrix such that $(A^{1/2})^\top A^{1/2} = A$.

We define $\mathcal{T}_{\mathrm{mat}}(a, b, c)$ to be the time of multiplying an $a \times b$ matrix with another $b \times c$ matrix. Note that $\mathcal{T}_{\mathrm{mat}}(a, b, c) = O(\mathcal{T}_{\mathrm{mat}}(a, c, b)) = O(\mathcal{T}_{\mathrm{mat}}(b, a, c))$.

For real symmetric matrices A and B of the same size, we use $A \approx_\varepsilon B$ if $(1 - \varepsilon)B \preceq A \preceq (1 + \varepsilon)B$.

For two matrices A and B, we use $A \odot B$ to denote the Hadamard product between A and B and we use $A \times B$ to denote the tensor product between A and B.

1.5.2 Thesis Structure

In Chapter 2, we design data structures for tasks in Section 1.1. In Chapter 3, we use data structures specifically geared towards inner product type tasks (see Task 1.1.1 and 1.1.2) to improve various sparsification algorithms. In Chapter 4, we give the first subquadratic algorithm for training deep, over-parametrized neural networks. In Appendix A, we include an implementation of the approximate furthest neighbor search data structure and its proof. In Appendix B, we include an implementation of the inverse maintenance data structure when input is a projection matrix, coupled with a vector. In Appendix C, we introduce a new variant of fast sketching matrix for tensors, using circulant matrix.

Chapter 2

Data Structures

We design data structures to solve the tasks posed in Section 1.1. For different tasks, we develop different data structures using techniques from a wide range of literature. Many of our data structures will be deterministic — this is essential when used in conjunction with an iterative process. For randomized data structures, we make sure that either each update/query uses fresh randomness, or we take extra care to make sure the data structure is robust against adaptive update/query sequence.

The data structures in Section 2.2.1, 2.2.2, 2.3 and 2.4 are based on the arXiv document https://arxiv.org/pdf/2204.03209.pdf coauthored by the thesis author. Section 2.5 is based on the arXiv document https://arxiv.org/pdf/2112.07628.pdf coauthored by the thesis author.

2.1 Tools

In this section, we recall some important tools and facts.

In this section, we present some probability tools.

We start with the standard 2-stable Gaussian distribution. We refer the readers to [30] for more details.

Fact 2.1.1 (Standard Gaussian is 2-stable)**.** *Let $Z, X_1, X_2, \ldots, X_k \sim \mathcal{N}(0,1)$ and $v \in \mathbb{R}^k$, then $\sum_{i=1}^k v_i X_i$ and $\|v\|_2 \cdot Z$ have the same distribution.*

Next, we present a concentration and anti-concentration bound for Gaussian distribution.

Fact 2.1.2 (Gaussian concentration bound)**.** *Let $X \sim \mathcal{N}(0,1)$ and $t > 0$, then we have*

- **Part 1 Concentration.** $\Pr[|X| \geq t] \leq 2\exp(-t^2/2)/t$.
- **Part 2 Anti-Concentration.** *There exists a constant $B > 0$ such that*

$$\Pr[|X| \geq t] \geq 2B \cdot \exp(-t^2/2)/\max\{1, t\}.$$

Definition 2.1.3. Let X be a random variable, we use $\|X\|_{L_q}$ to denote $(\mathbb{E}[|X|^q])^{1/q}$. By Minkowski's inequality, $\|\cdot\|_{L_q}$ is a norm when $q \geq 1$.

Lemma 2.1.4 (Hanson-Wright inequality [36])**.** *For σ_1, σ_n independent Rademachers and $A \in \mathbb{R}^{n \times n}$, for all $q \geq 1$,*

$$\|\sigma^\top A \sigma - \mathbb{E}[\sigma^\top A \sigma]\|_{L_q} \leq O(1) \cdot (\sqrt{q} \cdot \|A\|_F + q \cdot \|A\|).$$

Lemma 2.1.5. *For Y distributed as* $\mathsf{Binomial}(N, \alpha)$ *for integer $N \geq 1$ and $\alpha \in (0, 1)$, let $1 \leq p \leq N$ and define $B := p/(\alpha N)$. Then*

$$\|Y\|_{L_p} \leq \begin{cases} \frac{p}{\log B}, & \text{if } B \geq e \\ \frac{p}{B}, & \text{if } B < e. \end{cases}$$

2.2 Trees

Trees are perhaps the most widely-used data structure primitives in computer science. In this thesis, we design trees that can support fast search and sample based on the inner product of inputs. Unlike the usual self-balancing tree or link-cut tree data structures, for inner product-typed query, there's often a naturally induced ordering. This means that we can preprocess data using this natural order, and search along a tree path only takes $O(\log m)$ levels. Unlike typical search trees in which compute each node takes $O(1)$ time, we'll have to compute the inner product in $O(d)$ time.

2.2.1 Matrix Search Tree: Input Sparsity Time Initialization and Fast Query

Given a list of matrices $\{M_1, \ldots, M_m\} \subset \mathbb{R}^{d \times d}$, we design a data structure to solve Task 1.1.4 and 1.1.5. The data structure proprocesses the list of matrices in input sparsity time, i.e., $\sum_{i \in [m]} \mathrm{nnz}(M_i)$. When query, it takes inner product between a query matrix A and a partial sum matrix stored at a tree node in $O(d^2)$ time and only traverses one path from root to a leaf. Note that when we are dealing with vector inputs, we need to spend $O(\sum_{i \in [m]} \mathrm{nnz}(v_i)^2)$ time forming the outer products $v_i v_i^\top$. Our data structure also supports sample based on the distribution defined by the inner product.

Algorithms.

Algorithm 1 Matrix Positive Search

data structure MATRIXPS ▷ Theorem 2.2.1
members
  $M_1, M_2, \cdots, M_m \subset \mathbb{R}^{d \times d}$ (matrix of each index)
  $S_0, S_1, S_2, \cdots, S_n \subset \mathbb{R}^{d \times d}$ (partial sum of each node)
  Binary tree T (each node is a tuple (i_1, i_2, S) where $i_1 < i_2$ are indices and $S = \sum_{i=i_1}^{i_2} S_i$)
end members

procedure INIT($M_1, M_2, \cdots, M_m \subset \mathbb{R}^d$)
  $S_0 = 0$
  for $i = 1$ to m **do**
    $M_i \leftarrow M_i$
    $s_i \leftarrow S_{i-1} + M_i$
  end for
  Insert $(1, m, S_m)$ as root of T
  while exists a leaf $l = (i_1, i_2, S)$ of T such that $i_2 - i_1 \geq 1$ **do**
    $k = \lfloor (i_1 + i_2)/2 \rfloor$
    Insert $(i_1, k, S_k - S_{i_1 - 1})$ as left child of l
    Insert $(k + 1, i_2, S_{i_2} - S_k)$ as right child of l
  end while
end procedure

procedure QUERYPOSITIVESEARCH($A \in \mathbb{R}^{d \times d}$) ▷ Lemma 2.2.2
  $r \leftarrow$ root of T
  while r is not a leaf of T **do**
    $r_1 \leftarrow$ left child of r, $r_2 \leftarrow$ right child of r
    $M_1 \leftarrow$ matrix of r_1, $M_2 \leftarrow$ matrix of r_2
    $p_1 \leftarrow \langle A, M_1 \rangle$, $p_2 \leftarrow \langle A, M_2 \rangle$
    if $p_1 > 0$ **then**
      $r \leftarrow r_1$
    else ▷ $p_2 > 0$
      $r \leftarrow r_2$
    end if
  end while
  return index of r
end procedure

Algorithm 2 Matrix Sample

data structure MATRIXPS ▷ Theorem 2.2.1
procedure QUERYSAMPLE($A \in \mathbb{R}^{d \times d}$) ▷ Lemma 2.2.3
  $r \leftarrow$ root of T
  while r is not a leaf of T **do**
    $r_1 \leftarrow$ left child of r, $r_2 \leftarrow$ right child of r
    $M_1 \leftarrow$ matrix of r_1, $M_2 \leftarrow$ matrix of r_2
    $p_1 \leftarrow \langle A, M_1 \rangle$, $p_2 \leftarrow \langle A, M_2 \rangle$
    Generate random number $b \in (0, 1)$
    if $b < \frac{p_1}{p_1 + p_2}$ **then**
      $r \leftarrow r_1$
    else
      $r \leftarrow r_2$
    end if
  end while
  return index of r
end procedure
end data structure

Running time and correctness

We summarize the correctness and running time of Algorithm 1 as follows:

Theorem 2.2.1 (Formal version of Theorem 1.1.12)**.** *There exists a data structure with the following procedures:*

- INIT($\{M_1, M_2, \cdots, M_m\} \subseteq \mathbb{R}^{d \times d}$). *It takes a sequence of matrices $M_1, M_2, \cdots, M_m$ as input, and preprocesses in time $O(\sum_{i=1}^m \mathrm{nnz}(M_i))$.*
- QUERYPOSITIVESEARCH($A \in \mathbb{R}^{d \times d}$). *Given a matrix A with the promise that $\sum_{i=1}^m \langle M_i, A \rangle > 0$, it returns an index i such that $\langle M_i, A \rangle > 0$ in time $O(d^2 \log m)$.*
- QUERYSAMPLE($A \in \mathbb{R}^{d \times d}$). *Given a positive semidefinite matrix $A \in \mathbb{R}^{d \times d}$, it samples an index i with probability $\frac{\langle M_i, A \rangle}{\sum_{i=1}^m \langle M_i, A \rangle}$ in time $O(d^2 \log m)$.*

Proof. We prove the data structure (see Algorithm 1) satisfies the requirements. In INIT, every node (i_1, i_2, M) stores the partial sum of matrices $\sum_{j=i_1}^{i_2} M_j$, the number of nodes is $O(m)$, then the preprocess time is $O(\sum_{i=1}^m \mathrm{nnz}(M_i))$ accounts for the sparsity of the input.

For QUERYPOSITIVESEARCH, see Lemma 2.2.2 and for QUERYSAMPLE, see Lemma 2.2.3. □

Lemma 2.2.2 (Positive Search)**.** *Given a matrix A with the promise that $\sum_{i=1}^m \langle M_i, A \rangle > 0$,* QUERYSEARCH *returns an index i such that $\langle M_i, A \rangle > 0$ in time $O(d^2 \log m)$.*

Proof. For QUERYSEARCH, note that the correctness holds obviously: given a node and its two children, suppose we know the inner product at the node is greater than 0, then it must be the case that at least one of its two children has value greater than 0. For the running time, each inner product takes $O(d^2)$ time, and we traverse a path on the tree of depth $O(\log m)$, so it takes $O(d^2 \log m)$ time in total. □

Lemma 2.2.3 (Sample)**.** *Given a positive semi-definite matrix $A \in \mathbb{R}^{d\times d}$,* QUERYSAMPLE *samples an index i with probability $\frac{\langle M_i, A\rangle}{\sum_{i=1}^m \langle M_i, A\rangle}$ in time $O(d^2 \log m)$.*

Proof. In QUERYSAMPLE, we sequentially sample its child node from the root to a leaf and return the index of the leaf. To prove its correctness, for an index interval $\{i_1, i_1+1, \cdots, i_2\}$, define $w_{[i_1,i_2]} = \sum_{i=i_1}^{i_2} \langle A, M_i\rangle$, and for a node $m = (i_1, i_2, M) \in T$, define $w_n = w_{[i_1,i_2]}$. Also for a parent $p \in T$ and a child c of p, define $\Pr[c|p] = \Pr[c \text{ is sampled} \mid p \text{ is sampled}]$, then $\Pr[c|p] = \frac{w_c}{w_p}$. Now for each leaf $l = (i, i, M_i) \in T$, suppose the path from root r to l is $r = r_0 \to r_1 \to \cdots \to r_k = l$, then

$$\begin{aligned}
\Pr[\text{QUERY outputs } l] &= \Pr[r_1|r]\Pr[r_2|r_1]\cdots\Pr[l|r_{k-1}] \\
&= \frac{w_{r_1}}{w_r}\frac{w_{r_2}}{w_{r_1}}\cdots\frac{w_l}{w_{r_{k-1}}} \\
&= \frac{w_l}{w_r} \\
&= \frac{w_l}{\sum_{i=1}^m w_i} \\
&= \frac{\langle A, M_l\rangle}{\langle A, \sum_{i=1}^m M_i\rangle}.
\end{aligned}$$

And note that the depth of T is $O(\log m)$, the running time of QUERYSAMPLE is $O(d^2 \log m)$ since computing each inner product takes $O(d^2)$ time. □

2.2.2 Faster Initialization via Fast Matrix Multiplication and Batching

We note that the MATRIXPS data structure is more general than some of the tasks, in which the input is given as a list of vectors $\{v_1, \ldots, v_m\} \subset \mathbb{R}^d$, we can speed up the initialization via fast matrix multiplication, in the expense of worse query time. In certain tasks we can balance the initialization time and query time to achieve a better overall performance.

The idea is to maintain a tree with only m/d nodes, with each of the leaf is a sum of d outer products $\sum_{i\in S} v_i v_i^\top$ for $S \subset [m]$ and $|S| = d$. During initialization, we can form each leaf in d^ω time, and since there are m/d leaves in total, it only takes $O(md^{\omega-1})$ time to initialize. We store the $d \times d$ matrix V where each column is v_i. During query, when we reach the leaf node, we can perform the matrix multiplication $V^\top AV$ and extract the diagonal entries in time $O(d^\omega)$.

Algorithm

Algorithm 3 Vector Search & Sample

data structure VECTORPS ▷ Theorem 2.2.4
members
  $v_1, v_2, \cdots, v_m \subset \mathbb{R}^d$ (vector of each index)
  $\{S_{i,j}\}_{i \in \{0,\ldots,\log(m/d)\}, j \in [2^{-i}m/d]} \in \mathbb{R}^{d \times d}$
end members

procedure INIT($v_1, v_2, \cdots, v_m \subset \mathbb{R}^d$)
  for $i = 1 \to m/d$ **do**
    $V_i \leftarrow \begin{bmatrix} | & | & \cdots & | \\ v_{(i-1)d+1} & v_{(i-1)d+2} & \cdots & v_{id} \\ | & | & \cdots & | \end{bmatrix}$
    $S_{0,i} \leftarrow V_i V_i^\top$
  end for
  for $i = 1 \to \log(m/d)$ **do**
    for $j = 1 \to 2^{-i}m/d$ **do**
      $S_{i,j} \leftarrow S_{i-1,2j-1} + S_{i-1,2j}$
    end for
  end for
end procedure

procedure QUERYPOSITIVESEARCH($A \in \mathbb{R}^{d \times d}$) ▷ Lemma 2.2.6
  $j \leftarrow 1$
  for $i = \log(m/d) \to 0$ **do**
    $L \leftarrow S_{i-1,2j-1}$, $R \leftarrow S_{i-1,2j}$
    $p_1 \leftarrow \langle A, L \rangle$, $p_2 \leftarrow \langle A, R \rangle$
    $j \leftarrow \begin{cases} 2j-1, & \text{with probability } p_1 > 0 \\ 2j, & \text{with probability } p_2 > 0. \end{cases}$
  end for
  $V \leftarrow V_j$
  $B \leftarrow V^\top A V$
  for $i = 1 \to d$ **do**
    if $B_{i,i} > 0$ **then**
      $i^* \leftarrow i$
      break
    end if
  end for
  return i^*
end procedure

Algorithm 4 Vector Sample

data structure VECTORPS ▷ Theorem 2.2.4
procedure QUERYSAMPLE($A \in \mathbb{R}^{d \times d}$) ▷ Lemma 2.2.7
  $j \leftarrow 1$
  for $i = \log(m/d) \to 1$ **do**
    $M \leftarrow S_{i,j}$
    $L \leftarrow S_{i-1,2j-1}$, $R \leftarrow S_{i-1,2j}$
    $p_1 \leftarrow \langle A, L \rangle$, $p_2 \leftarrow \langle A, R \rangle$
    $p \leftarrow \langle A, M \rangle$
    $j \leftarrow \begin{cases} 2j-1, & \text{with probability } p_1/p \\ 2j, & \text{with probability } p_2/p. \end{cases}$
  end for
  $V \leftarrow V_j$
  $B \leftarrow V^\top AV$
  return i with probability $B_{i,i}/\mathrm{tr}[B]$
end procedure
end data structure

Running time and correctness

Theorem 2.2.4 (Formal version of Theorem 1.1.13)**.** *There exists a data structure with the following procedures:*

- INIT($\{v_1, v_2, \cdots, v_m\} \subseteq \mathbb{R}^d$). *It takes a sequence of vectors $v_1, v_2, \cdots, v_m$ as input, and preprocesses in time $O(md^{\omega-1})$ and in space $O(md)$.*
- QUERYSAMPLE($A \in \mathbb{R}^{d\times d}$). *Given a positive semidefinite matrix $A \in \mathbb{R}^{d\times d}$, it samples an index i with probability $\frac{\langle M_i, A\rangle}{\sum_{i=1}^m \langle M_i, A\rangle}$ in time $O(d^2 \log m + d^\omega)$.*
- QUERYPOSITIVESEARCH($A \in \mathbb{R}^{d\times d}$). *Given a matrix A with the promise that $\sum_{i=1}^m \langle M_i, A\rangle > 0$, it returns an index i such that $\langle M_i, A\rangle > 0$ in time $O(d^2 \log m + d^\omega)$.*

Proof. We prove the data structure (see Algorithm 3) satisfies the requirements. In INIT, we will perform m/d matrix multiplications of $d \times d$ matrix, yields a time of $O(md^{\omega-1})$. We then compute m/d sums of $d \times d$ matrices, which takes $O(md)$ time, or $\mathrm{nnz}(V^2)$ time. Note that the space is only $O(md)$, since we have constructed a tree of $O(m/d)$ nodes, with each node stores a $d \times d$ matrix. We note an invariant by our construction: for matrix $S_{i,j}$, it represents the sum of outer products $\sum_{k=i_1}^{i_2} v_k v_k^\top$, where $i_1 = 2^i(j-1)d + 1$ and $i_2 = 2^i jd$, hence $S_{\log(n/d),1} = \sum_{i=1}^m v_i v_i^\top$.

For QUERYSAMPLE, we prove in Lemma 2.2.7. For QUERYSEARCH, we prove in Lemma 2.2.6. □

We will show that each matrix $S_{i,j}$ stores the proper sum of $v_i v_i^\top$ over a desired range.

Lemma 2.2.5. *Let $i \in \{0, 1, \ldots, \log(m/d)\}$ and $j \in [2^{-i}m/d]$, then we have*

$$S_{i,j} = \sum_{k=1}^{2^i d} v_{2^i j - 2^i + k} v_{2^i j - 2^i + k}^\top.$$

Proof. We start with the bottom level where $i = 0$. Note that $S_{0,j} = V_j V_j^\top$, where $V_j = \begin{bmatrix} | & | & \cdots & | \\ v_{(j-1)d+1} & v_{(j-1)d+2} & \cdots & v_{jd} \\ | & | & \cdots & | \end{bmatrix}$. Use the outer product formulation of matrix multiplication, we have that

$$\begin{aligned} S_{0,j} &= V_j V_j^\top \\ &= \sum_{k=1}^d (V_j)_{*,k} (V_j)_{*,k}^\top \\ &= \sum_{k=1}^d v_{(j-1)d+k} v_{(j-1)d+k}^\top \\ &= v_{(j-1)d+1} v_{(j-1)d+1}^\top + v_{(j-1)d+2} v_{(j-1)d+2}^\top + \ldots + v_{jd} v_{jd}^\top. \end{aligned}$$

For internal levels, we can show by induction. For $i = 1$, note that $S_{1,j} = S_{0,2j-1} + S_{0,2j}$, we know that $S_{0,2j-1} = \sum_{k=1}^{d} v_{(2j-2)d+k} v_{(2j-2)d+k}^\top$ and $S_{0,2j} = \sum_{k=1}^{d} v_{(2j-1)d+k} v_{(2j-1)d+k}^\top$, hence

$$S_{1,j} = \sum_{k=1}^{2d} v_{(2j-2)d+k} v_{(2j-2)d+k}^\top.$$

Assume this holds up until some level l, i.e., $S_{l,j} = \sum_{k=1}^{2^l d} v_{2^l j - 2^l + k} v_{2^l j - 2^l + k}^\top$, then

$$\begin{aligned} S_{l+1,j} &= S_{l,2j-1} + S_{l,2j} \\ &= (\sum_{k=1}^{2^l d} v_{2^l(2j-1)-2^l+k} v_{2^l(2j-1)-2^l+k}^\top) + (\sum_{k=1}^{2^l d} v_{2^l(2j)-2^l+k} v_{2^l(2j)-2^l+k}^\top) \\ &= \sum_{k=1}^{2^{l+1} d} v_{2^{l+1} j - 2^{l+1} + k} v_{2^{l+1} j - 2^{l+1} + k}^\top. \end{aligned}$$

Hence, we complete the proof. Note that when $i = \log(m/d)$, $j = 1$ and

$$\begin{aligned} S_{m/d,1} &= \sum_{k=1}^{m} v_{(m/d)-(m/d)+k} \\ &= \sum_{k=1}^{m} v_k. \end{aligned}$$

□

Lemma 2.2.6 (Positive Search)**.** *Given a matrix A with the promise that $\sum_{i=1}^{m} \langle v_i v_i^\top, A \rangle > 0$,* QUERYPOSITIVESEARCH *returns an index i such that $\langle v_i v_i^\top, A \rangle > 0$ in time $O(d^2 \log(m/d) + d^\omega)$.*

Proof. To see the correctness, we note a simple if and only if statement: given numbers $a_1, \ldots, a_m$ such that $\sum_{i=1}^{m} a_i > 0$, then there must exist an i such that $a_i > 0$, otherwise the sum must be negative. For our search procedure, we can prove the correctness inductively: at root, since we know that $\sum_{i=1}^{m} v_i^\top A v_i > 0$, then it must be the case that either $\sum_{i=1}^{m/2} v_i^\top A v_i > 0$ or $\sum_{i=m/2+1}^{n} v_i^\top A v_i > 0$, otherwise the root sum must be negative. Suppose this holds to level k, and we are deciding where to go for level $k+1$, note by induction hypothesis, for level k, the inner product must be positive, then it must be the case that one of its children has a positive inner product, otherwise the sum of inner product will be negative. Also, each node stores the correct partial sum, as shown in Lemma 2.2.5.

At the bottom level for leaf node j, we compute $B = V_j^\top A V_j$, the claim is the diagonal entry $B_{i,i} = v_{(j-1)d+i}^\top A v_{(j-1)d+i}$, to see this, note that

$$(V_j^\top A V_j)_{i,i} = (V_j^\top \begin{bmatrix} | & | & \cdots & | \\ A v_{(j-1)d+1} & A v_{(j-1)d+2} & \cdots & A v_{jd} \\ | & | & \cdots & | \end{bmatrix})_{i,i}$$

$$= v_{(j-1)d+i}^\top A v_{(j-1)d+i}.$$

This completes the correctness proof.

For the running time, each inner product takes $O(d^2)$ time, and we traverse a path on the tree of depth $O(\log(m/d))$, for the leaf, it takes $O(d^\omega)$ time. This concludes our proof. □

Lemma 2.2.7 (Sample)**.** *Given a positive semidefinite matrix $A \in \mathbb{R}^{d \times d}$,* QUERYSAMPLE *samples an index i with probability $\frac{\langle v_i v_i^\top, A \rangle}{\sum_{i=1}^m \langle v_i v_i^\top, A \rangle}$ in time $O(d^2 \log(m/d) + d^\omega)$.*

Proof. To prove we sample index i with probability $\frac{\langle v_i v_i^\top, A \rangle}{\sum_{i=1}^m \langle v_i v_i^\top, A \rangle}$, suppose i comes from the matrix $S_{0,j}$, and we use $\Pr[S_{0,j}]$ to denote the probability of sampling the matrix $S_{0,j}$.

$$\begin{aligned}
& \Pr[\text{QUERYSAMPLE outputs } i] \\
= & \Pr[i \mid S_{0,j}] \Pr[S_{0,j}] \\
= & \Pr[i \mid S_{0,j}] \Pr[S_{0,j} \mid S_{1,\lfloor j/2 \rfloor}] \cdot \ldots \cdot \Pr[S_{\log(m/d)-1,\lfloor 2^{1-\log(m/d)} j \rfloor} \mid S_{\log(m/d),1}] \\
= & \frac{(V_j^\top A V_j)_{i,i}}{\mathrm{tr}[V_j^\top A V_j]} \frac{\langle A, S_{0,j} \rangle}{\langle A, S_{1,\lfloor j/2 \rfloor}} \cdot \ldots \frac{\langle A, S_{\log(m/d)-1,\lfloor 2^{1-\log(m/d)} j \rfloor} \rangle}{\langle A, S_{\log(m/d),1} \rangle} \\
= & \frac{v_i^\top A v_i}{\langle A, S_{\log(m/d),1} \rangle} \\
= & \frac{\langle v_i v_i^\top, A \rangle}{\langle \sum_{i=1}^m v_i v_i^\top, A \rangle},
\end{aligned}$$

the last line is by $S_{\log(m/d),1} = \sum_{i=1}^m v_i v_i^\top$.

For the running time, note that from the root to the level above leaves, it takes $O(d^2)$ per level, and there are $\log(m/d)$ levels in total. For the leaf, it takes $O(d^\omega)$ time. Hence, it takes $O(d^2 \log(m/d) + d^\omega)$ time in total.

□

Remark 2.2.8. *The* VECTORPS *data structure can be viewed as using a crude estimation for all levels above the bottom level, and for the bottom level, we use a more refined computation to exactly estimate $v_i^\top A v_i$. This means we have to spend more time at the bottom level, but this is fine since we also gain speedup from the initialization. In the setting of a dense graph or a matrix with $m \geq d^2$ rows, we achieve a initialization time of $md^{\omega-1}$ and overall iteration cost $\varepsilon^{-2} d^{\omega+1}$, these two terms balance out. In contrast, with the* MATRIXPS *data structure, it might incur $md^2 \approx d^4$ time for initialization. Such a high-level idea of the tradeoff between crude and refined computation has also been utilized in balancing sample complexity in completely different field (see sparse Fourier transform in the continuous setting [63]). Our case is a different scenario, since we care about the running time perspective of this tradeoff.*

2.2.3 Correlation Trees

In this section, we showcase two data structures for Task 1.1.7. The motivation for designing these data structures comes from training over-parametrized neural networks with ReLU or

shifted ReLU activations. Let $X \in \mathbb{R}^{n \times d}$ consists of n data points of dimension d, the hidden layer is a matrix $W \in \mathbb{R}^{m \times d}$ with $m \gg n$. For each data point x, the forward pass can be presented as $\sigma_b(Wx)$, where σ_b is the (shifted) ReLU activation defined as $\sigma_b(x) = \max\{x - b, 0\}$. In order to use first order method such as gradient descent to optimize the network, it is important to find the set $\{w_i : \langle w_i, x \rangle \geq b, i \in [m]\}$ efficiently. Naively one needs to pay $O(md)$ time for each data point, which will take $\Omega(nmd)$ time per training iteration.

To speed up this process, [75] uses some deterministic high-dimensional data structure to find and update this set fast. Unfortunately, their preprocessing time is exponential in terms of dimension d.

We alleviate this issue via trees based on inner product, but used in a different fashion.

Correlation DTree data structure

We first present a data structure that preprocesses all weights $w_1, \ldots, w_m$ for each data point x_i. During query, we simply look at one tree.

Theorem 2.2.9 (Correlation DTree data structure, formal version of Theorem 1.1.16)**.** *There exists a data structure with the following procedures:*

- INIT($\{w_1, w_2, \cdots, w_m\} \subset \mathbb{R}^d, \{x_1, x_2, \cdots, x_n\} \subset \mathbb{R}^d, n \in \mathbb{N}, m \in \mathbb{N}, d \in \mathbb{N}$). *Given a series of weights $w_1, w_2, \cdots, w_m$ and datas $x_1, x_2, \cdots, x_n$ in d-dimensional space, it preprocesses in time $O(nmd)$*
- UPDATE($z \in \mathbb{R}^d, r \in [m]$). *Given a weight z and index r, it updates weight w_r with z in time $O(n \cdot (d + \log m))$*
- QUERY($i \in [n], \tau \in \mathbb{R}$). *Given an index i indicating data point x_i and a threshold τ, it finds all index $r \in [m]$ such that $\langle w_r, x_i \rangle > \tau$ in time $O(|\widetilde{S}(\tau)| \cdot \log m)$, where $\widetilde{S}(\tau) := \{r : \langle w_r, x_i \rangle > \tau\}$*

Algorithm

Algorithm 5 Correlation DTree data structure

data structure CORRELATIONDTREE ▷ Theorem 2.2.9
members
  $W \in \mathbb{R}^{m \times d}$ (m weight vectors)
  $X \in \mathbb{R}^{n \times d}$ (n data points)
  Binary tree $T_1, T_2, \cdots, T_n$ ▷ n binary search trees
end members

public:
procedure INIT($w_1, w_2, \cdots, w_m \in \mathbb{R}^d, m, x_1, x_2, \cdots, x_n \in \mathbb{R}^d$, n, m, d) ▷ Lemma 2.2.10
  for $i = 1 \to n$ **do**
    $x_i \leftarrow x_i$
  end for
  for $j = 1 \to m$ **do**
    $w_j \leftarrow w_j$
  end for
  for $i = 1 \to n$ **do** ▷ for data point, we create a tree
    for $j = 1 \to m$ **do**
      $u_j \leftarrow \langle x_i, w_j \rangle$
    end for
    $T_i \leftarrow$ MAKETREE$(u_1, \cdots, u_m)$ ▷ Each node stores the maximum value for his two children
  end for
end procedure
end data structure

Algorithm 6 Correlation DTrees

data structure CORRELATIONTREE ▷ Theorem 2.2.9
public:
procedure UPDATE($z \in \mathbb{R}^d, r \in [m]$) ▷ Lemma 2.2.11
    $w_r \leftarrow z$
    for $i = 1 \to n$ **do**
        $l \leftarrow$ the l-th leaf of tree T_i
        l.value $= \langle z, x_i \rangle$
        while l is not root **do**
            $p \leftarrow$ parent of l
            p.value $\leftarrow \max\{p.\text{value}, l.\text{value}\}$
            $l \leftarrow p$
        end while
    end for
end procedure
public:
procedure QUERY($i \in [n], \tau \in \mathbb{R}_{\geq 0}$) ▷ Lemma 2.2.12
    return QUERYSUB($\tau, \text{root}(T_i)$)
end procedure

private:
procedure QUERYSUB($\tau \in \mathbb{R}_{\geq 0}, r \in T$)
    if r is leaf **then**
        return r
    else
        $r_1 \leftarrow$ left child of r, $r_2 \leftarrow$ right child of r
        if r_1.value $\geq \tau$ **then**
            $S_1 \leftarrow$QUERYSUB(τ, r_1)
        end if
        if r_2.value $\geq \tau$ **then**
            $S_2 \leftarrow$QUERYSUB(τ, r_2)
        end if
    end if
    return $S_1 \cup S_2$
end procedure
end data structure

Running time

The goal of this secion is to prove the running time of INIT, UPDATE and QUERY.

Lemma 2.2.10 (Running time of INIT)**.** *Given a series of weights $\{w_1, w_2, \cdots, w_m\} \subset \mathbb{R}^d$ and datas $\{x_1, x_2, \cdots, x_n\} \subset \mathbb{R}^d$, it preprocesses in time $O(nmd)$*

Proof. The INIT consists of two independent forloop and two recursive forloops. The first forloop (start from line 10) has n interations, which takes $O(n)$ time. The second forloop (start from line 13) has m iterations, which takes $O(m)$ time. Now we consider the recursive forloop. The outer loop (line 16) has n iterations. In inner loop has m iterations. In each iteration of the inner loop, line 18 takes $O(d)$ time. Line 20 takes $O(m)$ time. Putting it all together, the running time of INIT is

$$\begin{aligned} & O(n + m + n(md + m)) \\ = & O(nmd) \end{aligned}$$

Thus, we complete the proof. □

Lemma 2.2.11 (Running time of UPDATE)**.** *Given a weight $z \in \mathbb{R}^d$ and index $j \in [m]$, it updates weight w_j with z in time $O(n \cdot (d + \log m))$*

Proof. The running time of UPDATE mainly comes from the forloop (line 5), which consists of n iterations. In each iteration, line 6 takes $O(\log m)$ time, line 7 takes $O(d)$ time and the while loop takes $O(\log m)$ time since it go through a path bottom up. Putting it together, the running time of UPDATE is $O(n(d + \log m))$. □

Lemma 2.2.12 (Running time of QUERY)**.** *Given a query $q \in \mathbb{R}^d$ and a threshold $\tau > 0$, it finds all index $i \in [n]$ such that $\langle w_i, q \rangle > \tau$ in time $O(|S(\tau)| \cdot \log m)$, where $S(\tau) := \{i : \langle w_i, q \rangle > \tau\}$*

Proof. The running time comes from QUERYSUB with input τ and $\mathrm{root}(T_i)$. In QUERYSUB, we start from the root node r and find indices in a recursive way. The INIT guarantees that for a node r satisfying $\mathrm{r.value} > \tau$, the sub-tree with root r must contains a leaf whose value is greater than τ If not satisfied, all the values of the nodes in the sub-tree with root r is less than τ. This guarantees that all the paths it search does not have any branches that leads to the leaf we don't need and it will report all the indiex i satisfying $\langle w_i, q \rangle > 0$. Note that the depth of T is $O(\log n)$, the running time of QUERY is $O(|S(\tau)| \cdot \log n)$ □

Correlation WTree data structure

We show that one can also preprocess all data points, and each query each of the weights.

Theorem 2.2.13 (Correlation WTree data structure, formal version of Theorem 1.1.17)**.** *There exists a data structure with the following procedures:*

- INIT($\{w_1, w_2, \cdots, w_m\} \subset \mathbb{R}^d, \{x_1, x_2, \cdots, x_n\} \subset \mathbb{R}^d, n \in \mathbb{N}, m \in \mathbb{N}, d \in \mathbb{N}$). *Given a series of weights $w_1, w_2, \cdots, w_m$ and datas $x_1, x_2, \cdots, x_n$ in d-dimensional space, it preprocesses in time $O(nmd)$*
- UPDATE($z \in \mathbb{R}^d, r \in [m]$). *Given a weight z and index r, it updates weight w_r with z in time $O(nd)$*

- QUERY($r \in [m], \tau \in \mathbb{R}$). *Given an index r indicating weight w_r and a threshold τ, it finds all index $i \in [n]$ such that $\langle w_r, x_i \rangle > \tau$ in time $O(|S(\tau)| \cdot \log m)$, where $S(\tau) := \{i : \langle w_r, x_i \rangle > \tau\}$.*

Algorithm

Algorithm 7 Correlation WTree data structure

data structure CORRELATIONWTREE ▷ Theorem 2.2.13
members
  $W \in \mathbb{R}^{m \times d}$ (m weight vectors)
  $X \in \mathbb{R}^{n \times d}$ (n data points)
  Binary tree $T_1, T_2, \cdots, T_M$ ▷ m binary search trees
end members

public:
procedure INIT($w_1, w_2, \cdots, w_m \in \mathbb{R}^d, m, x_1, x_2, \cdots, x_n \in \mathbb{R}^d$, n, m, d) ▷ Lemma 2.2.14
  for $i = 1 \to n$ **do**
    $x_i \leftarrow x_i$
  end for
  for $j = 1 \to m$ **do**
    $w_j \leftarrow w_j$
  end for
  for $i = 1 \to m$ **do** ▷ for weight, we create a tree
    for $j = 1 \to n$ **do**
      $u_j \leftarrow \langle x_i, w_j \rangle$
    end for
    $T_i \leftarrow$ MAKETREE$(u_1, \cdots, u_n)$ ▷ Each node stores the maximum value for his two children
  end for
end procedure
end data structure

Algorithm 8 Correlation WTrees

data structure CORRELATIONWTREE ▷ Theorem 2.2.13
public:
procedure UPDATE($z \in \mathbb{R}^d, r \in [m]$) ▷ Lemma 2.2.15
    $w_r \leftarrow z$
    for $j = 1 \to n$ **do**
        $u_j \leftarrow \langle x_j, w_r \rangle$
        $T_i \leftarrow$ MAKETREE$(u_1, \cdots, u_n)$ ▷ Each node stores the maximum value for his two children
    end for
end procedure
public:
procedure QUERY($r \in [m], \tau \in \mathbb{R}_{\geq 0}$) ▷ Lemma 2.2.16
    return QUERYSUB$(\tau, \mathrm{root}(T_r))$
end procedure

private:
procedure QUERYSUB($\tau \in \mathbb{R}_{\geq 0}, r \in T$)
    if r is leaf **then**
        return r
    else
        $r_1 \leftarrow$ left child of r, $r_2 \leftarrow$ right child of r
        if r_1.value $\geq \tau$ **then**
            $S_1 \leftarrow$QUERYSUB(τ, r_1)
        end if
        if r_2.value $\geq \tau$ **then**
            $S_2 \leftarrow$QUERYSUB(τ, r_2)
        end if
    end if
    return $S_1 \cup S_2$
end procedure
end data structure

Running time

The goal of this secion is to prove the running time of INIT, UPDATE and QUERY.

Lemma 2.2.14 (Running time of INIT)**.** *Given a series of weights* $\{w_1, w_2, \cdots, w_m\} \subset \mathbb{R}^d$ *and datas* $\{x_1, x_2, \cdots, x_n\} \subset \mathbb{R}^d$*, it preprocesses in time* $O(nmd)$

Proof. The INIT consists of two independent forloop and two recursive forloops. The first forloop (start from line 10) has n interations, which takes $O(n)$ time. The second forloop (start from line 13) has m iterations, which takes $O(m)$ time. Now we consider the recursive forloop. The outer loop (line 16) has m iterations. In inner loop has n iterations. In each iteration of the inner loop, line 18 takes $O(d)$ time. Line 20 takes $O(n)$ time. Putting it all together, the running

time of INIT is

$$\begin{aligned} & O(n + m + m(nd + n)) \\ = & O(nmd) \end{aligned}$$

Thus, we complete the proof. □

Lemma 2.2.15 (Running time of UPDATE)**.** *Given a weight $z \in \mathbb{R}^d$ and index $r \in [m]$, it updates weight w_j with z in time $O(nd)$*

Proof. In this procedure, it generates a new tree for weight w_r with n leaves, which takes $O(nd)$ time. Thus, we complete the proof. □

Lemma 2.2.16 (Running time of QUERY)**.** *Given a query $q \in \mathbb{R}^d$ and a threshold $\tau > 0$, it finds all index $i \in [n]$ such that $\langle w_i, q \rangle > \tau$ in time $O(|S(\tau)| \cdot \log m)$, where $S(\tau) := \{i : \langle w_i, q \rangle > \tau\}$*

Proof. The running time comes from QUERYSUB with input τ and $\mathrm{root}(T_i)$. In QUERYSUB, we start from the root node r and find indices in a recursive way. The INIT guarantees that for a node r satisfying $\mathrm{r.value} > \tau$, the sub-tree with root r must contains a leaf whose value is greater than τ If not satisfied, all the values of the nodes in the sub-tree with root r is less thanτ. This guarantees that all the paths it search does not have any branches that leads to the leaf we don' want and it will report all the indiex i satisfying $\langle w_i, q \rangle > 0$. Note that the depth of T is $O(\log n)$, the running time of QUERY is $O(|S(\tau)| \cdot \log n)$ □

2.3 Approximate Furthest Neighbor Search

In this section, we present data structure that solves Task 1.1.6 approximately but efficiently. To motivate our data structure, we note that finding the smallest inner product is equivalent of finding a vector that maximizes its distance with the query point q, i.e., the x_i such that $\|x_i - q\|_2$ is maximized. This is the so-called *furthest neighbor search* data structure. While deterministic implementations of such data structure exists, they typically suffer from the curse of dimensionality [1]. Hence, we consider the approximate variant of such data structure.

For later discussions, we will assume the dataset X are in unit Euclidean ball, and so are all query points q. In general, we have no control over the norm of the query point q and X. To address this issue, we consider an inner product preserve transformation by [59]:

Definition 2.3.1 ([59])**.** Given the query set $X \subset \mathbb{R}^d$ and a dataset $Y \subset \mathbb{R}^d$, we performs the following transformations for any $x \in X$ and $y \in Y$.

$$\varphi(x) = \begin{bmatrix} \frac{x^\top}{D_X} & 0 & \sqrt{1 - \frac{\|x\|_2^2}{D_X^2}} \end{bmatrix}^\top, \psi(y) = \begin{bmatrix} \frac{y^\top}{D_Y} & \sqrt{1 - \frac{\|y\|_2^2}{D_Y^2}} & 0 \end{bmatrix}^\top,$$

where D_X is larger than the maximum diameter of X and and D_Y is larger than the maximum diameter of Y. In this way, we map $x \in X$ and $y \in Y$ to unit vectors. In this way, we have $\|\varphi(x) - \psi(y)\|_2^2 = 2 - 2\langle \varphi(x), \psi(y) \rangle$. Moreover, we have $\arg\min_{y \in Y} \|\varphi(x) - \psi(y)\|_2 = \arg\max_{y \in Y} \langle x, y \rangle$ and $\arg\max_{y \in Y} -\|\varphi(x) - \psi(y)\|_2 = \arg\min_{y \in Y} \langle x, y \rangle$

Remark 2.3.2. *In our later applications, we implicitly assume all points have undergone such transformations in preprocessing phase. We also remark that in query phase, the set Y consists of a single query point, it suffices to pick D_Y as $\|y\|_2$, in this case, the transformation can be viewed as normalizing the query vector. If computing the inner product between x and y is required, we can retrieve the original x and y by its first d dimension, and by storing D_X as a variable in the data structure.*

For data structures presented in this section and Section 2.4, we assume all vectors in dataset and query point have undergone such transformations. Throughout this section, we will use n to denote the number of points in dataset and d to denote the dimension of points.

Definition 2.3.3 (Min-IP)**.** Given an n-point dataset $P \subset \mathbb{S}^{d-1}$ on the sphere, the goal of the Minimum Inner Product Search (Min-IP) is to build a data structure that, given a query $q \in \mathbb{S}^{d-1}$, retrieve the solution of $\arg\min_{p\in P}\langle p, q\rangle$.

The naive brutal force algorithm solves Min-IP in $O(nd)$ time. However, there exists algorithms that achieve time complexity sublinear in n with relaxation on the retrieved vector. These algorithms aim at solving the approximate Min-IP problem.

Definition 2.3.4 (Approximate Min-IP)**.** Let $c \in (0, 1)$ and $\tau \in (0, 1)$. Given an n-point dataset $P \subset \mathbb{S}^{d-1}$ on the sphere, the goal of the (c, τ)-Minimum Inner Product Search (Min-IP) is to build a data structure that, given a query $q \in \mathbb{S}^{d-1}$ with the promise that $\min_{p\in P}\langle p, q\rangle \leq \tau$, it reports a point $p' \in P$ with similarity $\langle p', q\rangle \leq \tau/c$.

The approximate Min-IP has a dual problem: approximate furthest neighbor (AFN). We could solve approximate Min-IP via solving AFN. To illustrate this, we first present the definition of AFN.

Definition 2.3.5 (Approximate Furthest Neighbor (AFN))**.** Let $\overline{c} > 1$ and $r \in (0, 2)$. Given an n-point dataset $P \subset \mathbb{S}^{d-1}$ on the sphere, the goal of the $(\overline{c}, r)$-Approximate Furthest-Neighbor (AFN) problem is to build a data structure that, given a query $q \in \mathbb{S}^{d-1}$ with the promise that $\max_{p\in P}\|p - q\|_2 \geq r$, it reports a point $p' \in P$ with distance $\|p' - q\|_2 \geq r/\overline{c}$.

Next, we show the connection between approximate Min-IP and AFN. In this discussion, we assume all vectors are unit vectors, later we'll see a transformation realizes this guarantee.

Lemma 2.3.6. *Given an n-point dataset $P \subset \mathbb{S}^{d-1}$ and a query point $q \in \mathbb{S}^{d-1}$, suppose for some $\overline{c} > 1$ and $r \in (0, 2)$, we have a $(\overline{c}, r)$-AFN data structure, then we can solve the (c, τ)-Min-IP problem for*

$$\tau = 1 - 0.5r^2, c = \frac{1 - 0.5r^2}{1 - 0.5r^2/\overline{c}^2}.$$

Proof. For any two points x, y with $\|x\|_2 = \|y\|_2 = 1$, we have $\|x - y\|_2^2 = 2 - 2\langle x, y\rangle$. This implies that if we have $\|x_i - q\|_2^2 \geq r^2$, then we have $\langle x_i, q\rangle \leq 1 - 0.5r^2$. Moreover, if we find a x_j such that $\|x_j - q\|_2^2 \geq r^2/\overline{c}^2$, then we have $\langle x_j, q\rangle \leq 1 - 0.5r^2/\overline{c}^2$. If we set $\tau = 1 - 0.5r^2$ and $c = \frac{1-0.5r^2}{1-0.5r^2/\overline{c}^2}$, then the above inner product guarantee becomes

$$\begin{aligned}\langle x_j, q\rangle &\leq 1 - 0.5r^2/\overline{c}^2 \\ &= 1 - 0.5r^2 + (1 - 1/\overline{c}^2)0.5r^2 \\ &= \tau + (1 - \frac{c - \tau}{c(1 - \tau)})(1 - \tau)\end{aligned}$$

$$= \tau/c$$

where the second-to-last line is because

$$\begin{aligned}\overline{c}^2 &= \frac{cr^2}{2c-2+r^2} \\ &= \frac{c(2-2\tau)}{2c-2+(2-2\tau)} \\ &= \frac{c(1-\tau)}{c-\tau}. \end{aligned} \tag{2.1}$$

This indicates that if we have a data structure for $(\overline{c}, r)$-AFN, it automatically becomes a data structure for (c, τ)-Min-IP with τ and c chosen as above. □

Next, we explore some structures on the function $\frac{c(1-\tau)}{c-\tau}$. We show that it increases as τ increases.

Lemma 2.3.7. *Let $c \in (0,1)$ and $\tau \in (0,1)$, we show that function $f(c,\tau) := \frac{c(1-\tau)}{c-\tau}$ is decreasing as c increase and increasing as τ increase.*

Proof. We take the derivative of $f(c,\tau)$ over c and get

$$\frac{\partial}{\partial c} f(c,\tau) = \frac{(\tau-1)\tau}{(c-\tau)^2} < 0$$

where the second step follows from $c > \tau$ and $\tau < 1$.

Therefore, $f(c,\tau) := \frac{c(1-\tau)}{c-\tau}$ is decreasing as c increase.

We take the derivative of $f(c,\tau)$ over τ and get

$$\begin{aligned}\frac{\partial}{\partial \tau} f(c,\tau) &= \frac{c(\tau^2 - 2c\tau + c)}{(c-\tau)^2} \\ &= \frac{c((\tau-c)\tau + c(1-\tau))}{(c-\tau)^2} > 0\end{aligned}$$

where the second step follows from $c > \tau$ and $\tau < 1$.

Therefore, $f(c,\tau) := \frac{c(1-\tau)}{c-\tau}$ is increasing as τ increases. □

We augment the Min-IP definition to tolerate additive errors.

Definition 2.3.8 (Additive approximate Min-IP)**.** Let $c \in (0,1)$ and $\tau \in (0,1)$. Let $\lambda \geq 0$. Given an n-point dataset $Y \subset \mathbb{S}^{d-1}$, the goal of the (c, τ, λ)-Min-IP is to build a data structure, given a query $x \in \mathbb{S}^{d-1}$ with the promise that $\min_{p \in P} \langle p, q \rangle \leq \tau$, it reports a data point $z \in Y$ such that $\langle x, z \rangle \leq c^{-1} \min_{p \in P} \langle p, q \rangle + \lambda$.

2.3.1 From AFN to approximate Min-IP

The AFN data structure we will be using is inspired by [39]. We include its complete algorithm and analysis in Appendix A.

We restate Theorem A.4.2 here.

Lemma 2.3.9 (Informal version of Theorem A.4.2)**.** *Let $P \subset \mathbb{R}^d$ be an n-point dataset, $\overline{c} > 1$, $r > 0$ and $\delta > 0$. Let $\varepsilon = \overline{c} - 1$. There exists a randomized dynamic data structure (against an oblivious adversary) that solves $(\overline{c}+\delta, r)$-AFN task using space $O((n^{1+1/\overline{c}^2}\log n + dn^{1/\overline{c}^2}\log n)\log\log(d/\varepsilon\delta) + dn)$ with the following operations:*

- INIT*: Preprocess P in $O((n^{1+1/\overline{c}^2}\log^2 n + dn^{1/\overline{c}^2}\log n)\log\log(d/\varepsilon\delta))$ time;*
- QUERY*: Given a point $q \in \mathbb{R}^d$, returns a $(\overline{c}+\delta)$-approximate furthest neighbor $p \in P$ with constant probability in $O(n^{1/\overline{c}^2}(d+\log n)\log n\log(d/\varepsilon\delta)\log\log(d/\varepsilon\delta))$ time;*
- INSERT*: Insert a point $p \in \mathbb{R}^d$ into the data structure in $O(n^{1/\overline{c}^2}\log^2 n\log\log(d/\varepsilon\delta) + d\log n)$ time;*
- DELETE*: Delete a point $p \in \mathbb{R}^d$ from the data structure in $O(n^{1/\overline{c}^2}\log^2 n\log\log(d/\varepsilon\delta) + d\log n)$ time.*

Next, we introduce several corollaries that simplify the time complexity in solving AFN.

Corollary 2.3.10. *Let $P \subset \mathbb{R}^d$ be an n-point dataset, $\overline{c} > \sqrt{2}$, and $r > 0$. There exists a randomized dynamic data structure (Alg. 26, 27) that solves $(2\overline{c}, r)$-AFN with query time $O(n^{0.5}(d+\log n)\log n\log d\log\log d)$, preprocessing time $O((n^{1.5}\log^2 n + dn^{0.5}\log n)\log\log d)$ and space $O((n^{1.5}\log^2 n + dn^{0.5}\log n)\log\log d + nd)$. Moreover, the dynamic data structure supports insert or delete in $O(n^{0.5}\log^2 n\log\log d + d\log n)$ time.*

Proof. If $\overline{c} > \sqrt{2}$, we have $1/\overline{c}^2 < 0.5$. We take this fact into the preprocessing, query, insert and delete time and get the following:

Space

$$O((n^{1+1/\overline{c}^2}\log^2 n + dn^{1/\overline{c}^2}\log n)\log\log(d/\varepsilon\delta) + nd) = O((n^{1.5}\log^2 n + dn^{0.5}\log n)\log\log d + nd)$$

Preprocesing time

$$O((n^{1+1/\overline{c}^2}\log^2 n + dn^{1/\overline{c}^2}\log n)\log\log(d/\varepsilon\delta)) = O((n^{1.5}\log^2 n + dn^{0.5}\log n)\log\log d)$$

Query time

$$O(n^{1/\overline{c}^2}(d+\log n)\log n\log(d/\varepsilon\delta)\log\log(d/\varepsilon\delta)) = O(n^{0.5}(d+\log n)\log n\log d\log\log d)$$

Insert/delete time

$$O(n^{1/\overline{c}^2}\log^2 n\log\log(d/\varepsilon\delta) + d\log n) = O(n^{0.5}\log^2 n\log\log d + d\log n)$$

□

Corollary 2.3.11. *Let $P \subset \mathbb{R}^d$ be an n-point dataset, $\overline{c} > 10$, and $r > 0$. There exists a randomized dynamic data structure (Alg. 26, 27) that solves $(2\overline{c}, r)$-AFN with query time $O(n^{0.01}(d+\log n)\log n\log d\log\log d)$, preprocessing time $O((n^{1.01}\log^2 n + dn^{0.01}\log n)\log\log d)$ and space $O((n^{1.01}\log^2 n + dn^{0.01}\log n)\log\log d + nd)$. Moreover, the dynamic data structure supports insert or delete in $O(n^{0.01}\log^2 n\log\log d + d\log n)$ time.*

Proof. If $\overline{c} > 10$, we have $1/\overline{c}^2 < 0.01$. We take this fact into the preprocessing, query, insert and delete time and get the following:

Space

$$O((n^{1+1/\overline{c}^2}\log^2 n + dn^{1/\overline{c}^2}\log n)\log\log(d/\varepsilon\delta) + nd) = O((n^{1.01}\log^2 n + dn^{0.01}\log n)\log\log d + nd)$$

Preprocesing time

$$O((n^{1+1/\overline{c}^2}\log^2 n + dn^{1/\overline{c}^2}\log n)\log\log(d/\varepsilon\delta)) = O((n^{1.01}\log^2 n + dn^{0.01}\log n)\log\log d)$$

Query time

$$O(n^{1/\overline{c}^2}(d+\log n)\log n\log(d/\varepsilon\delta)\log\log(d/\varepsilon\delta)) = O(n^{0.01}(d+\log n)\log n\log d\log\log d)$$

Insert/delete time

$$O(n^{1/\overline{c}^2}\log^2 n\log\log(d/\varepsilon\delta) + d\log n) = O(n^{0.01}\log^2 n\log\log d + d\log n)$$

□

Using the AFN data structure, we design an approximate Min-IP search data structure.

Algorithm 9 Minimum Inner Product Search

```
1:  data structure MINIMUM INNER PRODUCT SEARCH                ▷ Theorem 2.3.24
2:  members
3:      APPROXIMATEFURTHESTNEIGHBOR AFN
4:  end members
5:
6:  procedure INIT(x_1, x_2, ··· , x_n, c̄, r)
7:      AFN.INIT(x_1, x_2, ··· , x_n, c̄, r)
8:  end procedure
9:
10: procedure INSERT(z ∈ R^d)
11:     AFN.INSERT(z)
12: end procedure
13:
14: procedure DELETE(i ∈ [n])
15:     AFN.DELETE(i)
16: end procedure
17:
18: procedure QUERYMIN(q ∈ R^d)
19:     x_i ← AFN.QUERY(q)
20:     return x_i
21: end procedure
22: end data structure
```

We take the adaptive query in iterative optimization algorithm into consideration and design a robust algorithm against adversary. Before proceeding to the main theorem of this section, we first consider a technical lemma regarding quantization.

Lemma 2.3.12. *Let $c \in (0,1)$, $\tau \in (0,1)$ and $\lambda \in (0,1)$. Given a set of n-points $Y \subset \mathbb{S}^{d-1}$, one can construct a data structure with $\mathcal{T}_{\mathrm{init}} \cdot \kappa$ preprocessing time and $\mathcal{S}_{\mathrm{space}} \cdot \kappa$ space so that for every $x \in \mathbb{S}^{d-1}$ in an adaptive sequence $X = \{x_1, \ldots, x_T\}$, the query time is $\widetilde{O}(dn^{0.01} \cdot \kappa)$:*

- *If* Min-IP$(x, Y) \leq \tau$*, then we output a vector in Y which is a (c, τ, λ)-*Min-IP *with respect to (x, Y) with probability at least $1 - \delta$.*
- *Otherwise, we output* fail.

where $\kappa := d \log(ndD_X/(\lambda\delta))$, and D_X is the diameter of all queries in X.

Proof. The failure probability for an adaptive sequence X is equivalent to the probability that at least one query $\widehat{q} \in \widehat{Q}$ fail in solving all κ number of (c, τ)-Min-IP. We bound this failure probability as

$$\Pr[\exists \widehat{q} \in \widehat{Q} \ \text{ s.t all } \ (c,\tau)\text{-Min-IP fail}] = n \cdot (\frac{dD_X}{\lambda})^d \cdot (1/10)^\kappa \leq \delta$$

where the last step follows from $\kappa := d \log(ndD_X/(\lambda\delta))$.

For the success queries, it introduces a λ error in the inner product. Thus, the results is (c, τ, λ)-Max-IP.

Then, following Corollary 2.3.11, we finish the proof. □

2.3.2 Efficient and Adaptive Sketchings for Tensors

Often times, the geometric search data structures are used in conjunction with dimensionality reduction tools, such as Johnson-Lindentrauss transform [43]. However, since we will use the data structure in an iterative process, it is important to make sure that it is robust against adaptive inputs. We therefore augment the standard Johnson-Lindenstrauss in a way so that its guarantee works for *all* vectors, instead of a fixed set of vectors.

Also, note Task 1.1.6 is defined on the inner products in the form of $\langle x_i x_i^\top, Q \rangle$, which means we will pass in rank 1 matrices into our data structure. To speeding up the construction of $x_i x_i^\top = x_i \otimes x_i$, we introduce a new type of Johnson-Lindentrasuss transforms that are sparse, and succeed with high probability. In fact, we generalize the sparse embedding [26, 29, 45, 46] matrix to process tensor-typed inputs fast.

Throughout this section, we will use m to denote the number of data points, d to denote the dimension of points, b to denote the sketching dimension and s to denote the column sparsity.

We first recall the Johnson-Lindenstrauss transform [43]:

Definition 2.3.13 (Johnson-Lindenstrauss transform (JLT))**.** Let $\{x_1, \ldots, x_m\} \in (\mathbb{R}^d)^m$, we say a distribution Π over $s \times d$ matrices is a (m, ε, δ)-JLT if for any $S \sim \Pi$, we have

$$\Pr[\|S(x_i - x_j)\|_2^2 \leq (1 \pm \varepsilon)\|x_i - x_j\|_2^2] \geq 1 - \delta, \ \ \forall (i,j) \in [m] \times m.$$

We remark that in order to obtain this property for all m^2 pairs of point, it suffices to obtain the following guarantee for any fixed point $x \in \mathbb{R}^d$:

$$\Pr[\|Sx\|_2^2 \leq (1 \pm \varepsilon)\|x\|_2^2] \geq 1 - \delta,$$

then union bound over all m^2 pairs of points, we are done.

2.3.3 TensorSparse: Efficient Tensor Product in Input-Sparsity Time

We recall the sparse embedding matrix [26, 29, 45, 46].

Definition 2.3.14. Let $h : [d] \times [s] \to [b/s]$ be a random $O(\log 1/\delta)$-wise independent hash function and $\sigma : [d] \times [s] \to \{\pm 1\}$ be $O(\log 1/\delta)$-wise independent. Then $R \in \mathbb{R}^{b \times d}$ is a sparse embedding matrix with sparsity parameter s if we set $R_{(j-1)b/s+h(i,j),i} = \sigma(i,j)/\sqrt{s}$ for all $(i,j) \in [d] \times [s]$ and all other entries to 0.

Alternatively, we can define the following:

$$R_{r,i} = \exists k \in [s] : \sigma(i,k)/\sqrt{s} \cdot \mathbf{1}[h(i,k) + (k-1)b/s = r]$$

We extend the construction of sparse embedding to handle tensor product of vectors, specifically, our goal is to design a sparse matrix that is similar to Def. 2.3.14, so that we can enjoy certain nice properties, such as it is a $(1, \varepsilon, \delta)$-JLT with $b = O(\varepsilon^{-2}\log(1/\delta))$, this again enables us to union bound over m points.

Definition 2.3.15 (TensorSparse)**.** Let $h_1, h_2 : [d] \times [s] \to [b/s]$ be $O(\log 1/\delta)$-wise independent hash functions and let $\sigma_1, \sigma_2 : [d] \times [s] \to \{\pm 1\}$ be $O(\log 1/\delta)$-wise independent random sign functions. Then, the degree two tensor sparse transform, $R : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}^b$ is given as:

$$R_{r,(i,j)} = \exists k \in [s] : \sigma_1(i,k)\sigma_2(j,k)/\sqrt{s} \cdot \mathbf{1}[((h_1(i,k) + h_2(j,k)) \bmod b/s) + (k-1)b/s = r]$$

We will show that for any fixed unit vector $x \in \mathbb{R}^{d^2}$, Rx preserves the length of x with good probability. To do so, we first exhibit some properties of our sketch.

Lemma 2.3.16. *The degree two* TensorSparse *transform (Def. 2.3.15) has the following property. We define $\delta_{r,(i,j)}$ as the Bernoulli random variable on whether the entry $R_{r,(i,j)}$ is non-zero or not. Then*

1. *Each column has support size s.*
2. *For all $r \in [b]$ and $(i,j) \in [d] \times [d]$, $\mathbb{E}[\delta_{r,(i,j)}] = s/b$.*
3. *Negative correlations of $\delta_{r,(i,j)}$'s defined as follows:*

$$\forall T \subset [b] \times [d] \times [d] \text{ and } |T| \leq \Theta(\log(1/\delta)), \quad \mathbb{E}\Big[\prod_{r,(i,j)\in T} \delta_{r,(i,j)}\Big] \leq \prod_{r,(i,j)\in T} \mathbb{E}[\delta_{r,(i,j)}] = \Big(\frac{s}{b}\Big)^{|T|}.$$

Proof. We prove three parts separately.

Part 1. To see each column has support size s, we partition each column into s blocks, where each block contains b/s entries and then show that each block has exactly 1 non-zero entry. Fix the block to be the k-th block and consider the (i,j)-th column, then we are looking at the values of hash functions $(h_1(i,k) + h_2(j,k)) \bmod b/s$, since both h_1 and h_2 have their ranges being $[b/s]$, this means $(h_1(i,k) + h_2(j,k)) \bmod b/s$ must have its value being in the range of $[b/s]$, and its value corresponding to the entry that is non-zero.

Part 2. We will again use the block-partition view and consider the k-th block of (i,j)-th column. For each index r, the probability that it is non-zero is equal to the probability that $(h_1(i,k) + h_2(j,k)) \bmod b/s = r - (k-1)b/s$. We first observe that if we are using a single 3-wise independent hashing function, then this probability is naturally s/m. Here, the hashing function we are considering is $H(i,j,k) := h_1(i,k) + h_2(j,k) \bmod b/s$, it is well-known that H

is also $\Theta(\log(1/\delta))$-wise independent [19, 64]. We hence conclude that $\Pr[\delta_{r,(i,j)} = 1] = \frac{s}{b}$ and therefore, $\mathbb{E}[\delta_{r,(i,j)}] = \frac{s}{b}$.

Part 3. To see the negative correlation, we let $t = |T|$, and we denote the elements in T as $(r_1, l_1), \ldots, (r_t, l_t)$. We define the following indicator random variable: $\mathbf{1}[\exists (r_i, l_i), (r_j, l_j) \in T$ s.t. $r_i \neq r_j$ belong to the same block and $l_i = l_j]$.

Note that if such event happens, then $\mathbb{E}[\prod_{(r,l)\in T} \delta_{r,l}] = 0$ since we can write it as

$$\begin{aligned}\mathbb{E}[\prod_{(r,l)\in T} \delta_{r,l}] &= \Pr[\bigwedge_{r,l\in T} \delta_{r,l} = 1] \\ &= \Pr[\delta_{r_1,l_1} = 1 \wedge \delta_{r_2,l_2} = 1] \cdot \Pr[\bigwedge_{(r,l)\in T, r\neq r_1,r_2, l\neq l_1,l_2} \delta_{r,l} = 1 \mid \delta_{r_1,l_1} = 1 \wedge \delta_{r_2,l_2} = 1].\end{aligned}$$

When the above event happens, then we are considering the case that $r_1 \neq r_2$ but they belong to the same block, and the column is the same. By construction, for each column, there is exactly one non-zero entry. Hence, $\Pr[\delta_{r_1,l_1} = 1 \wedge \delta_{r_2,l_2} = 1] = 0$, and we conclude the expectation is 0.

Suppose the above event does not happen, then we will make use the fact that our hashing function H is $\Theta(\log(1/\delta))$-wise independent, and $\delta_{r,l} = 1$ is equivalent to for some $k \in [s]$, we have $H(l, k) = r$. The above event does not happen is equivalent to

$$\begin{aligned}\Pr[\bigwedge_{(r,l)\in T} \exists k \in [s], H(l,k) = r] &= \prod_{(r,l)\in T} \Pr[\exists k \in [s], H(l,k) = r] \\ &= \prod_{(r,l)\in T} \Pr[\delta_{r,l} = 1] \\ &= \prod_{(r,l)\in T} \mathbb{E}[\delta_{r,l}],\end{aligned}$$

where the first step is due to H is $\Theta(\log(1/\delta))$-wise independence. Therefore, we conclude that the random variables $\delta_{r,l}$'s are negatively correlated. □

Remark 2.3.17. *We note that we only require our hashing function H and sign function σ to be $\Theta(\log(1/\delta))$-wise independent, since in our later proofs, we will only consider the q-th power of an expression Z which involves the term $\prod_{(r,l)\in T} \delta_{r,l}$ for $|T| \leq q$. Thus, the expectation of Z are term-by-term dominated by the case that all $\delta_{r,l}$ are i.i.d. Bernoulli with expectation s/b. This justifies our later use of Lemma 2.1.5 and Hanson-Wright inequality.*

We will adapt an analysis from [26] to conclude that TensorSparse is a JLT:

Lemma 2.3.18. *If R is a* TensorSparse *matrix as defined in Def. 2.3.15, with target dimension $m \geq \Omega(\log(1/\delta)/\varepsilon^2)$ and sparsity parameter $s = \varepsilon m$, then*

$$\Pr[|\|Rx\|_2^2 - 1| > \varepsilon] \leq \delta.$$

Proof. We first observe that

$$\|Rx\|_2^2 = \frac{1}{s} \sum_{r=1}^{b} \sum_{i,j=1}^{d^2} \delta_{r,i}\delta_{r,j}\sigma_{r,i}\sigma_{r,j}x_i x_j$$

$$= \frac{1}{s}\sum_{r=1}^{b}\sum_{i=1}^{d^2}\delta_{r,i}x_i^2 + \frac{1}{s}\sum_{r=1}^{b}\sum_{i\neq j}^{d^2}\delta_{r,i}\delta_{r,j}\sigma_{r,i}\sigma_{r,j}x_ix_j,$$

for the first term (diagonal term), we have

$$\begin{aligned}\frac{1}{s}\sum_{r=1}^{b}\sum_{i=1}^{d^2}\delta_{r,i}x_i^2 &= \sum_{i=1}^{d^2}x_i^2(\frac{1}{s}\sum_{r=1}^{b}\delta_{r,i}) \\ &= \|x\|_2^2 \\ &= 1,\end{aligned}$$

where the second step follows from the fact that each column of R has support size s. We define the intermediate variable $Z := \|Rx\|_2^2 - 1$, which as shown by proceeding calculations, captures the off-diagonal term. Consider the following terms: we first define $A_{x,\delta}$ which is a block diagonal matrix with b blocks, where the k-th block is defined as $\frac{1}{s}x^{(k)}(x^{(k)})^\top$ but with the diagonal zeroed out, with $(x^{(k)})_i = \delta_{k,i}x_i$. Note that by construction, $A_{x,\delta} \in \mathbb{R}^{bd^2\times bd^2}$. We further define the following length bd^2 vector $\sigma \in \mathbb{R}^{bd^2}$, where $\sigma_{r,i}$ is the sign generated for the entry (r,i) of R.

It is not hard to see that $Z = \frac{1}{s}\sum_{r=1}^{b}\sum_{i\neq j}^{d^2}\delta_{r,i}\delta_{r,j}\sigma_{r,i}\sigma_{r,j}x_ix_j = \sigma^\top A_{x,\delta}\sigma$. Let $\|X\|_{L_q} := (\mathbb{E}[|X|^q])^{1/q}$. Since σ is a vector with each entry being independent Rademacher random variable, by Hanson-Wright inequality, we have

$$\begin{aligned}\|\sigma^\top A_{x,\delta}\sigma\|_{L_q} &\leq \|\sqrt{q}\cdot\|A_{x,\delta}\|_F + q\cdot\|A_{x,\delta}\|\|_{L_q} \\ &\leq \sqrt{q}\cdot\|\|A_{x,\delta}\|_F\|_{L_q} + q\cdot\|\|A_{x,\delta}\|\|_{L_q},\end{aligned}$$

since $A_{x,\delta}$ is block diagonal, its spectral norm is the largest spectral norm of any block. Note that the spectral norm of k-th block is

$$\begin{aligned}\|\frac{1}{s}\cdot x^{(k)}(x^{(k)})^\top\| &\leq \frac{1}{s}\cdot\|x^{(k)}\|_2^2 \\ &\leq \frac{1}{s},\end{aligned}$$

where the first step is the sub-multiplicativity of spectral norm and the spectral norm of a vector is its ℓ_2 norm, and the second line follows from $\|x^{(k)}\|_2 \leq \|x\|_2 = 1$.

Next, we define $Q_{i,j} = \sum_{r=1}^{b}\delta_{r,i}\delta_{r,j}$, so

$$\begin{aligned}\|A_{x,\delta}\|_F^2 &= \frac{1}{s^2}\sum_{r=1}^{b}\sum_{i\neq j}^{d^2}\delta_{r,i}\delta_{r,j}x_i^2x_j^2 \\ &= \frac{1}{s^2}\sum_{i\neq j}^{d^2}Q_{i,j}x_i^2x_j^2.\end{aligned}$$

Recall that for any column i of R, there exists exactly s non-zero entries, so we suppose $\delta_{r_t,i} = 1$ for all distinct r_t.

Consider the event that $\delta_{r_t,j} = 1$, and let Y_t be the indicator random variable for this event. By Lemma 2.3.16, we assume Y_t's are independent, so that the sum $Q_{i,j} = \sum_{t=1}^{s} Y_t$ has the distribution of $\mathsf{Binomial}(s, s/b)$. Combining with Lemma 2.1.5, we have that $\|Q_{i,j}\|_{L_{q/2}} \leq q/2$. Thus,

$$\begin{aligned}
\|\|A_{x,\delta}\|_F\|_{L_q} &= \|\|A_{x,\delta}\|_F^2\|_{L_{q/2}}^{1/2} \\
&= \|\frac{1}{s^2}\sum_{i\neq j} x_i^2 x_j^2 Q_{i,j}\|_{L_{q/2}}^{1/2} \\
&\leq \frac{1}{s}(\sum_{i\neq j} x_i^2 x_j^2 \|Q_{i,j}\|_{L_{q/2}})^{1/2} \\
&\leq O\left(\frac{\sqrt{q}}{s}\right).
\end{aligned}$$

Put things together, we have

$$\|\sigma^\top A_{x,\delta}\sigma\|_{L_q} \leq O\left(\frac{q}{s}\right). \tag{2.2}$$

Set $q = \Theta(\log(1/\delta)) = \Theta(s^2/b)$, we have $\|Z\|_{L_q} \leq O(\frac{s}{b})$, then by Markov inequality, we have

$$\Pr[|\|Rx\|_2^2 - 1| > \varepsilon] = \Pr[|\sigma^\top A_{x,\delta}\sigma| > \varepsilon] < \varepsilon^{-q} \cdot C^q(m^{-q/2} + s^{-q}) < \delta,$$

as desired. □

Note that our construction resembles the TensorSketch matrix [11, 61], more specifically, we can view our tensor sparse embedding as s distinct TensorSketch matrices, each with dimension $b/s \times d^2$. Hence, to compute the tensor product between two vectors, we can run the TensorSketch algorithm for s blocks, yielding an overall running time of $O(s \cdot (\mathrm{nnz}(x) + \mathrm{nnz}(y)) + b\log(b/s))$ for computing $S(x \otimes y)$.

We summarize the JLT result and efficient computation of tensor in the following theorem:

Theorem 2.3.19. *Let $\{x_1, \ldots, x_m\} \in (\mathbb{R}^{d^2})^m$. Let $\varepsilon \in (0,1)$ be precision parameter and $\delta \in (0,1)$ be success probability. Let $R \in \mathbb{R}^{b\times d^2}$ be a* TensorSparse *matrix (Def. 2.3.15). Suppose $b = \Omega(\varepsilon^{-2}\log(m/\delta))$ and $s = \varepsilon m$ be the sparsity parameter, then we have R is an (m, ε, δ)-JLT (Def. 2.3.13).*

Moreover, if $x = u \otimes v$ for some $u, v \in \mathbb{R}^d$, then Rx can be computed in time $O(s \cdot (\mathrm{nnz}(u) + \mathrm{nnz}(v)) + b\log(b/s))$.

Proof. The JLT result is by apply union bound over all m^2 pairs of points using Lemma 2.3.18. The running time is by using the TensorSketch algorithm for s blocks. □

For further applications, we prove a simple result regarding the Frobenius norm of R.

Lemma 2.3.20. *Let $R \in \mathbb{R}^{b\times d^2}$ be a* TensorSparse *matrix (Def. 2.3.15), then we have*

$$\|R\|_F = d.$$

Proof. We observe that each column of R has exactly s non-zero entries, each has magnitude $\frac{1}{\sqrt{s}}$, hence each column is a unit length vector. There are d^2 columns in total, yielding a Frobenius norm of d. □

2.3.4 Robust Sketches Against Adaptive Adversary

We note that the above discussion only applies when we consider an *independent* set of points, i.e., all points we want to preserve using TensorSRHT or TensorSparse are picked oblivious with respect to the randomness of the sketch. However, this is no longer the case for our application — specifically, the query we send for iteration $t+1$ is *dependent* on the answer we receive at iteration t.

One idea is to require a sketching matrix that preserves the length of *all* vectors in a subspace. Unfortunately, this will result in a sketching dimension of roughly $\Theta(d^2/\varepsilon^2)$, which essentially diminishes the necessity of using sketching. To address this problem, we exploit the following idea: we use a number of independent sketches of small dimension, and we show that with high probability, a good fraction of them will do well on a (potentially) adversary query. We will show that the dimension-saving by using lower-dimensional sketching matrices will have to be paid back by the number of sketches required. However, this has one distinctive advantage for our applications: we will then operate our AFN data structures on much lower dimensions, hence the preprocessing time and query time can be significantly improved.

We prove the following lemma:

Lemma 2.3.21. *Let $V := \{v_1, \ldots, v_m\} \in (\mathbb{R}^d)^m$, $\varepsilon \in (0,1)$ and $\delta \in (0,1)$. Furthermore, let $\{S_i\}_{i=1}^k \subset \mathbb{R}^{b\times d}$ for $k \geq \Omega((d+\log(1/\delta))\log(md))$ such that each S_i is an independent $(m+1,\varepsilon,0.99)$-JLT matrix (Def. 2.3.13) with $\|S_i\|_F \leq d$. Then we have*

$$\forall q \in \mathbb{S}^{d-1}, \forall v \in V, \sum_{i=1}^{k} \mathbf{1}[\|S_i(q-v)\|_2^2 \leq (1 \pm O(\varepsilon))\|q-v\|_2^2 + \alpha] \geq 0.95k$$

with probability at least $1-\delta$ and $\alpha \leq O(\frac{1}{(md)^9})$.

Proof. We will prove via a standard γ-net argument. Let N be a γ-net of $\mathbb{S}^{d-1}$ with $\gamma = \frac{c}{(md)^{10}}$ for some small enough constant c, and it is not hard to see that $|N| \leq (md)^{O(d)}$. Let $u \in N$, define the following event:

$$W_i(u) = \|S_i u\|_2^2 \leq (1+O(\varepsilon)) \text{ and } \forall v_i, v_j \in V, |u^\top S_i^\top S_i(v_i - v_j) - u^\top(v_i - v_j)| \leq O(\varepsilon)\|v_i - v_j\|_2,$$

i.e., the length of u is preserved by S_i and for any pair of points in V, the inner product is also preserved by S_i. We note that we only need this property to hold with respect to the set of points $V \cup \{u\}$, since S_i is a $(m+1,\varepsilon,0.99)$-JLT, we know this event holds with probability at least 0.99.

By an application of Hoeffding's inequality on the random variables $\sum_{i=1}^k W_i(u)$, we have that

$$\Pr[\sum_{i=1}^{k} W_i(u) \leq 0.97k] \leq \exp(-2k),$$

we then union bound over all points in N:

$$\Pr[\forall u \in N, \sum_{i=1}^{k} W_i(u) \leq 0.97k] \leq \exp(-2k) \cdot (md)^{O(d)}$$

$$= (\frac{1}{md})^{O(d)} \cdot (md)^{O(d)} \cdot \exp(-\log(1/\delta)\log(md))$$
$$\leq \delta/4.$$

We will condition on this event happen throughout the rest of the proof. To extend this bound from all points in N to the entire unit sphere, consider any $q \in \mathbb{S}^{d-1}$ and pick a net point $u \in N$ such that $\|q-u\|_2 \leq \gamma$. Let $i \in [k]$ be the index such that $W_i(u)$ happens. We shall bound the term $\|S_i(q-v)\|_2$ for $v \in V$:

$$\begin{aligned}\|S_i(q-v)\|_2 &\leq \|S_i(q-u)\|_2 + \|S_i(u-v)\|_2 \\ &\leq d \cdot \gamma + (1 \pm O(\varepsilon))\|u-v\|_2 \\ &\leq d \cdot \gamma + (1 \pm O(\varepsilon))(\|q-v\|_2 - \gamma) \\ &= (1 \pm O(\varepsilon))\|q-v\|_2 + (d - (1 \pm O(\varepsilon)))\gamma \\ &\leq (1 \pm O(\varepsilon))\|q-v\|_2 + \alpha.\end{aligned}$$

The conclusion of the lemma follows. □

Remark 2.3.22. *We note that by using the γ-net argument, we get a weaker conclusion compared to standard Johnson-Lindenstrauss lemma, namely, we preserve the distance with $(1 \pm O(\varepsilon))$ relative error and α additive error. Fortunately, the magnitude of α is small enough so that it won't affect the quality of our downstream task too much.*

As an example, consider the following adaptive robust AFN*: we use k different independent data structures where each one has an independent JLT matrix S_i. At each query point q, we shall sample $\Theta(\log b)$ data structures and output the one with the best quality.*

As a direct consequence, we have the following result for TensorSparse:

Corollary 2.3.23. *Let $V := \{v_1, \ldots, v_m\} \in (\mathbb{R}^d)^m$, $\varepsilon \in (0,1)$ and $\delta \in (0,1)$. Furthermore, let $\{R_i\}_{i=1}^k \in \mathbb{R}^{b \times d}$ for $k \geq \Omega((d + \log(1/\delta))\log(md))$ such that each R_i is an independent* TensorSparse *matrix with $b = \Theta(\varepsilon^{-2}\log m)$ rows and $\|R_i\|_F = d$. Then we have*

$$\forall q \in \mathbb{S}^{d-1}, \forall v \in V, \sum_{i=1}^{k} \mathbf{1}[\|S_i(q-v)\|_2^2 \leq (1 \pm O(\varepsilon))\|q-v\|_2^2 + \alpha] \geq 0.95k$$

with probability at least $1-\delta$ and $\alpha \leq O(\frac{1}{(md)^9})$.

Proof. The result follows from Theorem 2.3.19 and Lemma 2.3.21. □

2.3.5 Putting Things Together

Now we have a powerful dimensionality reduction tool that computes the tensor product fast and robust against adaptive adversary, and an efficient approximate furthest neighbor search data structure that are also robust against adaptive inputs. We combine them together to get our meta result in this section.

Theorem 2.3.24 (Formal version of Theorem 1.1.14)**.** *Let $c \in (0,1)$, $\tau \in (0,1)$, $\lambda \in (0,1)$, $\varepsilon \in (0,1)$ and $\delta \in (0,1)$. We define the following additional parameters:*

- $\alpha = O(\frac{1}{(nd)^9})$, *the additive error by Lemma 2.3.21;*
- $s \leq d$, *the dimension of JLT;*
- $k = O((d + \log(1/\delta)) \log(nd))$, *number of independent JLT sketches;*
- $\kappa = s \log(ns/(\lambda\delta))$*;*
- $\widetilde{\lambda} = O(\sqrt{\frac{c-\tau}{c(1-\tau)}}) \cdot (\lambda + \alpha)$, *the additive error of* Min-IP.

Let $\mathcal{T}_S(x)$ denote the time of applying S to a vector $x \in \mathbb{R}^d$. Given a set of n-points $Y \subset \mathbb{S}^{d-1}$ on the sphere, one can build a dynamic data structure with preprocessing time $\mathcal{T}_{\mathsf{init}} \cdot \kappa \cdot k + \mathcal{T}_S(Y) \cdot k$, space $\mathcal{S}_{\mathsf{space}} \cdot \kappa \cdot k$ insert time $(\mathcal{T}_{\mathsf{insert}} \cdot \kappa + \mathcal{T}_S(x)) \cdot k$ and delete time $(\mathcal{T}_{\mathsf{delete}} \cdot \kappa + \mathcal{T}_S(x)) \cdot k$ so that for every query $x \in \mathbb{S}^{d-1}$ in an adaptive sequence $X = \{x_1, \ldots, x_T\}$, the query time is $\widetilde{O}(\mathcal{T}_{\mathsf{query}} \cdot \kappa + \mathcal{T}_S(x))$:

- *if* Min-IP$(x, Y) \leq \tau$*, then we output a vector in Y that is a $(c, \tau, \widetilde{\lambda})$-*Min-IP *with respect to (x, Y).*
- *otherwise, we output* fail.

Further,

- *If $c \in (\tau, \frac{8\tau}{(1-\varepsilon)^2\tau + 2\varepsilon + 7})$, we have $\mathcal{T}_{\mathsf{init}} = O((n^{1.5} \log^2 n + sn^{0.5} \log n) \log\log s)$, $\mathcal{S}_{\mathsf{space}} = O((n^{1.5} \log^2 n + sn^{0.5} \log n) \log\log s + ns)$, $\mathcal{T}_{\mathsf{query}} = O(n^{0.5}(s + \log n) \log n \log s \log\log s)$ and $\mathcal{T}_{\mathsf{insert}} = \mathcal{T}_{\mathsf{delete}} = O(n^{0.5} \log^2 n \log\log s + s \log n)$*
- *If $c \in (\tau, \frac{400\tau}{(1-\varepsilon)^2\tau + 2\varepsilon + 399})$, we have $\mathcal{T}_{\mathsf{init}} = O((n^{1.01} \log^2 n + sn^{0.01} \log n) \log\log s)$, $\mathcal{S}_{\mathsf{space}} = O((n^{1.01} \log^2 n + sn^{0.01} \log n) \log\log s + ns)$, $\mathcal{T}_{\mathsf{query}} = O(n^{0.01}(s + \log n) \log n \log s \log\log s)$ and $\mathcal{T}_{\mathsf{insert}} = \mathcal{T}_{\mathsf{delete}} = O(n^{0.01} \log^2 n \log\log s + s \log n)$*

Finally, the probability that all queries succeed is at least $1 - \delta$.

Proof. We first use Lemma 2.3.21 to initiate $k \geq \Omega((d + \log(1/\delta)) \log(nd))$ different JLT matrices with parameters $(m + 1, \varepsilon, 0.99)$. Then, for each JLT matrix $S_i \in \mathbb{R}^{s \times d}$, we run the quantization process on it. Specifically, this requires us to use $\kappa = s \log(ns/(\lambda\delta))$ independent AFN data structures due to Lemma 2.3.12.

Throughout the proof, we will condition on the event that there exists some $i \in [k]$ such that S_i preserves the pair-wise distances between any query point and points in X. To simplify the notation, we use S to denote the corresponding JLT matrix S_i.

We consider the following: given a query point $Sx \in \mathbb{R}^s$, we quantize it into a point $\widehat{x} \in \mathbb{R}^s$, then we use $\widehat{x}$ as our query. Let Sy be the furthest neighbor of $\widehat{x}$, the AFN data structure will output a point Sy' with the guarantee that $\|Sy' - \widehat{x}\|_2 \geq \|Sy - \widehat{x}\|_2 / \overline{c}$. Towards the end, we wish to have a bound on the term $\|x - y'\|_2$ in terms of $\|x - y\|_2$.

$$\begin{aligned}
\|Sy' - Sx\|_2 &= \|Sy' - \widehat{x} + \widehat{x} - Sx\|_2 \\
&\geq \|Sy' - \widehat{x}\|_2 - \|Sx - \widehat{x}\|_2 \\
&\geq \|Sy - \widehat{x}\|_2 / \overline{c} - \lambda \\
&\geq \|Sy - Sx + Sx - \widehat{x}\|_2 / \overline{c} - \lambda \\
&\geq (\|Sy - Sx\|_2 - \lambda) / \overline{c} - \lambda \\
&\geq \overline{c}^{-1} \cdot ((1 - \varepsilon)\|y - x\|_2 - \alpha - \lambda) - \lambda,
\end{aligned}$$

on the other hand, we know that $\|x-y'\|_2 \geq \frac{\|Sy'-Sx\|_2-\alpha}{1+\varepsilon}$, we hence conclude that

$$\begin{aligned}\|x-y'\|_2 &\geq \frac{\overline{c}^{-1}\cdot(1-\varepsilon)\|x-y\|_2-(1+\overline{c}^{-1})\lambda-(1+\overline{c}^{-1})\alpha}{1+\varepsilon}\\ &= \underbrace{\overline{c}^{-1}\cdot(1-O(\varepsilon))}_{\widetilde{c}^{-1}}\|x-y\|_2-\underbrace{(1-O(\varepsilon))\cdot((1+\overline{c}^{-1})\cdot\lambda+(1+\overline{c}^{-1})\cdot\alpha)}_{\widetilde{\lambda}}.\end{aligned}$$

By further setting $\widetilde{r}=\frac{r}{1-\varepsilon}$, we conclude we get a $(\widetilde{c},\widetilde{r})$-AFN data structure with additive error $\widetilde{\lambda}$. Moreover, this $(\widetilde{c},\widetilde{r})$-AFN data structure would also be a data structure for (c,τ)-Min-IP with $\tau=1-0.5\widetilde{r}^2$ and $c=\frac{1-0.5r^2}{1-0.5\widetilde{r}^2/\widetilde{c}^2}$. Using Eq. (2.1), we have $\widetilde{c}^2=\frac{c(1-c\tau)}{c-\tau}$.

Next, we present how to obtain the desired query, preprocessing insert, and delete time complexity in the statement.

Part 1. Let $\widetilde{c}=2\overline{c}/(1-\varepsilon)$, we conclude that $\overline{c}^2=\frac{c(1-c\tau)(1-\varepsilon)^2}{4(c-\tau)}$. If $\tau\in(0,1)$ and $c\in(\tau,\frac{8\tau}{(1-\varepsilon)^2\tau+2\varepsilon+7})$, we have

$$\begin{aligned}\overline{c}^2 &= \frac{c(1-\tau)(1-\varepsilon)^2}{4(c-\tau)}\\ &> (1-\varepsilon)^2\cdot\frac{8\tau}{(1-\varepsilon)^2\tau+2\varepsilon+7})\cdot\frac{1-\tau}{4(\frac{8\tau}{(1-\varepsilon)^2\tau+2\varepsilon+7})-\tau)}\\ &> (1-\varepsilon)^2\cdot\frac{8\tau}{(1-\varepsilon)^2\tau-(1-\varepsilon)^2+8})\cdot\frac{1-\tau}{4(\frac{8\tau}{(1-\varepsilon)^2\tau-(1-\varepsilon)^2+8})-\tau)}\\ &= \frac{2(1-\varepsilon)^2\tau(1-\tau)}{8\tau-(1-\varepsilon)^2\tau^2+(1-\varepsilon)^2\tau-8\tau}\\ &= 2\end{aligned}$$

where the second and third steps follow from Lemma 2.3.7.

Then, we use Corollary 2.3.10 with $\overline{c}^2>2$ and obtain the query time $O(n^{0.5}(s+\log n)\log n\log s\log\log s)$, preprocessing time $O((n^{1.5}\log^2 n+sn^{0.5}\log n)\log\log s)$ and space $O((n^{1.5}\log^2 n+sn^{0.5}\log n)\log\log s+ns)$. Moreover, the dynamic data structure supports insert or delete in $O(n^{0.5}\log^2 n\log\log s+s\log n)$ time.

Part 2. Let $\widetilde{c}=2\overline{c}/(1-\varepsilon)$, we conclude that $\overline{c}^2=\frac{c(1-c\tau)(1-\varepsilon)^2}{4(c-\tau)}$. If $\tau\in(0,1)$ and $c\in(\tau,\frac{400\tau}{(1-\varepsilon)^2\tau+2\varepsilon+399})$, we have

$$\begin{aligned}\overline{c}^2 &= \frac{c(1-\tau)(1-\varepsilon)^2}{4(c-\tau)}\\ &> (1-\varepsilon)^2\cdot\frac{400\tau}{(1-\varepsilon)^2\tau+2\varepsilon+399})\cdot\frac{1-\tau}{4(\frac{400\tau}{(1-\varepsilon)^2\tau+2\varepsilon+399})-\tau)}\\ &> (1-\varepsilon)^2\cdot\frac{400\tau}{(1-\varepsilon)^2\tau-(1-\varepsilon)^2+400})\cdot\frac{1-\tau}{4(\frac{400\tau}{(1-\varepsilon)^2\tau-(1-\varepsilon)^2+400})-\tau)}\\ &= \frac{100(1-\varepsilon)^2\tau(1-\tau)}{400\tau-(1-\varepsilon)^2\tau^2+(1-\varepsilon)^2\tau-400\tau}\end{aligned}$$

$= 10$

where the second and third steps follow from Lemma 2.3.7.

Then, we use Corollary 2.3.11 with $\overline{c}^2 > 100$ and obtain the query time $O(n^{0.01}(s+\log n)\log n \log s \log\log s)$, preprocessing time $O((n^{1.01}\log^2 n + sn^{0.01}\log n)\log\log s)$ and space $O((n^{1.01}\log^2 n + sn^{0.01}\log n)\log\log s + ns)$. Moreover, the dynamic data structure supports insert or delete in $O(n^{0.01}\log^2 n \log\log s + s\log n)$ time.

Next, we analyze the additive error. Use the relationship $\overline{c}^2 = \frac{c(1-\tau)(1-\varepsilon)^2}{4(c-\tau)}$ we derived above, we can further simplify $\widetilde{\lambda}$:

$$(1 - O(\varepsilon)) \cdot ((1 + \overline{c}^{-1}) \cdot \lambda + (1 + \overline{c}^{-1}) \cdot \alpha) \leq O(1) \cdot \sqrt{\frac{c-\tau}{c(1-\tau)}} \cdot (\lambda + \alpha).$$

Therefore, we simplify $\widetilde{\lambda} \leq O(\sqrt{\frac{c-\tau}{c(1-\tau)}} \cdot (\lambda+\alpha))$, we conclude that we get a $(c, \tau, \widetilde{\lambda})$-Min-IP. □

2.4 Adaptive Inner Product Estimations

In this section, we design a data structure with the following query feature: given a query vector $q \in \mathbb{R}^d$ and a preprocessed dataset $\{x_1, \ldots, x_m\} \subset \mathbb{R}^d$, it approximately estimates all the inner products $\langle q, x_i \rangle$.

The data structure builds upon the adaptive distance estimation data structures introduced in [22, 23]. Such data structures can output estimates to $\|q - x_i\|_2$ for any $i \in [m]$, we use an inner product preserve reduction to show that it also preserves the inner product.

While such data structure is useful for Task 1.1.9, we will mainly use it for Task 1.1.6. This is because, for the optimization problems that require us to realize Task 1.1.9 have more structures, and thus simpler and more efficient data structures can be adapted.

Definition 2.4.1 (Adaptive Inner Product Estimation (AIPE))**.** Let $X = \{x_1, \ldots, x_m\} \in (\mathbb{R}^d)^m$ be a dataset of dimension d and radius D and let $q \in \mathbb{R}^d$ be a query point in unit Euclidean ball. The *Adaptive Inner Product Estimation* (AIPE) data structure, D, has the following guarantee: with probability at least $1 - \delta$ we have for any $i \in [m]$,

$$(1 + \varepsilon)\langle x_i, q \rangle - D\varepsilon \leq w_i \leq (1 - \varepsilon)\langle x_i, q \rangle + D\varepsilon,$$

where w_i denotes the inner product estimation between x_i and q.

We have the following result from [23]:

Lemma 2.4.2 (Theorem 1.4 of [23])**.** *Let $\varepsilon, \delta \in (0, 1/2)$. Then, there exists a data structure for Distance Estimation in Euclidean space which is initialized correctly with probability at least $1 - \delta$ and supports the following operations:*

- *Output a correct answer to a possibly adaptively chosen distance estimation query with probability at least $1 - \delta$, i.e.,*

$$(1 - \varepsilon)\|x_i - q\|_2 \leq d_i \leq (1 + \varepsilon)\|x_i - q\|_2,$$

where d_i denotes the distance estimation between x_i and q.

- *Add input $x \in \mathbb{R}^d$ to the dataset X.*

Furthermore, the query and update (insert/delete) time of the data structure are $\widetilde{O}(\varepsilon^{-2}(m + d)\log 1/\delta)$ and $\widetilde{O}(\varepsilon^{-2}d\log 1/\delta)$ respectively while the data structure is constructed in time $\widetilde{O}(\varepsilon^{-2}md\log 1/\delta)$.

Now, we are ready to present our AIPE data structure.

Algorithm 10 Adaptive Inner Product Estimation

data structure ADAPTIVE INNER PRODUCT ESTIMATION ▷ Theorem 2.4.6
members
  ADAPTIVEDISTANCEESTIMATION ADE
end members

procedure INIT($x_1, x_2, \cdots, x_m, \varepsilon, \delta$)
  ADE.INIT($x_1, x_2, \cdots, x_m, \varepsilon, \delta$)
end procedure

procedure INSERT($z \in \mathbb{R}^d$)
  ADE.INSERT(z)
end procedure

procedure DELETE($i \in [m]$)
  ADE.DELETE(i)
end procedure

procedure QUERY($q \in \mathbb{R}^d$) ▷ Lemma 2.4.5
  $d_1, d_2, \cdots, d_m \leftarrow$ ADE.QUERY(q)
  for $i = 1, 2, \cdots, m$ **do**
    $w_i = 1 - \frac{1}{2}d_i^2$
  end for
  return $\{w_i\}_{i=1}^m$
end procedure

procedure QUERYMIN($q \in \mathbb{R}^d$) ▷ Lemma 2.4.3
  $d_1, d_2, \cdots, d_m \leftarrow$ ADE.QUERY(q)
  $i \leftarrow \arg\max_{i \in [m]} d_i$
  return x_i
end procedure
end data structure

We first show that given an ADE data structure, we can solve the AFN data structure problem.

Lemma 2.4.3. *Let $X = \{x_1, \ldots, x_m\} \in (\mathbb{S}^{d-1})^m$ be the dataset and $q \in \mathbb{S}^{d-1}$ be a query vector. Suppose for some $r \in (0, 2)$, $\max_{x \in X} \|x - q\|_2 \geq r$. Then, procedure* QUERYMIN(q) *in Algorithm 10 solves the $(1 + \varepsilon, r)$-*AFN *data structure problem.*

Proof. Let $x \in X$ be the point in X that maximizes the distance with q, also, we have $\|x-q\|_2 \geq r$.Let d_x denote the distance estimation corresponds to x outputted by the ADE data structure. Suppose for some $y \in X$, $d_y \geq d_x$, then we have

$$\begin{aligned} d_y &\geq d_x \\ &\geq (1-\varepsilon)\|x-q\|_2 \\ &\geq (1-\varepsilon)r \\ &\geq r/(1+2\varepsilon), \end{aligned}$$

this concludes our proof. □

As a corollary, it automatically induces a Min-IP data structure.

Corollary 2.4.4. *Let $X = \{x_1, \ldots, x_m\} \in (\mathbb{S}^{d-1})^m$ be the dataset and $q \in \mathbb{S}^{d-1}$ be a query vector. Suppose for some $r \in (0,2)$, $\max_{x \in X} \|x-q\|_2 \geq r$. Given a $(1+\varepsilon, r)$-AFN data structure, it can solve the (c,τ)-Min-IP problem with*

$$\tau = 1 - 0.5r^2,\ c = \frac{(1+\varepsilon)^2\tau}{(1+\varepsilon)^2 - 1 + \tau}.$$

Similarly, the AIPE problem can be solved using ADE.

Lemma 2.4.5. *Let $X = \{x_1, \ldots, x_m\} \subset \mathbb{R}^{d-1}$ be the dataset with m points and radius D, let $q \in \mathbb{S}^{d-1}$ be the query vector. The procedure* QUERY(q) *in Algorithm 10 outputs a list of estimates $\{w_i\}_{i=1}^m$ such that*

$$(1+\varepsilon)\langle x_i, q\rangle - D\varepsilon \leq w_i \leq (1-\varepsilon)\langle x_i, q\rangle + D\varepsilon.$$

Proof. Throughout the proof, we assume transformation Q has been applied to all points $x_i \in X$ and transformation P has been applied to query vector q.

By Definition 2.3.1, we have

$$\|P(q) - Q(x_i)\|_2^2 = 2 - 2 \cdot D^{-1}\langle q, x_i\rangle \tag{2.3}$$

By Lemma 2.4.2, we have

$$(1-\varepsilon)^2\|P(q)-Q(x_i)\|_2^2 \leq d_i^2 \leq (1+\varepsilon)^2\|P(q)-Q(x_i)\|_2^2, \forall i \in [n]$$

Then we have

$$1 - \frac{(1+\varepsilon)^2\|P(q)-Q(x_i)\|_2^2}{2} \leq 1 - \frac{d_i^2}{2} \leq 1 - \frac{(1-\varepsilon)^2\|P(q)-Q(x_i)\|_2^2}{2}$$

Applying Eq. (2.3) we get

$$1 - \frac{(1+3\varepsilon)(2 - 2\cdot D^{-1}\langle q, x_i\rangle)}{2} \leq 1 - \frac{d_i^2}{2} \leq 1 - \frac{(1-3\varepsilon)(2 - 2\cdot D^{-1}\langle q, x_i\rangle)}{2}$$

Thus, we get

$$(1+3\varepsilon)\langle q, x_i\rangle - 3D\varepsilon \leq D\cdot(1 - \frac{d_i^2}{2}) \leq (1-3\varepsilon)\langle q, x_i\rangle + 3D\varepsilon.$$

□

We summarize results regarding Algorithm 10 in the following main theorem.

Theorem 2.4.6 (Adaptive Inner Product Estimation, formal version of Theorem 1.1.15)**.** *There is a data structure uses $\widetilde{O}(\varepsilon^{-2}md\log(1/\delta))$ space for the Adaptive Inner Product Estimation Problem with the following procedures:*

- INIT$(\{x_1, x_2, \ldots, x_m\} \subset \mathbb{R}^d, \varepsilon \in (0,1), \delta \in (0,1))$*: Given data points $\{x_1, x_2, \ldots, x_n\} \subset \mathbb{R}^d$ with radius D, an accuracy parameter ε and a failure probability δ as input, the data structure preprocesses in time $\widetilde{O}(\varepsilon^{-2}md\log(1/\delta))$.*
- INSERT$(z \in \mathbb{R}^d)$*: Given a vector z, the data structure insert z in time $\widetilde{O}(\varepsilon^{-2}d\log(1/\delta))$.*
- DELETE$(i \in [m])$*: Given an index i, the data structure deletes x_i in time $\widetilde{O}(\varepsilon^{-2}d\log(1/\delta))$.*
- QUERY$(q \in \mathbb{R}^d)$*: Given a query point $q \in \mathbb{R}^d$, the* QUERY *operation takes q as input and approximately estimates the inner product of q and all the data points $\{x_1, x_2, \ldots, x_m\} \subset \mathbb{R}^d$ in time $\widetilde{O}(\varepsilon^{-2}(m+d)\log(1/\delta))$ i.e. it provides a set of estimates $\{\widetilde{w}_i\}_{i=1}^m$ such that:*

$$\forall i \in [m], (1+\varepsilon)\langle q, x_i\rangle - D\varepsilon \leq \widetilde{w}_i \leq (1-\varepsilon)\langle q, x_i\rangle + D\varepsilon$$

  with probability at least $1-\delta$, even for a sequence of adaptively chosen queries.
- QUERYMIN$(q \in \mathbb{R}^d)$*: Given a query point $q \in \mathbb{R}^d$, the* QUERYMIN *operation takes q as input and solves the $(1+\varepsilon, r)$-*AFN *data structure problem, where $r \in (0,2)$ satisfies $\max_{x\in X} \|x-q\|_2/D \geq r$, in time $\widetilde{O}(\varepsilon^{-2}d\log(1/\delta))$.*

Proof. **Proof of INIT.** The running time follows from the initialization time of Lemma 2.4.2.

Proof of INSERT and DELETE. The running time follows from the update time of Lemma 2.4.2.

Proof of QUERY. The correctness follows from Lemma 2.4.5, for the running time, it follows from Lemma 2.4.2.

Proof of QUERYMIN. The correctness follows from Lemma 2.4.3, for the running time, it follows from Lemma 2.4.2.

□

Remark 2.4.7. ADE *data structure is robust against adaptive queries, which is especially feasible during an iterative process. During query, to find the vector that approximates the minimum inner product, we need to perform a linear scan over all m vectors, this makes it useful when number of iterations is rather small, in which linear scan is affordable. The initialization time of the data structure is also nearly linear in the size of input.*

2.5 Low Rank Maintenance: Simple Restart

Low rank maintenance task (Task 1.1.10) is a simple yet very useful primitive that finds its applications in many optimization algorithms. Consider we are given a matrix $W \in \mathbb{R}^{m\times m}$, at each iteration, W receives a low rank update $UV^\top$, where $U, V \in \mathbb{R}^n$. Clearly, if $n \ll m$, then one can use a clever data structure to store W and its updates in a fashion such that the matrix-product query can be answered in time $o(m^2)$. We present a data structure that realizes this guarantee.

Before moving, we define some notions related to rectangular matrix multiplication.

Definition 2.5.1 ([8, 33, 85])**.** Let ω be the matrix multiplication exponent such that it takes $n^{\omega+o(1)}$ time to multiply two $n \times n$ matrices. Let α be the dual exponent of the matrix multiplication which is the supremum among all $a \geq 0$ such that it takes $n^{2+o(1)}$ time to multiply an $n \times n$ by $n \times n^a$ matrix.

Additionally, we define the function $\omega(\cdot)$ where $\omega(b)$ denotes the exponent of multiplying an $n \times n$ matrix by an $n \times n^b$ matrix. Hence, we have $\omega(1) = \omega$ and $\omega(\alpha) = 2$.

The overall idea of our low rank maintenance data structure is as follows: we keep accumulating the low rank change, when the rank of the change reaches a certain threshold (m^α), then we restart the data structure and update the weight matrix.

Algorithm 11 Low rank maintenance data structure

data structure LOWRANKMAINTENANCE ▷ Lemma 2.5.2
  members
    $r_\ell, \forall \ell \in [L]$ ▷ r_ℓ denotes the accumulated rank of the change
    $W_\ell, \forall \ell \in [L]$ ▷ $\{W_\ell\}_{\ell=1}^L \in (\mathbb{R}^{m \times m})^L$
    $\Delta W_\ell, \forall \ell \in [L]$ ▷ $\{\Delta W_\ell\}_{\ell=1}^L \in (\mathbb{R}^{m \times m})^L$
  end members

  procedures
    INIT$(\{W_1(0), \dots W_L(0)\})$ ▷ Initialize the data structure
    UPDATE(U_ℓ, V_ℓ) ▷ Update the low rank representation
    QUERY(ℓ, y) ▷ Compute the matrix-vector product between ΔW_ℓ and y
  end procedures
end data structure

Lemma 2.5.2 (Formal version of Theorem 1.1.18)**.** *There exists a deterministic data structure (Algorithm 11) such that maintains*

$$\Delta W_1, \dots, \Delta W_L$$

such that

- *The procedure* INIT *(Algorithm 12) takes $O(m^2 L)$ time.*
- *The procedure* UPDATE *(Algorithm 12) takes $O(nm^{2-\alpha+o(1)})$ amortized time, where $\alpha = \omega(2)$.*
- *The procedure* QUERY *(Algorithm 12) takes $O(m \cdot (\mathrm{nnz}(y) + r_\ell))$ time, where r_ℓ is the rank of ΔW_ℓ when* QUERY *is called.*

Proof. The runtime for INIT is straightforward, for QUERY, notice that we are multiplying vector y with a (possibly) dense matrix $W_\ell \in \mathbb{R}^{m \times m}$, which takes $O(\mathrm{nnz}(y) \cdot m)$ time, and an accumulated low rank matrix ΔW_ℓ with rank r_ℓ. By using the low rank decomposition $\Delta W_\ell = UV^\top$ with $U, V \in \mathbb{R}^{m \times r_\ell}$, the time to multiply y with ΔW is $O(mr_\ell)$. Combine them together, we get a running time of $O(m \cdot (\mathrm{nnz}(y) + r_\ell))$.

It remains to analyze the amortized cost of UPDATE. Note that if $r_\ell < m^a$, then we just pay $O(1)$ time to update corresponding variables in the data structure. If $r_\ell = m^a$, then we will

Algorithm 12 Procedures of LRM data structure

procedure INIT($\{W_1(0), \ldots, W_L(0)\}$) ▷ Lemma 2.5.2
  $W_\ell \leftarrow W_\ell(0)$
  $\Delta W_\ell \leftarrow 0, \forall \ell \in [L]$
  $r_\ell \leftarrow 0, \forall \ell \in [L]$
end procedure

procedure UPDATE($U_\ell \in \mathbb{R}^{m \times n}, V_\ell \in \mathbb{R}^{m \times n}$) ▷ Lemma 2.5.2
  $\Delta W_\ell \leftarrow \Delta W_\ell + U_\ell V_\ell^\top$ without forming the product and sum the two matrices
  $r_\ell \leftarrow r_\ell + n$
  if $r_\ell = m^a$ where $a = \omega(2)$ **then**
    $W_\ell \leftarrow W_\ell + \Delta W_\ell$ ▷ Takes $O(m^2)$ time
    $r_\ell \leftarrow 0$
    $\Delta W_\ell \leftarrow 0$
  end if
end procedure

procedure QUERY($\ell \in [L], y \in \mathbb{R}^m$) ▷ Lemma 2.5.2
  $z \leftarrow W_\ell \cdot y + \Delta W_\ell \cdot y$ ▷ Takes $O(\mathrm{nnz}(y) \cdot m + m r_\ell)$ time
  return z
end procedure

explicitly form the $m \times m$ matrix ΔW_ℓ. To form it, notice we have accumulated r_ℓ/n different sums of rank-n decompositions, which can be represented as

$$U = [U_\ell(1), U_\ell(2), \ldots, U_\ell(r_\ell/n)] \in \mathbb{R}^{m \times r_\ell},$$
$$V = [V_\ell(1), V_\ell(2), \ldots, V_\ell(r_\ell/n)] \in \mathbb{R}^{m \times r_\ell},$$

and $\Delta W_\ell = UV^\top$, which takes $O(m^{2+o(1)})$ time to compute since $r_\ell = m^a$ and $a = \omega(2)$. Finally, note that this update of W_ℓ only happens once per r_ℓ/n number of calls to UPDATE, therefore we can charge each step by $O(\frac{m^2}{r_\ell/n}) = O(m^{2-a}n) = O(m^{2-\alpha}n)$, arrives at our final amortized running time. □

Remark 2.5.3. *Currently, the dual matrix multiplication exponent $\alpha \approx 0.31$ [33], hence the amortized time for* UPDATE *is $O(nm^{1.69})$. If $m \geq n^4$, then we achieve an update time of $o(m^2)$. Similarly, the time for* QUERY *is $O(m \cdot (\mathrm{nnz}(y) + r_\ell)) = O(m \cdot \mathrm{nnz}(y) + m^{1+\alpha}) = O(m \cdot \mathrm{nnz}(y) + m^{1.31})$, as long as $\mathrm{nnz}(y) = o(m)$, then its running time is also $o(m^2)$.*

2.6 Projection Maintenance via Inverse Maintenance and the Power of Sketching

Given a diagonal matrix $W \in \mathbb{R}^{n \times n}$ with non-negative entries on the diagonal, and $A \in \mathbb{R}^{d \times n}$ where $d \leq n$ with rank d, the goal is to maintain the projection $P = \sqrt{W}A^\top (AWA^\top)^{-1} A\sqrt{W}$

under ℓ_2 multiplicative changes to W and support the matrix vector product Ph for some vector $h \in \mathbb{R}^n$. This is a key core step in speeding up the robust interior point method for linear programming [27, 72] and empirical risk minimization [55]. We present a unified framework that uses Schur completement to reduce the projection maintenance into an inverse maintenance, coupled with the clever use of sketching techniques to reduce the query complexity.

Fact 2.6.1 (Schur complement)**.** *Given four matrices A, B, C, D, we have the following identity assuming that all inverses exist:*

$$\begin{bmatrix} A & B \\ C & D \end{bmatrix}^{-1} = \begin{bmatrix} A^{-1} + A^{-1}B(D - CA^{-1}B)^{-1}CA^{-1} & -A^{-1}B(D - CA^{-1}B)^{-1} \\ -(D - CA^{-1}B)^{-1}CA^{-1} & (D - CA^{-1}B)^{-1} \end{bmatrix}$$

2.6.1 From Projection to Inverse Maintenance

While projection maintenance has rather complicated form, we can pack the corresponding terms into a large matrix and use Schur complement in a clever way.

Lemma 2.6.2 (Original version)**.** *Let $A \in \mathbb{R}^{d \times n}$ be a matrix of rank d and let $W \in \mathbb{R}^{n \times n}$ be a diagonal matrix with non-zero diagonal entries and let $h \in \mathbb{R}^n$. Then*

$$\begin{bmatrix} W^{-1} & A^\top & \sqrt{W}^{-1} & 0 \\ A & 0 & 0 & 0 \\ 0 & 0 & -I & 0 \\ (\sqrt{W}^{-1})^\top & 0 & 0 & -I \end{bmatrix}^{-1} \begin{bmatrix} \mathbf{0}_n \\ \mathbf{0}_d \\ -h \\ -h \end{bmatrix} = \begin{bmatrix} \star \\ \star \\ \star \\ \sqrt{W}A^\top(AUA^\top)^{-1}A\sqrt{W}h \end{bmatrix} \tag{2.4}$$

where $\star$ represents some entries that do not care about.

Proof. Let $A = W^{-1} \in \mathbb{R}^{n \times n}, B = A^\top \in \mathbb{R}^{n \times d}, C = A \in \mathbb{R}^{d \times n}, D = 0 \in n^{d \times d}$ in Fact 2.6.1, then the matrix has full-rank (i.e. it is invertible) and the top-left block of the inverse is $W - WA^\top(AWA^\top)^{-1}AW \in \mathbb{R}^{n \times n}$. Further, consider the following block-matrix and its inverse:

$$\begin{bmatrix} M & N & 0 \\ 0 & -I & 0 \\ N^\top & 0 & -I \end{bmatrix}^{-1} = \begin{bmatrix} M^{-1} & M^{-1}N & 0 \\ 0 & -I & 0 \\ N^\top M^{-1} & N^\top M^{-1}N & -I \end{bmatrix}$$

Note that

$$M = \begin{bmatrix} W^{-1} & A^\top \\ A & 0 \end{bmatrix}$$

By the Fact 2.6.1 and set $A = W^{-1} \in \mathbb{R}^{n \times n}, B = A^\top \in \mathbb{R}^{n \times d}, C = A \in \mathbb{R}^{d \times n}, D = 0 \in \mathbb{R}^{d \times d}$, then we have

$$M^{-1} = \begin{bmatrix} W + WA^\top(0 - AWA^\top)^{-1}AW & -WA^\top(0 - AWA^\top)^{-1} \\ -(0 - AWA^\top)AW & (0 - AWA^\top)^{-1} \end{bmatrix}$$

When $M \in \mathbb{R}^{(n+d) \times (n+d)}$ is the previous block-matrix and $N \in \mathbb{R}^{(n+d) \times n}$ block-matrix $(\sqrt{W}^{-1}, 0_{n \times d})^\top$, i.e.,

$$N = \begin{bmatrix} \sqrt{W}^{-1} \\ \mathbf{0}_{d \times n} \end{bmatrix}$$

Then we compute the $N^\top M^{-1} N \in \mathbb{R}^{n \times n}$:

$$\begin{aligned} N^\top M^{-1} N &= \sqrt{W}^{-1} (W - WA^\top (AWA^\top)^{-1} AW) \sqrt{W}^{-1} \\ &= I - \sqrt{W} A^\top (AWA^\top)^{-1} A \sqrt{W} \end{aligned}$$

Consider the matrix multiplication in Eq. (2.4) and its last n coordinates, we have

$$\begin{aligned} \begin{bmatrix} N^\top M^{-1} & N^\top M^{-1} N & -I \end{bmatrix} \begin{bmatrix} \mathbf{0}_{n+d} \\ -h \\ -h \end{bmatrix} &= 0 - N^\top M^{-1} N h + h \\ &= -(I - \sqrt{W} A^\top (AWA^\top) A \sqrt{W}) h + h \\ &= \sqrt{W} A^\top (AWA^\top) A \sqrt{W} h \end{aligned}$$

Therefore we know that Eq. (2.4) holds and complete the proof. □

Sometimes, we want to put sketching matrix on the left or on the right. By slightly modifying the matrix, we can also achieve these objectives.

Definition 2.6.3 (M matrix)**.** Let $A \in \mathbb{R}^{d \times n}$ with rank d and $W \in \mathbb{R}^{n \times n}$ be a diagonal matrix with non-zero elements on the diagonal. We define the matrix $M \in \mathbb{R}^{(n+d) \times (n+d)}$ as follows:

$$M = \begin{bmatrix} W^{-1} & A^\top \\ A & 0 \end{bmatrix}$$

Lemma 2.6.4. *Let $M \in \mathbb{R}^{(n+d) \times (n+d)}$ be defined as in Definition 2.6.3, then*

$$M^{-1} = \begin{bmatrix} W - WA^\top (AWA^\top)^{-1} AW & WA^\top (AWA^\top)^{-1} \\ (AWA^\top)^{-1} AW & -(AWA^\top)^{-1} \end{bmatrix}$$

Proof. We have

$$\begin{aligned} M^{-1} &= \left(\begin{bmatrix} W^{-1} & A^\top \\ A & 0 \end{bmatrix} \right)^{-1} \\ &= \begin{bmatrix} W + WA^\top (0 - AWA^\top)^{-1} AW & -WA^\top (0 - AWA^\top)^{-1} \\ -(0 - AWA^\top)^{-1} AW & (0 - AWA^\top)^{-1} \end{bmatrix} \\ &= \begin{bmatrix} W - WA^\top (AWA^\top)^{-1} AW & WA^\top (AWA^\top)^{-1} \\ (AWA^\top)^{-1} AW & -(AWA^\top)^{-1} \end{bmatrix} \end{aligned}$$

where the first step follows from Fact 2.6.1 and the second step comes from simplifying the terms. □

Definition 2.6.5 (L matrix)**.** Let $A \in \mathbb{R}^{d \times n}$ be rank d and $W \in \mathbb{R}^{n \times n}$ be a diagonal matrix with non-zero elements on the diagonal. We define the matrix $L \in \mathbb{R}^{(3n+d) \times (3n+d)}$ as follows

$$L = \begin{bmatrix} W^{-1} & A^\top & W^{-1/2} & 0 \\ A & 0 & 0 & 0 \\ 0 & 0 & -I & 0 \\ (W^{-1/2})^\top & 0 & 0 & -I \end{bmatrix}$$

To get a better view of the inverse of L, we define the matrix N first.

Definition 2.6.6 (N matrix)**.** Let $W \in \mathbb{R}^{n\times n}$ be a diagonal matrix with non-zero diagonal element. We define the matrix $N \in \mathbb{R}^{(n+d)\times n}$ as follows:

$$N = \begin{bmatrix} W^{-1/2} \\ \mathbf{0}_{d\times n} \end{bmatrix}$$

We can express the inverse of $L \in \mathbb{R}^{(3n+d)\times(3n+d)}$ using $M \in \mathbb{R}^{(n+d)\times(n+d)}$ and $N \in \mathbb{R}^{(n+d)\times n}$.

Lemma 2.6.7. *Let $L \in \mathbb{R}^{(3n+d)\times(3n+d)}$ be defined in Definition 2.6.5. Then,*

$$L^{-1} = \begin{bmatrix} M^{-1} & M^{-1}N & 0 \\ 0 & -I & 0 \\ N^\top M^{-1} & N^\top M^{-1} N & -I \end{bmatrix}$$

where $M \in \mathbb{R}^{(n+d)\times(n+d)}$ and $N \in \mathbb{R}^{(n+d)\times n}$ are defined in Definition 2.6.3 and 2.6.6.

One important feature of the $L \in \mathbb{R}^{(3n+d)\times(3n+d)}$ matrix is multiplying its inverse with a proper vector gives the desired matrix vector product of interest.

Lemma 2.6.8 (Restatement of Lemma 2.6.2)**.** *Let L be defined as in Definition 2.6.5. Then we have*

$$L^{-1} \begin{bmatrix} \mathbf{0}_{n+d} \\ h \\ h \end{bmatrix} = \begin{bmatrix} \star \\ \star \\ \sqrt{W}A^\top (AWA^\top)^{-1} A\sqrt{W}h \end{bmatrix}$$

Lemma 2.6.9 (Sketch on the left)**.** *Let $R \in \mathbb{R}^{n\times(3n+d)}$, let $L \in \mathbb{R}^{(3n+d)\times(3n+d)}$ be the matrix defined in Definition 2.6.5, consider the matrix*

$$\begin{bmatrix} L & 0 \\ R & -I \end{bmatrix},$$

then we have

$$\left(\begin{bmatrix} L & 0 \\ R & -I \end{bmatrix} \right)^{-1} = \begin{bmatrix} L^{-1} & 0 \\ RL^{-1} & -I \end{bmatrix}$$

Proof. We have

$$\begin{aligned} \left(\begin{bmatrix} L & 0 \\ R & -I \end{bmatrix} \right)^{-1} &= \begin{bmatrix} L^{-1} & 0 \\ -(-I)^{-1}RL^{-1} & (-I)^{-1} \end{bmatrix} \\ &= \begin{bmatrix} L^{-1} & 0 \\ RL^{-1} & -I \end{bmatrix} \end{aligned}$$

where the first step comes from Fact 2.6.1 and the second step comes from simplying the terms. □

As we have shown, it is possible to multiply a conforming matrix R either on the left or on the right of $L^{-1} \in \mathbb{R}^{(3n+d)\times(3n+d)}$. We now show how to design proper sketching matrices. We start with the discussion on sketching on the left.

Theorem 2.6.10. *Let $\mathsf{R} \in \mathbb{R}^{n\times n}$ be a collection of sketching matrices, define $R \in \mathbb{R}^{n\times(3n+d)}$ to be the following matrix:*

$$R = \begin{bmatrix} \mathbf{0}_{n\times(2n+d)} & \mathsf{R} \end{bmatrix}$$

Then we have

$$\left(\begin{bmatrix} L & \mathbf{0}_{(3n+d)\times n} \\ R & -I_n \end{bmatrix} \right)^{-1} \begin{bmatrix} \mathbf{0}_{n+d} \\ h \\ h \\ \mathbf{0}_n \end{bmatrix} = \begin{bmatrix} \star \\ \mathsf{R}\sqrt{W}A^\top (AWA^\top)^{-1} A\sqrt{W}h \end{bmatrix}$$

Proof. By Lemma 2.6.9, we know

$$\left(\begin{bmatrix} L & \mathbf{0}_{(3n+d)\times n} \\ R & -I_n \end{bmatrix} \right)^{-1} = \begin{bmatrix} L^{-1} & \mathbf{0}_{(3n+d)\times n} \\ RL^{-1} & -I_n \end{bmatrix}.$$

Compute the matrix vector product gives us

$$\begin{bmatrix} L^{-1} & \mathbf{0}_{(3n+d)\times n} \\ RL^{-1} & -I_n \end{bmatrix} \begin{bmatrix} \mathbf{0}_{n+d} \\ h \\ h \\ \mathbf{0}_n \end{bmatrix} = \begin{bmatrix} \star \\ RL^{-1} \begin{bmatrix} \mathbf{0}_{n+d} \\ h \\ h \end{bmatrix} \end{bmatrix}$$

We have

$$\begin{aligned} RL^{-1} \begin{bmatrix} \mathbf{0}_{n+d} \\ h \\ h \end{bmatrix} &= R \begin{bmatrix} \star \\ \star \\ \sqrt{W}A^\top (AWA^\top)^{-1} A\sqrt{W}h \end{bmatrix} \\ &= \begin{bmatrix} \mathbf{0}_{n\times(2n+d)} & \mathsf{R} \end{bmatrix} \begin{bmatrix} \star \\ \star \\ \sqrt{W}A^\top (AWA^\top)^{-1} A\sqrt{W}h \end{bmatrix} \\ &= \mathsf{R}\sqrt{W}A^\top (AWA^\top)^{-1} A\sqrt{W}h \end{aligned}$$

where the first step follows that $L^{-1} \begin{bmatrix} \mathbf{0}_{n+d} \\ h \\ h \end{bmatrix} = \begin{bmatrix} \star \\ \star \\ \sqrt{W}A^\top (AWA^\top)^{-1} A\sqrt{W}h \end{bmatrix}$ by Lemma 2.6.2, the second step follows from the definition of R, and the final step comes from the matrix multiplication.

This completes the proof. □

Both [27] and [72] can be identified as sketching on the right. We show how to maintain their sketching as inverse.

Lemma 2.6.11 (Sketch on the right)**.** *Let $R \in \mathbb{R}^{(3n+d)\times n}$, let $L \in \mathbb{R}^{(3n+d)\times(3n+d)}$ be the matrix defined in Definition 2.6.5, consider the matrix*

$$\begin{bmatrix} L & R \\ 0 & -I \end{bmatrix},$$

then we have

$$\left(\begin{bmatrix} L & R \\ 0 & -I \end{bmatrix}\right)^{-1} = \begin{bmatrix} L^{-1} & L^{-1}R \\ 0 & -I \end{bmatrix}$$

Proof. We have

$$\begin{aligned} \left(\begin{bmatrix} L & R \\ 0 & -I \end{bmatrix}\right)^{-1} &= \begin{bmatrix} L^{-1} & -L^{-1}R(-I)^{-1} \\ 0 & (-I)^{-1} \end{bmatrix} \\ &= \begin{bmatrix} L^{-1} & L^{-1}R \\ 0 & -I \end{bmatrix} \end{aligned}$$

where the first step comes from Fact 2.6.1 and the second step comes from simplifying the terms. □

Theorem 2.6.12. *Let $\mathsf{R} = \begin{bmatrix} R_1^\top & R_2^\top & \cdots & R_T^\top \end{bmatrix} \in \mathbb{R}^{n\times n}$ be a collection of sketching matrices. Let $T = \sqrt{n}$. Define $\mathsf{B} \in \mathbb{R}^{(3n+d)\times n}$ to be the following matrix:*

$$\mathsf{B} = \begin{bmatrix} \mathbf{0}_{(n+d)\times n} \\ \mathsf{R}^\top \\ \mathsf{R}^\top \end{bmatrix}$$

Then we have

$$\left(\begin{bmatrix} L & \mathsf{B} \\ \mathbf{0}_{n\times(3n+d)} & -I_n \end{bmatrix}\right)^{-1} \begin{bmatrix} \mathbf{0}_{3n+d} \\ I_t \mathsf{R} h \end{bmatrix} = \begin{bmatrix} \star \\ \sqrt{W}A^\top (AWA^\top)^{-1} A\sqrt{W} R_t^\top R_t h \end{bmatrix}$$

where I_t is a diagonal matrix whose $\frac{n(t-1)}{T}$th to $\frac{nt}{T}$th diagonal entries are 1 and other diagonal entries are 0 such that $\mathsf{R}^\top I_t \mathsf{R} = R_t^\top R_t$.

Proof. By Lemma 2.6.11, we know

$$\left(\begin{bmatrix} L & \mathsf{B} \\ \mathbf{0}_{n\times(3n+d)} & -I_n \end{bmatrix}\right)^{-1} = \begin{bmatrix} L^{-1} & L^{-1}\mathsf{B} \\ \mathbf{0}_{n\times(3n+d)} & -I_n \end{bmatrix}.$$

Compute the matrix vector product gives us

$$\begin{bmatrix} L^{-1} & L^{-1}\mathsf{B} \\ \mathbf{0}_{n\times(3n+d)} & -I_n \end{bmatrix} \begin{bmatrix} \mathbf{0}_{3n+d} \\ I_t \mathsf{R} h \end{bmatrix} = \begin{bmatrix} \star \\ L^{-1} R I_t \mathsf{R} h \end{bmatrix}$$

We have

$$
\begin{aligned}
L^{-1}\mathsf{B}I_t\mathsf{R}h &= L^{-1}\begin{bmatrix}\mathbf{0}_{n+d}\\ \mathsf{R}^\top I_t\mathsf{R}h\\ \mathsf{R}^\top I_t\mathsf{R}h\end{bmatrix}\\
&= \begin{bmatrix}\star\\ \star\\ \sqrt{W}A^\top(AWA^\top)^{-1}A\sqrt{W}\mathsf{R}^\top I_t\mathsf{R}h\end{bmatrix}\\
&= \begin{bmatrix}\star\\ \star\\ \sqrt{W}A^\top(AWA^\top)^{-1}A\sqrt{W}R_t^\top R_t h\end{bmatrix}
\end{aligned}
$$

where the first step follows from the definition of B, the second step follows from Lemma 2.6.2, and the final step follows that $\mathsf{R}^\top I_t\mathsf{R} = R_t^\top R_t$.

This completes the proof. □

Remark 2.6.13. *We include these constructions here as a showcase of one attempt to unify the projection maintenance task. However, the inverse maintenance itself is not the key to realize the speedup in [27, 55, 72], as we will show in next section, it is the* coordinate-wise embedding property *that is key to these developments.*

For completeness, we include the version where h can be maintained inside the matrix and an algorithm to update, query and reset the data structure in Appendix B.

2.6.2 Coordinate-wise Embedding

In [72], they propose a unified framework for sketching or sampling called coordinate-wise embedding property:

Definition 2.6.14 ((α,β,δ)-coordinate wise embedding)**.** We say a randomized matrix $R \in \mathbb{R}^{b\times n}$ satisfying (α,β,δ)-coordinate wise embedding if

1. $\mathbb{E}_{R\sim\Pi}[g^\top R^\top Rh] = g^\top h$,
2. $\mathbb{E}_{R\sim\Pi}[(g^\top R^\top Rh)^2] \leq (g^\top h)^2 + \frac{\alpha}{b}\|g\|_2^2\|h\|_2^2$,
3. $\Pr_{R\sim\Pi}\left[|g^\top R^\top Rh - g^\top h| \geq \frac{\beta}{\sqrt{b}}\|g\|_2\|h\|_2\right] \leq \delta$.

Remark 2.6.15. *Given a randomized matrix $R \in \mathbb{R}^{b\times n}$ satisfying (α,β,δ)-coordinate wise embedding and any orthogonal projection $P \in \mathbb{R}^{n\times n}$, above definition implies*

1. $\mathbb{E}_{R\sim\Pi}[PR^\top Rh] = Ph$,
2. $\mathbb{E}_{R\sim\Pi}[(PR^\top Rh)_i^2] \leq (Ph)_i^2 + \frac{\alpha}{b}\|h\|_2^2$,
3. $\Pr_{R\sim\Pi}\left[|(PR^\top Rh)_i - (Ph)_i| \geq \frac{\beta}{\sqrt{b}}\|h\|_2\right] \leq \delta$.

since $\|P\|_2 \le 1$ *implies* $\|P_{i,:}\|_2 \le 1$ *for all* $i \in [n]$.

In [72], they use certain family of sketching matrices satisfying the coordinate-wise embedding to speed up the computation of the approximate matrix vector product Ph. Their approach is also oblivious, in the sense that sketching matrices are initialized before we have access to the vector h.

In [27], they use a diagonal sampling matrix $D \in \mathbb{R}^{n\times n}$ with roughly b non-zero entries on the diagonal. We design the matrix $R \in \mathbb{R}^{b\times n}$ as follows: let S be the set of indices of non-zeros in D, then we set $R_{i,i} = D_{i,i}$ for $i \in S$.

The matrix D is designed as follows: given $h \in \mathbb{R}^n$, we have

$$D_{i,i} = \begin{cases} \frac{1}{p_i} & \text{with probability } p_i := b \cdot \left(\frac{h_i^2}{\|h\|_2^2} + \frac{1}{n} \right) \\ 0 & \text{otherwise} \end{cases}$$

We prove the above diagonal sampling matrix satisfies the first two conditions of Definition 2.6.14.

Lemma 2.6.16 (Formal version of Theorem 1.1.20)**.** *Let* $D \in \mathbb{R}^{n\times n}$ *be the sampling matrix defined as above. For any* $g \in \mathbb{R}^n$*, we have*

- $\mathbb{E}_D[g^\top Dh] = g^\top h$.
- $\mathbb{E}_D[(g^\top Dh)^2] = (g^\top h)^2 + \frac{1}{b}\|g\|_2^2 \|h\|_2^2$.
- $\Pr_D[|g^\top Dh - g^\top h| \ge \frac{\log(1/\delta)}{\sqrt{b}} \|g\|_2 \|h\|_2] \le \delta$.

Proof. In expectation, we have

$$\begin{aligned} \mathbb{E}[D_{i,i}] &= p_i \cdot \frac{1}{p_i} + (1 - p_i) \cdot 0 \\ &= 1 \end{aligned}$$

hence, the matrix in expectation is identity.

For variance, we have

$$\begin{aligned} \mathbb{E}[(g^\top Dh)^2] &= \mathbb{E}[(\sum_{i=1}^n g_i D_{i,i} h_i)^2] \\ &= \mathbb{E}[\sum_{i=1}^n (g_i D_{i,i} h_i)^2 + \sum_{i\neq j} 2 g_i D_{i,i} h_i g_j D_{j,j} h_j] \\ &= \underbrace{\sum_{i=1}^n \mathbb{E}[(g_i D_{i,i} h_i)^2]}_{A} + 2 \underbrace{\sum_{i\neq j} \mathbb{E}[g_i D_{i,i} h_i g_j D_{j,j} h_j]}_{B} . \end{aligned}$$

We bound A (diagonal term) and B (off-diagonal term) separately.

For A, we have

$$A = \mathbb{E}[\sum_{i=1}^n g_i^2 D_{i,i}^2 h_i^2]$$

$$
\begin{aligned}
&= \sum_{i=1}^{n} g_i^2 \mathbb{E}[D_{i,i}^2] h_i^2 \\
&= \sum_{i=1}^{n} \frac{1}{p_i} g_i^2 h_i^2 \\
&= \frac{1}{b} \sum_{i=1}^{n} \frac{1}{\frac{h_i^2}{\|h\|_2^2} + \frac{1}{n}} g_i^2 h_i^2 \\
&= \frac{1}{b} \sum_{i=1}^{n} \frac{n\|h\|_2^2}{nh_i^2 + \|h\|_2^2} g_i^2 h_i^2 \\
&\leq \frac{1}{b} \sum_{i=1}^{n} \frac{n\|h\|_2^2}{nh_i^2} g_i^2 h_i^2 \\
&= \frac{1}{b} \|h\|_2^2 \|g\|_2^2
\end{aligned}
$$

For B, we have

$$
\begin{aligned}
B &= \mathbb{E}[\sum_{i \neq j} g_i D_{i,i} h_i g_j D_{j,j} h_j] \\
&= \sum_{i \neq j} g_i h_i g_j h_j \mathbb{E}[D_{i,i} D_{j,j}] \\
&= \sum_{i \neq j} g_i h_i g_j h_j \\
&= \sum_{i \in [n]} g_i h_i \sum_{j \in [n] \setminus \{i\}} g_j h_j \\
&= \sum_{i \in [n]} g_i h_i (\sum_{j \in [n]} g_j h_j - g_i h_i) \\
&= \sum_{i \in [n]} g_i h_i (g^\top h - g_i h_i) \\
&= (g^\top h)^2 - \sum_{i \in [n]} g_i^2 h_i^2 \\
&\leq (g^\top h)^2.
\end{aligned}
$$

Put it together, we have

$$
\mathbb{E}[(g^\top D h)^2] \leq (g^\top h)^2 + \frac{1}{b} \|g\|_2^2 \|h\|_2^2.
$$

For probability, let Y_i denote $g_i D_{i,i} h_i - g_i h_i$, note that $\mathbb{E}[Y_i] = 0$ and the variance of Y_i is

$$
\begin{aligned}
\mathbf{Var}[Y_i] &= \mathbb{E}[Y_i^2] \\
&= g_i^2 \mathbb{E}[(D_{i,i} - 1)^2] h_i^2
\end{aligned}
$$

$$
\begin{aligned}
&= g_i^2 h_i^2 (\mathbb{E}[D_{i,i}^2] - 2\mathbb{E}[D_{i,i}] + 1) \\
&= g_i^2 h_i^2 (\frac{1}{p_i} - 1) \\
&= g_i^2 h_i^2 \frac{1}{b} \frac{1}{\frac{h_i^2}{\|h\|_2^2} + \frac{1}{n}} \\
&= g_i^2 h_i^2 \frac{1}{b} \frac{n\|h\|_2^2}{n h_i^2 + \|h\|_2^2} \\
&\leq \frac{1}{b} g_i^2 \|h\|_2^2.
\end{aligned}
$$

We also need an absolute value bound:

$$
\begin{aligned}
|Y_i| &= |g_i (D_{i,i} - 1) h_i| \\
&\leq |g_i h_i|.
\end{aligned}
$$

By Bernstein inequality, we have

$$
\begin{aligned}
\Pr[|\sum_{i=1}^n Y_i| \geq \frac{\beta}{\sqrt{b}} \|g\|_2 \|h\|_2] &\leq \exp\left(- \frac{\frac{\beta^2}{b} \|g\|_2^2 \|h\|_2^2}{\frac{1}{b}\|g\|_2^2 \|h\|_2^2 + \max_{i \in [n]} \frac{\beta}{\sqrt{b}} |g_i h_i| \|g\|_2 \|h\|_2 / 3} \right) \\
&\leq \exp\left(- \frac{\frac{\beta^2}{b} \|g\|_2^2 \|h\|_2^2}{\frac{\beta}{b} \|g\|_2^2 \|h\|_2^2} \right) \\
&= \exp(-\beta),
\end{aligned}
$$

picking $\beta = \log(1/\delta)$, we obtain the desired result. □

Remark 2.6.17. *Our above argument shows that the sampling matrix used in [27] is a* $(1, \log(1/\delta), \delta)$*-coordinate-wise embedding.*

The above lemma ultimately unifies [27, 55, 72]. From the perspective of inserting the sketching/sampling matrix, [27] and [72] are very similar in the sense that both of them compute $PR^\top Rh$ with different R. For [55], its sketch is in the form

$$
R^\top RPh,
$$

in the language of coordinate-wise embedding, we use the sketching matrix to preserve the norm between I and Ph, however, this will make central path infeasible.

Chapter 3

Faster Sparsification via Faster Inner Product Data Structures

Given a matrix $V \in \mathbb{R}^{m \times d}$, the task of sparsifying the matrix by selecting a small number of re-scaled rows while preserving the spectral structure of the matrix is one of the most important primitives in graphs and numerical linear algebra tasks. The celebrated work by Batson, Spielman and Srivastava [12] shows that it's possible to obtain a sparsified matrix with only $\varepsilon^{-2}d$ rows. While nearly linear time algorithm has been obtained by Lee and Sun [54], it remains a major open problem whether it is possible to compute such spectral sparsifier deterministically. Computing spectral sparsifier deterministically has implications to various graph problems in the adaptive streaming model and derandomize some important numerical linear algebra problems. However, the fastest deterministic algorithm still takes time $\Omega(d^4)$ [90].

Our main contribution is a deterministic algorithm that runs in time $O(d^{\omega+1})$ for computing a spectral sparsifier. To achieve such performance, we model the first algorithm in [12] as a data structure task, and get a direct speedup from their iterative process.

Apart from linear-sized spectral sparsifier, we also obtain faster algorithms for two related problems: the vector packing problem posed by [84] and the experimental design problem by [7].

This chapter is based on the arXiv document: https://arxiv.org/pdf/2204.03209.pdf, coauthored by the thesis author.

3.1 Linear-Sized Spectral Sparsifier

In this section, we speed up the construction of deterministic spectral sparsifier. We show that solving such problem is equivalent of Task 1.1.4. We also provide an alternative implementation of the almost-linear time algorithm by Lee and Sun [53]. We show that their iterative process can be solved as Task 1.1.5.

3.1.1 Problem Setup

In this section, we setup the problem. Given a full rank matrix $V \in \mathbb{R}^{m \times d}$ with $m \geq d$, the goal is to pick only $s = \Theta(\varepsilon^{-2}d)$ rescaled rows of V to form $\widetilde{V} \in \mathbb{R}^{s \times d}$ such that $(1-\varepsilon)V^\top V \preceq$

$\widetilde{V}^\top \widetilde{V}(1+\varepsilon)V^\top V$. Since V is full rank, we can normalize $V^\top V$ and assume it's identity. The task can be defined as follows:

Definition 3.1.1. Suppose we are given m vectors $v_i, \ldots, v_m \in \mathbb{R}^d$ satisfying $\sum_{i=1}^m v_i v_i^\top = I$, we want to find scalars $\{s_i\}_{i=1}^m$ satisfying

$$|\{s_i : s_i \neq 0\}| = O(d/\varepsilon^2),$$

such that

$$(1-\varepsilon)\cdot I \preceq \sum_{i=1}^m s_i v_i v_i^\top \preceq (1+\varepsilon)\cdot I.$$

We define several notions that will be used extensively in the proof of BSS sparsifier.

Definition 3.1.2. Let $A \in \mathbb{R}^{d\times d}$ be a symmetric matrix with eigenvalues $\lambda_1, \ldots, \lambda_d$ and $u, \ell \in \mathbb{R}$, define:

$$\Phi^u(A) := \mathrm{tr}[(uI - A)^{-1}] = \sum_{i=1}^d \frac{1}{u-\lambda_i}$$
$$\Phi_\ell(A) := \mathrm{tr}[(A - \ell I)^{-1}] = \sum_{i=1}^d \frac{1}{\lambda_i - \ell}.$$

3.1.2 The BSS Algorithm

The BSS algorithm is as follows: the algorithm starts by maintaining two "barriers" of eigenvalues $u_0 = \frac{d}{\varepsilon}$ and $\ell_0 = -\frac{d}{\varepsilon}$. Iteratively, the algorithm searches for an index $i \in [m]$ such that the inner product between $v_i v_i^\top$ and a quantity related to lower barrier is large while the inner product related to barrier is small. Then we add this outer product $v_i v_i^\top$ with a scaling into the matrix A we are forming. After $\Theta(d/\varepsilon^2)$ iterations, we've constructed a matrix A with the property that $A \approx_\varepsilon I$.

We formalize the algorithm as follows:

Algorithm 13 BSS algorithm

procedure BSS($\{v_1, \ldots, v_m\} \in (\mathbb{R}^d)^m$)
  $u_0 \leftarrow \frac{d}{\varepsilon}, \ell_0 \leftarrow -\frac{d}{\varepsilon}$
  $A_0 \leftarrow \mathbf{0}_{d\times d}$
  $T \leftarrow \frac{d}{\varepsilon^2}$
  $\delta_U \leftarrow 1, \delta_L \leftarrow \frac{1}{1+2\varepsilon}$
  for $t = 1 \to T$ **do**
    $u_t \leftarrow u_{t-1} + \delta_U, \ell_t \leftarrow \ell_{t-1} + \delta_L$
    $L_t \leftarrow \frac{(A_{t-1}-\ell_t I)^{-2}}{\Phi_{\ell_t}(A_{t-1})-\Phi_{\ell_{t-1}}(A_{t-1})} - (A_{t-1} - \ell_t I)^{-1}$
    $U_t \leftarrow \frac{(u_t I - A_{t-1})^{-2}}{\Phi_{u_{t-1}}(A_{t-1})-\Phi_{u_t}(A_{t-1})} + (u_t I - A_{t-1})^{-1}$
    Find an index j such that

$$v_j^\top (L_t - U_t) v_j \geq 0$$

    $c \leftarrow \frac{v_j^\top (L_t+U_t)v_j}{2}$
    $A_t \leftarrow A_{t-1} + \frac{1}{c} \cdot v_j v_j^\top$
  end for
  return A_T/d
end procedure

The central lemma that guarantees the BSS algorithm to find a good sparsifier that satisfies both upper and lower bound is the following:

Lemma 3.1.3 (Lemma 3.5 of [12]). *Suppose $A \in \mathbb{R}^{d\times d}$ satisfying $\ell I \prec A \prec uI$, let $\varepsilon \in (0,1)$ and suppose $\Phi_u(A) \leq \varepsilon, \Phi_\ell(A) \leq \varepsilon$, and $\varepsilon, \delta_U, \delta_L$ satisfying*

$$0 \leq \frac{1}{\delta_U} + \varepsilon \leq \frac{1}{\delta_L} - \varepsilon,$$

then we have

1. **Lower bounding lower barrier.**

$$\sum_{i=1}^m v_i^\top \left(\frac{(A-(\ell+\delta_L)I)^{-2}}{\Phi_{\ell+\delta_L}(A)-\Phi_\ell(A)} - (A-(\ell+\delta_L)I)^{-1}\right) v_i \geq \frac{1}{\delta_L} - \varepsilon.$$

2. **Upper bounding upper barrier.**

$$\sum_{i=1}^m v_i^\top \left(\frac{((u+\delta_U)I-A)^{-2}}{\Phi^u(A)-\Phi^{u+\delta_U}(A)} + ((u+\delta_U)I-A)^{-1}\right) v_i \leq \frac{1}{\delta_U} + \varepsilon.$$

We also record two lemmas that control the growth of lower and upper barriers.

Lemma 3.1.4 (Lemma 3.3 of [12]). *Suppose $A \prec uI$ and $v \in \mathbb{R}^d$ is any vector. If*

$$c \geq v^\top \left(\frac{((u+\delta_U)I-A)^{-2}}{\Phi^u(A)-\Phi^{u+\delta_U}(A)} + ((u+\delta_U)I-A)^{-1}\right) v,$$

then

$$\Phi^{u+\delta_U}(A + \frac{1}{c} \cdot vv^\top) \leq \Phi^u(A) \text{ and } A + \frac{1}{c} \cdot vv^\top \prec (u+\delta_U)I.$$

Lemma 3.1.5 (Lemma 3.4 of [12])**.** *Suppose $A \succ \ell I$, $\Phi_\ell(A) \leq 1/\delta_L$ and $v \in \mathbb{R}^d$ is any vector. If*

$$0 < c \leq v^\top (\frac{(A-(\ell+\delta_L)I)^{-2}}{\Phi_{\ell+\delta_L}(A) - \Phi_\ell(A)} - (A-(\ell+\delta_L)I)^{-1})v,$$

then

$$\Phi_{\ell+\delta_L}(A + \frac{1}{c} \cdot vv^\top) \leq \Phi_\ell(A) \text{ and } A + \frac{1}{c} \cdot vv^\top \succ (\ell+\delta_L)I.$$

Combining the above 3 lemmas, we derive a lemma that justifies that in each iteration of Alg. 13, we can always find an index j satisfying the inequality on line 10. To simplify notation, we define $L_t := (\frac{(A-(\ell+\delta_L)I)^{-2}}{\Phi_{\ell+\delta_L}(A)-\Phi_\ell(A)} - (A-(\ell+\delta_L)I)^{-1})$ and $U_t := (\frac{((u+\delta_U)I-A)^{-2}}{\Phi^u(A)-\Phi^{u+\delta_U}(A)} + ((u+\delta_U)I - A)^{-1})$.

Lemma 3.1.6. *Suppose $A \in \mathbb{R}^{d\times d}$ satisfying $\ell I \prec A \prec uI$, let $\varepsilon \in (0,1)$ and suppose $\Phi^u(A) \leq \varepsilon, \Phi_\ell(A) \leq \varepsilon$, and $\varepsilon, \delta_U, \delta_L$ satisfying*

$$0 \leq \frac{1}{\delta_U} + \varepsilon \leq \frac{1}{\delta_L} - \varepsilon,$$

then there exists an index $j \in [m]$ and a positive value c such that

1. **Witness of gap between lower and upper barriers.**

$$v_j^\top L_t v_j \geq c \geq v_j^\top U_t v_j.$$

2. **Spectral property.**

$$(\ell+\delta_L)I \prec A + \frac{1}{c} \cdot v_j v_j^\top \prec (u+\delta_U)I.$$

Moreover, if we further have

$$0 \leq \frac{1}{\delta_U} + \varepsilon < \frac{1}{\delta_L} - \varepsilon,$$

then the witness of gap between lower and upper barriers has a strict inequality between the two quantities:

$$v_j^\top (\frac{(A-(\ell+\delta_L)I)^{-2}}{\Phi_{\ell+\delta_L}(A) - \Phi_\ell(A)} - (A-(\ell+\delta_L)I)^{-1})v_j > c$$
$$> v_j^\top (\frac{((u+\delta_U)I-A)^{-2}}{\Phi^u(A) - \Phi^{u+\delta_U}(A)} + ((u+\delta_U)I - A)^{-1})v_j.$$

Proof. By Lemma 3.1.3, we have the following:

$$\sum_{i=1}^{m} v_i^\top (\frac{(A-(\ell+\delta_L)I)^{-2}}{\Phi_{\ell+\delta_L}(A)-\Phi_\ell(A)} - (A-(\ell+\delta_L)I)^{-1})v_i \geq \frac{1}{\delta_L} - \varepsilon,$$
$$\sum_{i=1}^{m} v_i^\top (\frac{((u+\delta_U)I-A)^{-2}}{\Phi^u(A)-\Phi^{u+\delta_U}(A)} + ((u+\delta_U)I-A)^{-1})v_i \leq \frac{1}{\delta_U} + \varepsilon.$$

By an averaging argument, there must exist an index $j \in [m]$ that witnesses this inequality, i.e.,

$$v_j^\top (\frac{(A-(\ell+\delta_L)I)^{-2}}{\Phi_{\ell+\delta_L}(A)-\Phi_\ell(A)} - (A-(\ell+\delta_L)I)^{-1})v_j \geq v_j^\top (\frac{((u+\delta_U)I-A)^{-2}}{\Phi^u(A)-\Phi^{u+\delta_U}(A)} + ((u+\delta_U)I-A)^{-1})v_j.$$

The spectral property is guaranteed by Lemma 3.1.5 and Lemma 3.1.4.

For the strict inequality, note that if we have $\frac{1}{\delta_L} - \varepsilon > \frac{1}{\delta_U} + \varepsilon$, then by Lemma 3.1.3 and again by an averaging argument, we conclude that witness also exhibits a strict inequality. □

We also include a proof for the main Theorem of [12] here, since we will need to later adapt our data structure for this problem.

Lemma 3.1.7 (Theorem 3.1 of [12])**.** *Suppose we are given m vectors $v_1, \ldots, v_m \in \mathbb{R}^d$ satisfying $\sum_{i=1}^m v_i v_i^\top = I$, then there exists a deterministic algorithm (Alg. 13) can find scalars $\{s_i\}_{i=1}^m$ satisfying*

$$|\{s_i : s_i \neq 0\}| = O(d/\varepsilon^2),$$

such that

$$(1-\varepsilon)\cdot I \preceq \sum_{i=1}^{m} s_i v_i v_i^\top \preceq (1+\varepsilon) I.$$

The algorithm (Alg. 13) runs in time $O(md^3/\varepsilon^2)$.

Proof. We first prove the correctness. By the update rule of Alg. 13, we know that we maintain the following invariants across all iterations due to Lemma 3.1.5 and Lemma 3.1.4

$$\Phi_{u_t}(A_t) \leq \Phi_{u_{t-1}}(A_{t-1}) \text{ and } \Phi_{\ell_t}(A_t) \leq \Phi_{\ell_{t-1}}(A_{t-1}),$$

which means it suffices to examine $\Phi_{u_0}(A_0)$ and $\Phi_{\ell_0}(A_0)$ respectively, recall that we choose $u_0 = \frac{d}{\varepsilon}$, $\ell_0 = -\frac{d}{\varepsilon}$, hence we have

$$\Phi_{u_0}(A_0) = \sum_{i=1}^{d} \frac{\varepsilon}{d}$$
$$= \varepsilon,$$
$$\Phi_{\ell_0}(A_0) = \sum_{i=1}^{d} \frac{\varepsilon}{d}$$

$$= \varepsilon.$$

To conclude the proof, we shall apply Lemma 3.1.3 for $T = \Theta(d/\varepsilon^2)$ times, so we verify the relations between $\varepsilon, \delta_U, \delta_L$:

$$\frac{1}{\delta_U} + \varepsilon = 1 + \varepsilon \geq 0,$$
$$\frac{1}{\delta_L} - \varepsilon = 1 + \varepsilon \geq \frac{1}{\delta_U} + \varepsilon.$$

This means we can apply Lemma 3.1.3 and have

$$(\ell_0 + T\delta_L)I \prec A_T \prec (u_0 + T\delta_U)I,$$

plug in the values of $\ell_0, u_0, \delta_L, \delta_U$ and T, we conclude that

$$(1 - \varepsilon - 2\varepsilon^2)I \prec A_T/d \prec (1 + \varepsilon)I.$$

Now we analyze the running time, the algorithm iterates for $T = \Theta(d/\varepsilon^2)$ iterations, and for each iteration t, we compute L_t and U_t in $O(d^\omega)$ time, and the search for index j takes $O(md^2)$ time. Hence, the total running time is $O(md^3/\varepsilon^2)$. □

3.1.3 Faster Deterministic Sparsification via Nonnegative Inner Product Search

We observe that the core of the deterministic BSS algorithm is an inner product search step: given a query matrix $L_t - U_t$, we need to find a vector v_i with $\langle v_i v_i^\top, L_t - U_t \rangle$. To speed up this process, we make use of the positive search tree we developed in prior section. In short, we first preprocess all vectors in $\min\{O(\mathrm{nnz}(V^2)), O(md^{\omega-1})\}$ time, then at query time, we simply perform the positive search to find the desired vector, in time $\widetilde{O}(d^2)$ or $O(d^\omega)$.

We present our algorithm as follows:

Algorithm 14 Faster Sparsification with Nonnegative Inner Product Search Tree

procedure FASTERSPARSIFICATION($V = \{v_1, \ldots, v_m\} \in (\mathbb{R}^d)^m$) ▷ Theorem 3.1.8
  $u_0 \leftarrow \frac{d}{\varepsilon}, \ell_0 \leftarrow -\frac{d}{\varepsilon}$
  $A_0 \leftarrow \mathbf{0}_{d\times d}$
  $T \leftarrow \frac{d}{\varepsilon^2}$
  $\delta_U \leftarrow 1, \delta_L \leftarrow \frac{1}{1+3\varepsilon}$
  if $md^{\omega-1} \leq \mathrm{nnz}(V^2)$ **then**
    DS $\leftarrow$ VECTORPS VECPS
  else
    DS $\leftarrow$ MATRIXPS MATPS
  end if
  DS.INIT(V)
  for $t = 1 \to T$ **do**
    $u_t \leftarrow u_{t-1} + \delta_U, \ell_t \leftarrow \ell_{t-1} + \delta_L$
    $L_t \leftarrow \frac{(A_{t-1}-\ell_t I)^{-2}}{\Phi_{\ell_t}(A_{t-1})-\Phi_{\ell_{t-1}}(A_{t-1})} - (A_{t-1} - \ell_t I)^{-1}$
    $U_t \leftarrow \frac{(u_t I - A_{t-1})^{-2}}{\Phi^{u_{t-1}}(A_{t-1})-\Phi^{u_t}(A_{t-1})} + (u_t I - A_{t-1})^{-1}$
    $Q \leftarrow L_t - U_t$
    $v_j \leftarrow$ DS.QUERYPOSITIVESEARCH(Q)
    $c_t \leftarrow \frac{v_j^\top (L_t+U_t) v_j}{2}$
    $A_t \leftarrow A_{t-1} + \frac{1}{c_t} \cdot v_j v_j^\top$
  end for
  return A_T/d
end procedure

The correctness of the above algorithm follows obviously: by using the Positive Inner Product Search Tree, we are guaranteed to find a vector with positive inner product, which suffices to proceed the BSS algorithm. We summarize the running time in the following theorem.

Theorem 3.1.8 (Formal version of Theorem 1.2.1)**.** *Suppose we have $\sum_{i=1}^m v_i v_i^\top = I$, let A be the output of Algorithm 14, then*

$$(1-\varepsilon)\cdot I \preceq A \preceq (1+\varepsilon)\cdot I$$

and $A = \sum_{i=1}^m s_i v_i v_i^\top$ for $|\{s_i : s_i \neq 0\}| = \varepsilon^{-2} d$.

Moreover, Algorithm 14 has runtime

$$\min\{\mathrm{nnz}(V^2), md^{\omega-1}\} + \varepsilon^{-2} d^{\omega+1}.$$

Proof. The correctness follows naturally. To see the running time, note that by Theorem 2.2.1 and Theorem 2.2.4, the initialization takes $\min\{\mathrm{nnz}(V^2), md^{\omega-1}\}$ time. For each iteration, we need to invert $d \times d$ matrices, which takes $O(d^\omega)$ time, and we need to query the Positive IP Search Tree, which takes $\widetilde{O}(d^2)$ (Theorem 2.2.1) or $O(d^\omega)$ (Theorem 2.2.4) time. Thus, each iteration takes $O(d^\omega)$ time, and there are $O(\varepsilon^{-2} d)$ iterations in total. Hence, the total running time is

$$\min\{\mathrm{nnz}(V^2), md^{\omega-1}\} + \varepsilon^{-2} d^{\omega+1}.$$

□

3.1.4 Deterministic Sparsification via VECTORPS: Comparisons and Extensions

We compare our algorithm with known algorithms in the literature, both deterministic and randomized.

[90]: Deterministic. To improve the running time of [12], [90] adapts the following strategy: it uses a deterministic lossy construction to first compute a sparsifier of size $\varepsilon^{-2}d\log d$, then it runs the BSS algorithm on the matrix with $\varepsilon^{-2}d\log d$. The deterministic sparsifier it computes can be viewed as an analogy of the leverage score sampling [80] in deterministic setting. In order to construct the lossy sparsifier, it makes use of the hyperbolic cosine function as a potential to progress. At each iteration, it has to compute the hyperbolic cosine potential over all rows of V, incurring a $O(md)$ cost per iteration. Similar to the BSS algorithm, it only selects one vector at each iteration, hence its running time is $\widetilde{O}(\varepsilon^{-2}md^2)$ for constructing the lossy sparsifier. For the case of $m = d^2$, which is the standard case for a dense graph or a tall skinny matrix, their algorithm has $\Omega(d^4)$ runtime.

[3]: Randomized. The work by Allen-Zhu, Liao and Orecchia provides an alternative view of constructing the spectral sparsifier, it shows that spectral sparsification can be solved as a regret minimization problem over PSD matrices. While leverage score sampling [80] has inherent connection with matrix multiplicative weights update [10, 60], the linear-sized sparsifier requires a new view of the problem. Using regret minimization and the popular follow-the-regularized-leader (FTRL) approach, they show that by using a q norm regularizer, one can obtain a linear-sized sparsifier using mirror descent. This also introduces a novel potential function that leads later breakthroughs. From an algorithmic perspective, by using Johnson-Lindenstrauss to reduce the dimension then compute all necessary information at each iteration, they obtain an improved running time of $\widetilde{O}(\varepsilon^{-O(1)}(md^2 + d^{3+1/q}))$. Unfortunately, without the use of Johnson-Lindenstrauss, their algorithm is no faster than [12].

[53]: Randomized. Motivated by the q norm potential function in [3], Lee and Sun show how to further speed up their algorithm via randomized sampling. A key (bonus) result of the q norm potential function is that the iterative process might only run for $O(\varepsilon^{-2}qd^{1/q})$ iterations. [53] exploits this feature and ensures that their algorithm only runs for $O(\varepsilon^{-2}qd^{3/q})$ iterations, at each iteration, they batch sample many vectors and add them into the target matrix. By using fast matrix multiplication, they achieve a per iteration cost of $md^{\omega-1}$, coupled with $\varepsilon^{-2}qd^{3/q}$ iterations, their algorithm has a running time of $\widetilde{O}(\varepsilon^{-2}qmd^{\omega-1+3/q})$. Since their algorithm heavily relies on the sampling process, it is certain that their algorithm is randomized.

[54]: Randomized. Note that since [53], reducing the iteration count has been a main theme of speeding up the construction of linear-sized sparsifier. [54] achieves the optimality by only

requires $O(\varepsilon^{-2})$ iterations, with a novel potential function that provides more leeway in the analysis. This also means that roughly for each iteration, one needs to select $O(d)$ vectors into the sparsifier. Intuitively, [54] reduces the iteration count by setting up a much stronger objective per iteration. To solve such an objective, they invoke a positive SDP solver [4]. The correctness of their SDP solver builds upon its internal randomness, hence it is unclear how to derandomize their method and achieve a similar running time.

Bootstrapping via Sketching: Randomized. A randomized alternative of [90] is to approximate leverage score quickly then run any fast randomized linear-sized algorithm (say, [54]) on the bootstrapped sparsifier. To quickly approximate leverage scores, a popular approach is to use randomized sketching and adaptive sampling [16]. Similar to any randomized method we have discussed above, the speed comes from the use of randomness and efficient sketching matrix, which is inherent random. Also, such method typically does not care about the dependence on d, since it typically applies a sketching matrix then perform QR decomposition on the sketched matrix, incurring a $\mathrm{poly}(d)$ dependence on the running time.

Comparison with Our Method. We note that all the faster algorithms that break the $\Omega(d^4)$ barrier of [90] are randomized methods, they are either slow when derandomized, or inherently rely on the randomness to progress the algorithm. This poses a challenge when one wants to design dynamic spectral sparsifiers against adaptive adversary based on these primitives. In contrast to their deterministic counterpart, where the robustness against adaptive queries is guaranteed, it is nontrivial to modify a static randomized algorithm for adaptivity without slowdowns. Obtaining efficient deterministic spectral sparsifier has sophisticated implications for various dynamic graph and matrix problems.

In many senses, while previous results [53, 54] give almost and nearly linear time algorithms for spectral sparsification, they all need to read the input data entirely for each iteration. This is fine when the iteration count is small, and in the case of graph, such $\varepsilon^{-2}m$ dependence seems inevitable since one replaces the primitive matrix operations such as inversion with a Laplacian solve. When the target matrix V is a general matrix, it is clear that reading input for each iteration is sub-optimal. From this perspective, our data structure formulation gives the right direction to achieve the *truly* optimal running time for this problem, and various sparsification problem using the potential function of [12]. It also opens up the door to further study of efficient and deterministic spectral sparsifier.

Extensions to Sparsify PSD Matrices. We remark that our framework can be further extended to sparsify sum of PSD matrices, using the regret minimization approach introduced by Allen-Zhu, Liao and Orecchia [3]. Observe that their mirror descent algorithm can also be viewed as a variant of nonnegative inner product search.

3.1.5 Randomized BSS via Efficient Sampling

Note that the vanilla BSS algorithm has $\varepsilon^{-2}d$ iterations, and at each iteration, it only adds one vector and requires inverting $d \times d$ matrices. To alleviate this issue, [53] gives a randomized

sampling algorithm that only has $O(\varepsilon^{-2}qd^{3/q})$ iterations and sample multiple vectors at each iteration. To this end, they give an $\widetilde{O}(\varepsilon^{-2}qmd^{\omega-1+3/q})$ algorithm. We note that their algorithm reads the entire input at each iteration and involves using fast matrix multiplication, hence it cannot exploit the sparsity of the input unless for graph case. We show that by using the Weighted Sampling Tree, we 1). only need to read the input once, 2). exploit the sparsity of the input.

Algorithm 15 Randomized BSS with Weighted Sampling Tree

procedure RANDOMIZEDBSS($V = \{v_1, \ldots, v_m\} \in (\mathbb{R}^d)^m$) ▷ Theorem 3.1.9
  $j \leftarrow 0$
  $u_0 \leftarrow (2d)^{1/q}, \ell_0 \leftarrow -(2d)^{1/q}$
  $A_0 \leftarrow \mathbf{0}_{d\times d}$
  DS $\leftarrow$ MATRIXPS MATPS
  DS.INIT(V)
  while $u_j - \ell_j < 4 \cdot (2d)^{1/q}$ **do**
    $W_j \leftarrow \mathbf{0}_{d\times d}$
    $R_j \leftarrow (u_j I - A_j)^{-1} + (A_j - \ell_j I)^{-1}$
    $N_j \leftarrow \frac{1}{d^{2/q}}\mathrm{tr}[R_j] \cdot \min\{\lambda_{\min}(u_j I - A_j), \lambda_{\min}(A_j - \ell_j I)\}$
    $\Delta_{u,j} \leftarrow (1+2\varepsilon) \cdot \frac{\varepsilon \cdot N_j}{q \cdot \mathrm{tr}[R_j]}, \Delta_{\ell,j} \leftarrow (1-2\varepsilon)\frac{\varepsilon \cdot N_j}{q \cdot \mathrm{tr}[R_j]}$
    for $k = 1 \to N_j$ **do**
      $M_k \leftarrow$ DS.QUERYSAMPLE(R_j)
      $W_j \leftarrow \frac{\varepsilon}{q} \cdot \langle M_k, R_j \rangle \cdot M_k$
    end for
    $A_{j+1} \leftarrow A_j + W_j$
    $u_{j+1} \leftarrow u_j + \Delta_{u,j}, \ell_{j+1} \leftarrow \ell_j + \Delta_{\ell,j}$
    $j \leftarrow j + 1$
  end while
  return A_j
end procedure

In [53], they need to compute the quantity $v_i^\top R_j v_i$ for all $i \in [m]$ at each iteration, by leveraging fast matrix multiplication, this can be done in $O(md^{\omega-1})$ time. Computing matrix inverses and minimum eigenvalue takes $O(d^\omega)$ time, hence, their total running time per iteration is $O(md^{\omega-1} + d^\omega)$. In contrast, our algorithm only needs to perform matrix inversion and minimum eigenvalue computation in $O(d^\omega)$, and sample N_j vectors each in $\widetilde{O}(d^2)$ time per iteration.

Theorem 3.1.9. *Suppose we have $\sum_{i=1}^m v_i v_i^\top = I$, let A be the output of Algorithm 15, then*

$$(1-\varepsilon) \cdot I \preceq A \preceq (1+\varepsilon) \cdot I$$

and $A = \sum_{i=1}^m s_i v_i v_i^\top$ for $|\{s_i : s_i \neq 0\}| = O(\varepsilon^{-2}qd)$.

Moreover, Algorithm 15 has runtime

$$\widetilde{O}(\mathrm{nnz}(V^2) + \varepsilon^{-2}qd^{\omega+3/q} + \varepsilon^{-2}qd^3).$$

Proof. Again, since we only change the sampling process without modifying the distribution, the correctness follows from Theorem 1.2 of [53].

For the running time, initialization takes $\widetilde{O}(\mathrm{nnz}(V^2))$ time, each query takes $\widetilde{O}(d^2)$ time and there are $\varepsilon^{-2}qd$ queries in total, yields $\widetilde{O}(\varepsilon^{-2}qd^3)$. There are $O(\varepsilon^{-2}qd^{3/q})$ iterations in total, each iteration takes $O(d^\omega)$ time, so the total cost overall all iterations and all queries is

$$\widetilde{O}(\varepsilon^{-2}qd^{\omega+3/q} + \varepsilon^{-2}qd^3).$$

This gives a total running time of

$$\widetilde{O}(\mathrm{nnz}(V^2) + \varepsilon^{-2}qd^{\omega+3/q} + \varepsilon^{-2}qd^3).$$

□

Remark 3.1.10. *Compared to the algorithm of [53], our algorithm has several advantages. To achieve the per iteration cost of $O(md^{\omega-1})$, [53] assumes the input is given in the format $\{v_1, \ldots, v_m\}$, to get the running time, they need to form a matrix $V \in \mathbb{R}^{m\times d}$ then multiply a $d \times d$ matrix, which takes $O(md^{\omega-1})$, then they can compute m values using extra $O(md)$ time. Suppose in the more general case, when one is only given rank 1 matrices M_i, then it is no longer feasible to do the fast matrix multiplication trick since the input size is md^2 (assume inputs are dense matrices). In such scenario, the running time of [53] becomes $\widetilde{O}(\varepsilon^{-2}qd^{3/q}(md^2 + d^\omega))$ or $\widetilde{O}(\varepsilon^{-2}qd^{3/q}(\mathrm{nnz}(V^2) + d^\omega))$ in the sparse case.*

3.2 Vector Packing, or One-Sided Kadison-Singer Problem

In this section, we improve the running time of the vector packing, or one-sided Kadison-Singer problem [84]. We observe that their iterative process essentially Task 1.1.6.

3.2.1 Problem Setup

We consider a simpler and one-sided version of the well-known Kadison-Singer problem studied by Weaver [84], which is similar to the restricted invertibility problem [81].

Question 3.2.1. *Does there exist a constant $N \in \mathbb{N}$, such that if $\{v_1, \ldots, v_m\} \in (\mathbb{R}^d)^m$ satisfying $\|v_i\|_2 = \frac{1}{\sqrt{N}}, \forall i \in [m]$, and*

$$\sum_{i=1}^m v_i v_i^\top = I,$$

then there exists a subset $S \subseteq \{1, \ldots, m\}$ such that for any $q \in (0, 1)$, we have

$$\|\sum_{i\in S} v_i v_i^\top\| \leq q - \frac{1}{\sqrt{N}}.$$

In Weaver's discrepancy theory II, 2013 [84], he presented a polynomial algorithm that has the following guarantee:

$$\|\sum_{i\in S} v_i v_i^\top\| \leq \frac{n}{m} + O(\frac{1}{\sqrt{N}}).$$

Here $n := |S|$. We will dedicate our efforts to design a faster algorithmic framework to achieve an approximate guarantee as Weaver's result.

3.2.2 Tools

We introduce some useful facts and tools that will be used later.

Fact 3.2.2. *For any PSD matrix $Z \in \mathbb{R}^{d\times d}$, we have* $\mathrm{tr}[Z^{1/2}] \leq \sqrt{d \cdot \mathrm{tr}[Z]}$.

Proof. Note that for any $d \times d$ positive semi-definite matrix $Z \succeq 0$, $\mathrm{tr}[Z^{1/2}] \leq \sqrt{d \cdot \mathrm{tr}[Z]}$ due to Cauchy-Schwartz inequality applied to the non-negative spectrum of $Z^{1/2}$. □

Fact 3.2.3 (Matrix Woodbury Identity, [86, 87])**.** *For matrices $M \in \mathbb{R}^{n\times n}$, $U \in \mathbb{R}^{n\times d}$, $C \in \mathbb{R}^{d\times d}$, $V \in \mathbb{R}^{d\times n}$,*

$$(M + UCV)^{-1} = M^{-1} - M^{-1}U(C^{-1} + VM^{-1}U)^{-1}VM^{-1}.$$

Fact 3.2.4. *Let A and B denote two diagonal matrices in $\mathbb{R}^{d\times d}$. Suppose $\forall i \neq j \in [n]$, we have $\beta_i - \alpha_i = \beta_j - \alpha_j$, and let $\gamma = \beta_i - \alpha_i$. We have*

$$\mathrm{tr}[A^{-1} - B^{-1}] = \gamma \cdot \mathrm{tr}[A^{-1}B^{-1}].$$

Proof. We have

$$\begin{aligned}
\mathrm{tr}[A^{-1} - B^{-1}] &= \sum_{i=1}^{k} \frac{1}{\alpha_i} - \frac{1}{\beta_i} \\
&= \sum_{i=1}^{k} \frac{\beta_i - \alpha_i}{\alpha_i\beta_i} \\
&= \gamma \sum_{i=1}^{k} \frac{1}{\alpha_i\beta_i} \\
&= \gamma \cdot \mathrm{tr}[A^{-1}B^{-1}]
\end{aligned}$$

Thus, we complete the proof. □

Fact 3.2.5 (Inequality for two monotone sequences)**.** *Suppose $a_1 \geq a_2 \geq \cdots \geq a_n \geq 0$, $b_1 \geq \cdots \geq b_n \geq 0$, then we have*

$$\sum_{i=1}^{n} a_i b_{n-i} \leq \frac{1}{n} \sum_{i=1}^{n} a_i \sum_{j=1}^{n} b_j$$

3.2.3 Approximate Greedy Lemma

In this section, we describe and analyze a high level greedy process to construct the set S with the guarantee given in [84]. We generalize his analysis by introducing an approximation factor β, which is particularly valuable when later, we want to use certain approximate data structure to implement the high-level idea. We start with the main lemma of this section.

Lemma 3.2.6 (Approximate greedy lemma)**.** *Let $N \in \mathbb{R}_+$, if $\{v_1, \ldots, v_m\}$ is a finite sequence of vectors in $\mathbb{R}^d$ satisfying $\|v_i\|_2 = \frac{1}{\sqrt{N}}, \forall i \in [m]$ and*

$$\sum_{i=1}^{m} v_i v_i^\top = I.$$

Then for any $n < m$ and any unit vector u, we can find a set S ($|S| = n$) such that

$$\|\sum_{i \in S} v_i v_i^\top\| \leq \beta \cdot (\frac{n}{m} + O(\frac{1}{\sqrt{N}})).$$

where $\beta \geq 1$.

Proof. Before proceeding to main body of the proof, we observe that for the choice of N, we know $\mathrm{tr}[v_i v_i^\top] = \|v_i\|_2^2 = \frac{1}{N}$ and $\sum_{i=1}^m v_i v_i^\top = I$, thus we have $m = dN$.

We define the following sequence of numbers

$$a_i = \frac{1}{\sqrt{N}} + (1 + \frac{1}{\sqrt{N} - 1})\frac{i}{m}, \forall i \in \{0, 1, \cdots, n\}.$$

Let S_j to be the set we have at round t, we also define the matrix T_j as

$$T_j := \frac{1}{\beta} \cdot \sum_{i \in S_j} v_i v_i^\top$$

We are going to find a set of indices $i_1, \ldots, i_n$ such that the following two things hold

- $\|T_j\| < a_j$,
- $\Phi^{a_0}(T_0) \geq \ldots \geq \Phi^{a_n}(T_n)$,

where Φ^a is the upper barrier potential as in Def. 3.1.2.

Assume the above two conditions hold, then we will have

$$\begin{aligned}
\left\|\sum_{i \in S_n} v_i v_i^\top\right\| &= \beta \cdot \|T_n\| \\
&< \beta \cdot a_n \\
&= \beta \cdot (\frac{1}{\sqrt{N}} + (1 + \frac{1}{\sqrt{N} - 1})\frac{n}{m}) \\
&\leq \beta \cdot (\frac{n}{m} + O(\frac{1}{\sqrt{N}})).
\end{aligned}$$

Therefore, it suffices to show how to construct S_j that satisfies above two conditions. We will prove via induction.

Base case. For base case, consider $j = 0$, note $a_0 = \frac{1}{\sqrt{N}} > 0$ and $T_0 = 0$, so $\|T_0\| < a_0$. For potential, we compute $\Phi^{a_0}(T_0)$:

$$\Phi^{a_0}(T_0) = \mathrm{tr}[(\frac{1}{\sqrt{N}}I)^{-1}] = d\sqrt{N}.$$

Inductive hypothesis. For inductive hypothesis, we suppose for some $j < n$, we have $\|T_j\| < a_j$ and $\Phi^{a_0}(T_0) \geq \ldots \geq \Phi^{a_j}(T_j)$.

Inductive step. We prove for $j+1$. Suppose $v_1, \ldots v_j$ have been chose and we use $\lambda_1 \leq \cdots \leq \lambda_d$ be the eigenvalue of T_j. Then the eigenvalues of $I - T_j$ are $1 - \lambda_1 \geq \cdots \geq 1 - \lambda_d$ and the eigenvalues of $(a_{j+1}I - T_j)^{-1}$ are $\frac{1}{a_{j+1}-\lambda_1} \leq \cdots \leq \frac{1}{a_{j+1}-\lambda_d}$. Note that T_j is a complex symmetric matrix, we can express it using its eigen-decomposition: $T_j = Q_j^{-1} D_j Q_j$, where $D_j \in \mathbb{C}^{d\times d}$ is a diagonal matrix, whose i-th entry is λ_i.

Then we have

$$\begin{aligned}
\mathrm{tr}[(a_{j+1}I - T_j)^{-1}(I - \beta T_j)] &= \mathrm{tr}[Q_j^{-1}(a_{j+1}I - D_j)^{-1}(I - \beta D_j)Q_j] \\
&= \mathrm{tr}[(a_{j+1}I - D_j)^{-1}(I - \beta D_j)] \\
&= \sum_{l=1}^{d} \frac{1}{a_{j+1} - \lambda_l}(1 - \beta\lambda_l) \\
&\leq \frac{1}{d}\sum_{l=1}^{d} \frac{1}{a_{j+1} - \lambda_l} \sum_{l=1}^{d}(1 - \beta\lambda_l) \\
&= \frac{1}{d} \cdot \mathrm{tr}[(a_{j+1}I - T_j)^{-1}] \cdot \mathrm{tr}[I - \beta T_j] \\
&\leq \frac{1}{d} \cdot \mathrm{tr}[(a_j I - T_j)^{-1}] \cdot \mathrm{tr}[I - \beta T_j] \\
&= \frac{1}{d}\Phi^{a_j}(T_j) \cdot \mathrm{tr}[I - \beta T_j] \\
&\leq \frac{1}{d}\Phi^{a_0}(T_0) \cdot \mathrm{tr}[I - \beta T_j] \\
&= \sqrt{N} \cdot \mathrm{tr}[I - \beta T_j], \qquad (3.1)
\end{aligned}$$

where the fourth step follows from sorting inequality 3.2.5, the sixth step follows from $a_{j+1} > a_j$, the eighth step follows from the inductive hypothesis.

Consequently, we have

$$\begin{aligned}
\Phi^{a_j}(T_j) - \Phi^{a_{j+1}}(T_j) &= \mathrm{tr}[(a_j I - T_j)^{-1} - (a_{j+1}I - T_j)^{-1}] \\
&= \mathrm{tr}[Q_j^{-1}(a_j I - D_j)^{-1}Q_j - Q_j^{-1}(a_{j+1}I - D_j)^{-1}Q_j] \\
&= \mathrm{tr}[(a_j I - D_j)^{-1} - (a_{j+1}I - D_j)^{-1}] \\
&= (a_{j+1} - a_j)\mathrm{tr}[(a_j I - D_j)^{-1}(a_{j+1}I - D_j)^{-1}] \\
&= (1 + \frac{1}{\sqrt{N} - 1})\frac{1}{m} \cdot \mathrm{tr}[(a_j I - T_j)^{-1}(a_{j+1}I - T_j)^{-1}]
\end{aligned}$$

$$\geq (1+\frac{1}{\sqrt{N}-1})\frac{1}{m}\cdot \mathrm{tr}[(a_{j+1}I-T_j)^{-2}] \tag{3.2}$$

where the forth step follows from Fact 3.2.4, and the fifth step follows from $a_{j+1}-a_j=\frac{1}{m}(1+\frac{1}{\sqrt{N}-1})$. The last step follows from

$$\frac{1}{(a_{j+1}-\lambda_l)^2}\leq \frac{1}{(a_j-\lambda_l)(a_{j+1}-\lambda_l)}.$$

Furthermore, we have

$$\begin{aligned}\mathrm{tr}[(a_{j+1}I-T_j)^{-2}(I-\beta T_j)] &= \sum_{l=1}^{d}\frac{1}{(a_{j+1}-\lambda_l)^2}(1-\beta\lambda_l)\\ &\leq \frac{1}{d}\sum_{l=1}^{d}\frac{1}{(a_{j+1}-\lambda_l)^2}\sum_{l=1}^{d}(1-\beta\lambda_l)\\ &= \frac{1}{d}\mathrm{tr}[(a_{j+1}I-T_j)^{-2}]\cdot\mathrm{tr}[I-\beta T_j]. \end{aligned}\tag{3.3}$$

Combining Eq. (3.2) and (3.3), we get

$$\begin{aligned}\frac{\mathrm{tr}[(a_{j+1}I-T_j)^{-2}(I-\beta T_j)]}{\Phi^{a_j}(T_j)-\Phi^{a_{j+1}}(T_j)} &\leq \frac{m}{d}\frac{\sqrt{N}-1}{\sqrt{N}}\cdot\mathrm{tr}[I-\beta T_j]\\ &= N(1-\frac{1}{\sqrt{N}})\cdot\mathrm{tr}[I-\beta T_j], \end{aligned}\tag{3.4}$$

where the last step follows from $m/d=N$.

Denote $\overline{S}=[m]\setminus S$, then we have

$$\begin{aligned}&\sum_{i\in\overline{S}}\left(\frac{v_i^\top(a_{j+1}I-T_j)^{-2}v_i}{\Phi^a(T_j)-\Phi^{a_{j+1}}(T_j)}+v_i^\top(a_{j+1}I-T_j)^{-1}v_i\right)\\ &=\frac{\mathrm{tr}[(a_{j+1}I-T_j)^{-2}(I-\beta T_j)]}{\Phi^{a_j}(T_j)-\Phi^{a_{j+1}}(T_j)}+\mathrm{tr}[(a_{j+1}I-T_j)^{-1}(I-\beta T_j)]\\ &\leq N(1-\frac{1}{\sqrt{N}})\cdot\mathrm{tr}[I-\beta T_j]+\sqrt{N}\cdot\mathrm{tr}[I-\beta T_j]\\ &=N\cdot\mathrm{tr}[I-\beta T_j]\\ &=m-j.\end{aligned}$$

The first step follows from $\sum_{i\in S'}v_iv_i^\top=I-\beta T_j\in\mathbb{C}^{d\times d}$, the second step follows from Eq. (3.1) and (3.4). The last step follows from $\mathrm{tr}[\beta T_j]=j/N$ and $m=Nd$. Thus we conclude there exists an element of $\overline{S}$ satisfying

$$\frac{v_{i^\star}^\top(a_{j+1}I-T_j)^{-2}v_{i^\star}}{\Phi^{a_j}(T_j)-\Phi^{a_{j+1}}(T_j)}+v_{i^\star}^\top(a_{j+1}I-T_j)v_{i^\star}\leq 1\leq\beta.$$

Thus choosing $S_{j+1}=S_j\cup\{i^\star\}\subseteq[m]$, and using Lemma 3.1.4, we conclude $\|T_{j+1}\|<a_{j+1}$ and $\Phi^{a_{j+1}}(T_{j+1})\leq\Phi^{a_j}(T_j)$. □

Remark 3.2.7. *If we choose $\beta = 1$, then the above theorem reduces to the original version proved by Weaver [84], which corresponds to the exact algorithms. In our generalized version, we show that if we scale down each copy of $v_i v_i^\top$ by a factor of β, then the final bound is just worse by a factor of β, compared to the bound obtained by Weaver. This means that at each step of algorithm, we can tolerate for a vector with only approximately small inner products, as long as we know the approximation ratio, we can scale matrix T down and pay back the factor at the final bound. This inspires the use of data structure that outputs approximate solution.*

As another side note, the proof provides an algorithm that runs in n iterations and picks one vector at each iteration. This means the algorithm can either have a few iterations, or a large amount of iterations. Depending on n, we provide different algorithms.

3.2.4 An $O(n(md^2 + d^\omega))$ Implementation

Algorithm 16 Vanilla greedy algorithm derived from [84], it takes $nm \cdot \mathcal{T}_{\mathrm{mat}}(d, d, d)$ time.

procedure VANILLAGREEDY($\{v_1, \ldots, v_m\}, N, n$) ▷ Theorem 3.2.8
  $T_0 \leftarrow \mathbf{0}_{d \times d}$
  $S \leftarrow \emptyset$
  for $j = 0 \to n$ **do**
    $a_j = \frac{1}{\sqrt{N}} + (1 + \frac{1}{\sqrt{N}-1})\frac{j}{m}$
  end for
  for $j = 0 \to n$ **do**
    for $i \in [m] \setminus S$ **do**
      $c_i \leftarrow (\Phi^{a_{j-1}}(T_j) - \Phi^{a_j}(T_j))^{-1} \cdot v_i^\top (a_j I - T_j)^{-2} v_i + v_i^\top (a_j I - T_j)^{-1} v_i$ ▷ $\mathcal{T}_{\mathrm{mat}}(d, d, d)$ time
    end for
    $i^* = \arg\min_{i \in [m] \setminus S} c_i$
    $T_{j+1} \leftarrow T_j + v_{i^*} v_{i^*}^\top$
    $S \leftarrow S \cup \{i^*\}$
  end for
  return S
end procedure

Note that Algorithm 16 is a straightforward implementation of the process derived from the proof of Lemma 3.2.6.

Theorem 3.2.8. *Let $N \in \mathbb{N}_+$, if $\{v_1, \ldots, v_m\}$ is a finite sequence of vectors in $\mathbb{R}^d$ satisfying $\|v_i\|_2 = \frac{1}{\sqrt{N}}, \forall i \in [m]$ and $\sum_{i=1}^m v_i v_i^\top = I$. Then for any $n < m$, there exists a deterministic algorithm that takes time $O(n(md^2 + d^\omega))$ to find a set S with cardinality n such that*

$$\| \sum_{i \in S} v_i v_i^\top \| \leq \frac{n}{m} + O(\frac{1}{\sqrt{N}}),$$

Proof. The correctness proof is straightforward, since Algorithm 16 implements the greedy process exactly. To analyze the runtime, note the expensive step is to compute quantity c_i at each

iteration, where it involves inverting a $d \times d$ matrix, which takes $O(d^\omega)$ time, and compute the quantity in the form of $v^\top A^{-1} v$, which takes $O(d^2)$ time. Note that at each round, we need to compute c_i for at most m vectors, and there are n rounds. Thus, the total running time is

$$O(n(md^2 + d^\omega)).$$

□

3.2.5 Small Iterations via AIPE Data Structure

We note that the number of iterations in Algorithm 16 is determined by the number of vectors in the set S, hence, we provide different algorithms for different choices of n. In this section, we specifically consider the setting where $n \ll m$. In this case, we use the AIPE data structure with fast preprocessing time but need to linear scan over all vectors at each iteration. This is fine in our setting, since n is small.

Algorithm 17 AIPE-based Implementation

procedure AIPE-BASED($d \in \mathbb{N}$, $m \in \mathbb{N}$, $n \in \mathbb{N}$, $V \subset \mathbb{R}^d$, $\varepsilon \in (0,1)$,$\tau \in (0,1)$) ▷ Theorem 3.2.10
  $T_0 \leftarrow \mathbf{0}_{d \times d}$
  $S \leftarrow \emptyset$
  Construct V ▷ $V = [v_1, v_2, \cdots, v_m]$
  for $j = 0 \to n$ **do**
    $a_j = \frac{1}{\sqrt{N}} + (1 + \frac{1}{\sqrt{N}-1}) \frac{j}{m}$
  end for
  ADAPTIVEINNERPRODUCTESTIMATION AIPE
  AIPE.INIT(V,$1+\varepsilon$,δ)
  for $j = 0 \to n$ **do**
    $M_j \leftarrow (a_j I - T_j)^{-1}$ ▷ $M_j \in \mathbb{R}^{d \times d}$, it takes d^ω time
    $N_j \leftarrow (a_{j-1} I - T_j)^{-1}$ ▷ $N_j \in \mathbb{R}^{d \times d}$, it takes d^ω time
    $q \leftarrow \mathrm{vec}((\mathrm{tr}[N_j] - \mathrm{tr}[M_j])^{-1} M_j M_j + M_j))$
    $i^* \leftarrow$ AIPE.QUERYMIN(q)
    $T_{j+1} \leftarrow T_j + v_{i^*} v_{i^*}^\top$
    $S \leftarrow S \cup \{i^*\}$
    AIPE.DELETE(i^*)
  end for
  return S
end procedure

Theorem 3.2.9 (Small Iterations)**.** *Let $\tau, \varepsilon, \delta \in (0,1)$ and $N \in \mathbb{N}_+$, if $V = \{v_1, \ldots, v_m\}$ is a finite sequence of vectors in $\mathbb{R}^d$ satisfying $\|v_i\|_2 = \frac{1}{\sqrt{N}}, \forall i \in [m]$ and $\sum_{i=1}^m v_i v_i^\top = I$. Then for any $n < m$, there exists a randomized algorithm (Algorithm 17) that takes time $\mathcal{T}$ to find a set S ($|S| = n$) such that with probability at least $1 - \delta$,*

$$\| \sum_{i \in S} v_i v_i^\top \| \leq \frac{1}{c} \cdot (\frac{n}{m} + O(\frac{1}{\sqrt{N}})).$$

Further, if $c \in (\tau, \frac{1.01\tau}{0.01+\tau})$*, then the running time is* $\mathcal{T} = \widetilde{O}(md^2 + n \cdot (m + d^\omega))$.

Proof. We first recall that the AIPE data structure provides a $(1+\varepsilon, r)$-AFN data structure by Lemma 2.4.3, this means that as long as we have $(1+\varepsilon)^2 = \frac{c-c\tau}{c-\tau}$, then it gives the guarantee for (c, τ)-Min-IP. Note that by the range of c, as long as $\tau = O(1)$, we have $\varepsilon = O(1)$. From now on, we assume the AIPE data structure produces a (c, τ)-Min-IP data structure. This implies that

$$\langle v_{i^*} v_{i^*}^\top, \tau \cdot \frac{(a_{j+1} I - T)^{-2}}{\Phi^{a_j}(T) - \Phi^{a_{j+1}}(T)} + (a_{j+1} I - T)^{-1} \rangle \leq \frac{\tau}{c}$$
$$\Rightarrow \langle v_{i^*} v_{i^*}^\top, \frac{(a_{j+1} I - T)^{-2}}{\Phi^{a_j}(T) - \Phi^{a_{j+1}}(T)} + (a_{j+1} I - T)^{-1} \rangle \leq \frac{1}{c}.$$

i.e., we obtain an index with $\frac{1}{c}$ approximation guarantee. As we showed in Theorem 3.2.6, if we proceed with adding c copies of $v_{i^*} v_{i^*}^\top$, we will end up with the following guarantee:

$$\left\| \sum_{i \in S} v_i v_i^\top \right\| \leq \frac{1}{c} \cdot (\frac{n}{m} + O(\frac{1}{\sqrt{N}})).$$

Regarding the running time the algorithm, we note that it is enough to pick $\varepsilon = O(1)$, therefore by Theorem 2.4.6, the initialization takes $\widetilde{O}(md^2)$ time. At each iteration, we pay $O(d^\omega)$ to invert matrices, the QUERYMIN procedure takes $\widetilde{O}(m + d^2)$ time and the DELETE procedure takes $\widetilde{O}(d^2)$ time per Theorem 2.4.6. □

3.2.6 Large Iterations via AFN Data Structure

When number of iterations n becomes large, the linear scan at each round becomes expensive, e.g., if $n = O(m^{0.5})$, then the overall iteration cost becomes $\widetilde{O}(m^{1.5})$. To resolve this issue, we utilize the AFN-based data structure developed in Appendix A, which has a slightly worse initialization time but much improved per iteration cost.

Theorem 3.2.10 (Formal version of Theorem 1.2.2)**.** *Let* $\tau, c, \delta \in (0, 1)$ *and* $N \in \mathbb{N}_+$*, if* $V = \{v_1, \ldots, v_m\}$ *is a finite sequence of vectors in* $\mathbb{R}^d$ *satisfying* $\|v_i\|_2 = \frac{1}{\sqrt{N}}, \forall i \in [m]$ *and* $\sum_{i=1}^m v_i v_i^\top = I$*. Then for any* $n < m$*, there exists a randomized algorithm that takes time* $\mathcal{T}$ *to find a set* S $(|S| = n)$ *such that with probability at least* $1 - \delta$*,*

$$\| \sum_{i \in S} v_i v_i^\top \| \leq \frac{2}{c} \cdot (\frac{n}{m} + O(\frac{1}{\sqrt{N}})).$$

Further, we have

- *If* $c \in (\tau, \frac{8\tau}{7+\tau})$*, then* $\mathcal{T} = \widetilde{O}((m^{1.5} + \mathrm{nnz}(V))d^2 + n \cdot (\sqrt{m} d^2 + d^\omega))$*;*
- *If* $c \in (\tau, \frac{400\tau}{399+\tau})$*, then* $\mathcal{T} = \widetilde{O}((m^{1.01} + \mathrm{nnz}(V))d^2 + n \cdot (m^{0.01} d^2 + d^\omega))$.

Proof. Note that since we are using the approximate Min-IP data structure with parameter c and τ, we are promised to get an index i^* such that

$$\langle v_{i^*} v_{i^*}^\top, \tau \cdot \frac{(a_{j+1} I - T)^{-2}}{\Phi^{a_j}(T) - \Phi^{a_{j+1}}(T)} + (a_{j+1} I - T)^{-1} \rangle \leq \frac{\tau}{c}$$

Algorithm 18 AFN-based implementation.

procedure AFN-BASED($d \in \mathbb{N}$, $m \in \mathbb{N}$, $n \in \mathbb{N}$, $V \subset \mathbb{R}^d$, $c \in (0,1)$,$\tau \in (0,1)$) ▷ Theorem 3.2.10
  $T_0 \leftarrow \mathbf{0}_{d \times d}$
  $S \leftarrow \emptyset$
  Construct V ▷ $V = [v_1, v_2, \cdots, v_m]$
  for $j = 0 \to n$ **do**
    $a_j = \frac{1}{\sqrt{N}} + (1 + \frac{1}{\sqrt{N}-1})\frac{j}{m}$
  end for
  MINIP MI
  MI.INIT(d^2,m,V,c,τ) ▷ The dimension input to the Min-IP has been reduced by JLT
  for $j = 0 \to n$ **do**
    $M_j \leftarrow (a_j I - T_j)^{-1}$ ▷ $M_j \in \mathbb{R}^{d \times d}$, it takes d^ω time
    $N_j \leftarrow (a_{j-1} I - T_j)^{-1}$ ▷ $N_j \in \mathbb{R}^{d \times d}$, it takes d^ω time
    $q \leftarrow \mathrm{vec}((\mathrm{tr}[N_j] - \mathrm{tr}[M_j])^{-1} M_j M_j + M_j))$
    $i^* \leftarrow$ MI.QUERYMIN(q)
    $T_{j+1} \leftarrow T_j + v_{i^*} v_{i^*}^\top$
    $S \leftarrow S \cup \{i^*\}$
    MI.DELETE($\mathrm{vec}(v_{i^*} v_{i^*}^\top)$)
  end for
  return S
end procedure

$$\Rightarrow \langle v_{i^*} v_{i^*}^\top, \frac{(a_{j+1} I - T)^{-2}}{\Phi^{a_j}(T) - \Phi^{a_{j+1}}(T)} + (a_{j+1} I - T)^{-1} \rangle \leq \frac{1}{c}.$$

i.e., we obtain an index with $\frac{1}{c}$ approximation guarantee. As we showed in Theorem 3.2.6, if we proceed with adding c copies of $v_{i^*} v_{i^*}^\top$, we will end up with the following guarantee:

$$\left\| \sum_{i \in S} v_i v_i^\top \right\| \leq \frac{1}{c} \cdot (\frac{n}{m} + O(\frac{1}{\sqrt{N}})).$$

It remains to show we can have a data structure with such guarantee, we shall make use of Theorem 2.3.24 combined with the transformation illustrated in 2.3.1, we complete the proof of correctness of the data structure.

Now, we prove the correctness of the running time, which follows directly from Theorem 2.3.24. Note that it would incur an additive $\frac{\tau}{c}$ to the guarantee of inner product, which means the quality of approximation becomes $\frac{2}{c}$, with a success probability at least $1 - \delta$.

This completes the proof. □

3.3 Experimental Design via Regret Minimization

In [7], they show that by using a regret minimization framework, one can round up a fractional solution to the experimental design problem to integral. Their algorithm is exactly solving

Task 1.1.6 per iteration.

3.3.1 Definitions and Problem Setup

Definition 3.3.1. Let $\Delta_{d\times d}$ be the class of matrices defined as

$$\Delta_{d\times d} := \{A \in \mathbb{R}^{d\times d} : A \succeq 0, \mathrm{tr}[A] = 1\}.$$

Definition 3.3.2. Let $\psi : \mathbb{R}^{d\times d} \to \mathbb{R}$ be defined as

$$\psi(A) = -2\mathrm{tr}[A^{1/2}],$$

where $A \in \mathbb{R}^{d\times d}$ is a positive semi-definite matrix.

Definition 3.3.3. We define the Bregman divergence function associated with ψ, $\Delta_\psi : \mathbb{R}^{d\times d} \times \mathbb{R}^{d\times d} \to \mathbb{R}$ as

$$\Delta_\psi(A, B) = \psi(B) - \psi(A) - \langle \nabla\psi(A), B - A\rangle.$$

Definition 3.3.4. We define the mirror descent matrices $\widetilde{A}_t \in \mathbb{R}^{d\times d}$ and $A_t \in \mathbb{R}^{d\times d}$ as follows:

$$\begin{aligned}\widetilde{A}_t :=& \arg\min_{A\succeq 0}\{\Delta_\psi(A_{t-1}, A) + \alpha\langle F_{t-1}, A\rangle\},\\ A_t :=& \arg\min_{A\in\Delta_{d\times d}} \Delta_\psi(\widetilde{A}_t, A).\end{aligned}$$

Definition 3.3.5. We define a sequence of matrices $A_0, A_1, \ldots \in \mathbb{R}^{d\times d}$ as follows:

$$A_0 := (c_0 I + \alpha Z_0)^{-2},$$

where $c_0 \in \mathbb{R}$, $Z_0 \in \mathbb{R}^{d\times d}$ is symmetric and $A_0 \succ 0$. We also define A_t as

$$A_t := (c_t I + \alpha Z_0 + \alpha\sum_{l=0}^{t-1} F_l)^{-2},$$

where $c_t \in \mathbb{R}$ is the unique constant such that $A_t \succ 0$ and $\mathrm{tr}[A_t] = 1$.

Note we give two alternative definitions of matrix A_t, as shown in Claim 3.3.8, these two definitions are equivalent.

Finally, we formally define the rounding up problem for experimental design.

Question 3.3.6. *Let $\pi \in [0,1]^m$ with $\|\pi\|_1 \le n$ and $\sum_{i=1}^m \pi_i x_i x_i^\top = I_d$. Let $\gamma \ge 3$ and $\varepsilon \in (0, \frac{1}{\gamma}]$. Does there exist a subset $S \subset [m]$ with $|S| \le n$ such that*

$$\lambda_{\min}\big(\sum_{i\in S} x_i x_i^\top\big) \ge 1 - \gamma\cdot\varepsilon?$$

3.3.2 Useful Facts from Previous Work

In this section, we list the facts and tools that will be useful for our proof. For the complete proofs of these facts, we refer readers to [7].

Claim 3.3.7 (Lemma 2.7 in [7])**.** *Let $\Delta_{d\times d}$ be defined as Definition 3.3.1. Suppose $A_0 = (c_0 I + \alpha Z_0)^{-2} \in \mathbb{R}^{d\times d}$, where $c_o I + \alpha Z_0 \in \mathbb{R}^{d\times d}$ is positive definite, then for any $U \in \Delta_{d\times d}$,*

$$\Delta_\psi(A_0, U) \leq 2\sqrt{d} + \alpha\langle Z_0, U\rangle.$$

Claim 3.3.8 (Claim 2.9 in [7])**.** *Let $\widetilde{A}_t, A_t \in \mathbb{R}^{d\times d}$ be the matrices defined in Def. 3.3.4, if*

$$\alpha v_t^\top A_t^{1/2} v_t < 1,$$

then we have

$$\widetilde{A}_t = (A_{t-1}^{-1/2} + \alpha F_{t-1})^{-2}.$$

Claim 3.3.9 (Claim 2.10 in [7])**.** *Let $\Delta_{d\times d}$ be defined as in Definition 3.3.1. Suppose $P_t^\top A_t^{1/2} P_t = [b\ \ d; d\ \ c] \in \mathbb{R}^{2\times 2}$, $J = \mathrm{diag}(1, -1)$, and $2\alpha v_t^\top A_t^{1/2} v_t < 1$ for $v_t \in \mathbb{R}^d$ and $A_t \in \Delta_{d\times d}$. Then*

$$\left(J + P_t^\top A_t^{1/2} P_t\right)^{-1} = \left(J + \begin{bmatrix} b & d \\ d & c \end{bmatrix}\right)^{-1} \succeq \left(J + \begin{bmatrix} 2b & 0 \\ 0 & 2c \end{bmatrix}\right)^{-1}.$$

Claim 3.3.10 (Claim 2.11 of [7])**.** *Suppose $Z \succeq 0$ is a $d \times d$ PSD matrix with $\lambda_{\min}(Z) \leq 1$. Let $\alpha > 0$ be a parameter and $A = (\alpha Z + cI)^{-2} \in \mathbb{R}^{d\times d}$, where $c \in \mathbb{R}$ is the unique real number such that $A \succeq 0$ and $\mathrm{tr}[A] = 1$. Then*

- $\alpha\langle A^{1/2}, Z\rangle \leq d + \alpha\sqrt{d}$,
- $\langle A, Z\rangle \leq \sqrt{d}/\alpha + \lambda_{\min}(Z)$.

3.3.3 Algorithm

Algorithm 19 Swapping algorithm with Min-IP data structure

procedure SWAP($X \in \mathbb{R}^{m \times d}, n \in \mathbb{N}_+, \pi \in [0,1]^m, \varepsilon \in (0, 1/\gamma], c \in (0,1), \tau \in (0,1)$) ▷ Theorem 3.3.16
  $\alpha \leftarrow \sqrt{d}\beta/\varepsilon$ and $T \leftarrow n/(c\varepsilon)$
  $X \leftarrow X(X^\top \mathrm{diag}(\pi) X)^{-1/2}$ ▷ Whitening
  $S_0 \subseteq [m]$ be an arbitrary subset of support n
  $t \leftarrow 1$
  if $mT < \mathrm{nnz}(V) d^2$ **then** ▷ Small iterations
    DS $\leftarrow$ ADAPTIVEINNERPRODUCTESTIMATION DS
  else ▷ Large iterations
    DS $\leftarrow$ MINIP DS
  end if
  DS.INIT(d^2, m, X, c, τ)
  while $t \leq T$ and $\lambda_{\min}(\sum_{i \in S_{t-1}} x_i x_i^\top) \leq 1 - \gamma\varepsilon$ **do**
    Let c_t be the constant s.t. $(c_t I + \alpha \sum_{i \in S_{t-1}} x_i x_i^\top)^{-2} \in \Delta_{d \times d}$ ▷ Binary search
    $A_t \leftarrow (c_t I_d + \alpha \sum_{i \in S_{t-1}} x_i x_i^\top)^{-2}$
    $q \leftarrow \mathrm{vec}(\frac{A_t}{(1-\varepsilon)/n} + 2\alpha A_t^{1/2})$
    /* Query q */
    $i_t \leftarrow$ DS.QUERYMIN(q) ▷ If DS is AIPE, then here it is QUERYMIN
    $j_t \leftarrow \arg\max_{j \in \overline{S}_{t-1}} B^+(x_j)$ ▷ Def. 3.3.12 with $\frac{1}{c}$ as β
    $S_t \leftarrow S_{t-1} \cup \{j_t\} \setminus \{i_t\}$
    $t \leftarrow t + 1$ ▷ Increase the counter
    /* Updating data structure by swapping j_t and i_t */
    DS.DELETE($x_{i_t} x_{i_t}^\top$)
    DS.INSERT($x_{j_t} x_{j_t}^\top$)
  end while
  return S_{t-1}
end procedure

3.3.4 Approximate Regret Lemma

In this section, we prove the approximate regret lemma. The key consequence of this lemma is to provide a lower bound of the eigenvalue $\lambda_{\min}(\sum_{i\in S} x_i x_i^\top)$.

Lemma 3.3.11 (Approximate regret lemma)**.** *Let $\beta \geq 1$. Suppose $F_t = u_t u_t^\top - v_t v_t^\top$ for vectors $u_t, v_t \in \mathbb{R}^d$ and $A_0, \ldots, A_{T-1} \in \Delta_{d\times d}$ are defined in Def. 3.3.5 some constant $\alpha > 0$. Then, if $\alpha v_t^\top A_t^{1/2} v_t < \beta/2$ for all t, we have for any $U \in \Delta_{d\times d}$,*

$$-\sum_{t=0}^{T-1}\langle F_t, U\rangle \leq \sum_{t=0}^{T-1}(-\frac{\beta u_t^\top A_t u_t}{\beta + 2\alpha u_t^\top A_t^{1/2} u_t} + \frac{\beta v_t^\top A_t v_t}{\beta - 2\alpha v_t^\top A_t^{1/2} v_t}) + \frac{\beta \Delta_\psi(A_0, U)}{\alpha}.$$

Proof. Throughout the proof, we let $\overline{\alpha} := \frac{\alpha}{\beta}$, note that $\overline{\alpha}$ has the property that $\overline{\alpha} v_t^\top A_t^{1/2} v_t < 1/2$, this enables us to use both Claim 3.3.8 and 3.3.9. The proof relies on the mirror descent matrices $\widetilde{A}_t$ and A_t we defined Def. 3.3.4, we need to modify the definition of $\widetilde{A}_t$ with $\overline{\alpha}$ instead of α. Per Claim 3.3.8, we know that $\widetilde{A}_t = (A_{t-1}^{-1/2} + \overline{\alpha} F_{t-1})^{-2}$, and because of their definitions, we know that $\nabla\psi(\widetilde{A}_t) - \nabla\psi(A_{t-1}) + \overline{\alpha} F_{t-1} = 0$ where the gradient is evaluated at $\widetilde{A}_t$. This means that

$$\begin{aligned}
\langle \alpha F_{t-1}, A_{t-1} - U\rangle &= \langle \nabla\psi(A_{t-1}) - \nabla\psi(\widetilde{A}_t), A_{t-1} - U\rangle \\
&= \Delta_\psi(A_{t-1}, U) - \Delta_\psi(\widetilde{A}_t, U) + \Delta_\psi(\widetilde{A}_t, A_{t-1}) \\
&\leq \Delta_\psi(\widetilde{A}_{t-1}, U) - \Delta_\psi(\widetilde{A}_t, U) + \Delta_\psi(\widetilde{A}_t, A_{t-1}). \qquad (3.5)
\end{aligned}$$

Above, the second inequality and the last inequality follow from standard inequlities and generalized Pythagorean Theorem of Bregman divergence. Now, consider the quantity $\Delta_\psi(\widetilde{A}_t, A_{t-1})$:

$$\begin{aligned}
\Delta_\psi(\widetilde{A}_t, A_{t-1}) &= \psi(A_{t-1}) - \psi(\widetilde{A}_t) - \langle \nabla\psi(\widetilde{A}_t), A_{t-1} - \widetilde{A}_t\rangle \\
&= -2\mathrm{tr}[A_{t-1}^{-1/2}] + 2\mathrm{tr}[\widetilde{A}_t^{1/2}] + \langle \widetilde{A}_t^{-1/2}, A_{t-1} - \widetilde{A}_t\rangle \\
&= \langle \widetilde{A}_t^{-1/2}, A_{t-1}\rangle + \mathrm{tr}[\widetilde{A}_t^{1/2}] - 2\mathrm{tr}[A_{t-1}^{1/2}] \\
&= \langle A_{t-1}^{-1/2} + \overline{\alpha} F_{t-1}, A_{t-1}\rangle + \mathrm{tr}[\widetilde{A}_t^{1/2}] - 2\mathrm{tr}[A_{t-1}^{1/2}] \\
&= \overline{\alpha}\langle F_{t-1}, A_{t-1}\rangle + \mathrm{tr}[\widetilde{A}_t^{1/2}] - \mathrm{tr}[A_{t-1}^{1/2}]. \qquad (3.6)
\end{aligned}$$

Combining Eqs. (3.5) and (3.6) and telescoping t from 1 to T yields

$$\begin{aligned}
-\overline{\alpha}\sum_{t=0}^{T-1}\langle F_t, U\rangle &\leq \Delta_\psi(A_0, U) - \Delta_\psi(\widetilde{A}_T, U) + \sum_{t=0}^{T-1}\mathrm{tr}[\widetilde{A}_{t+1}^{1/2}] - \mathrm{tr}[A_t^{1/2}] \\
&\leq \Delta_\psi(A_0, U) + \sum_{t=0}^{T-1}\mathrm{tr}[\widetilde{A}_{t+1}^{1/2}] - \mathrm{tr}[A_t^{1/2}], \qquad (3.7)
\end{aligned}$$

where the second inequality follows from the non-negativitiy of Bregman divergence.

It remains to upper bound $\mathrm{tr}[\widetilde{A}_{t+1}^{1/2}] - \mathrm{tr}[A_t^{1/2}]$.

Set P_t as $\sqrt{\overline{\alpha}}[u_t \;\; v_t] \in \mathbb{R}^{d\times 2}$ and $J = \mathrm{diag}(1, -1) \in \mathbb{R}^{2\times 2}$, we have $\overline{\alpha} F_t = P_t J P_t^\top$. By the definition of $\widetilde{A}_{t+1}^{1/2}$ and the matrix Woodbury formula (Fact. 3.2.3), we have

$$\mathrm{tr}[\widetilde{A}_{t+1}^{1/2}] = \mathrm{tr}[(A_t^{-1/2} + P_t J P_t^\top)^{-1}] = \mathrm{tr}[A_t^{1/2} - A_t^{1/2} P_t (J + P_t^\top A_t^{1/2} P_t)^{-1} P_t^\top A_t^{1/2}]. \qquad (3.8)$$

By linearity of trace operator, it suffices to give a spectral lower bound on the 2×2 matrix $(J + P_t^\top A_t^{1/2} P_t)^{-1/2}$. We will use Claim 3.3.9 as a lower bound:

$$\begin{aligned}
\mathrm{tr}[\widetilde{A}_{t+1}^{1/2}] - \mathrm{tr}[A_t^{1/2}] = & -\mathrm{tr}[-A_t^{1/2} P_t (J + P_t^\top A_t^{1/2} P_t)^{-1} P_t^\top A_t^{1/2}] \\
\leq & -\mathrm{tr}[-A_t^{1/2} P_t (J + \mathrm{diag}(2\overline{\alpha} u_t^\top A_t^{1/2} u_t, 2\overline{\alpha} v_t^\top A_t^{1/2} v_t))^{-1} P_t^\top A_t^{1/2}] \\
= & -\frac{\overline{\alpha} u_t^\top A_t u_t}{1 + 2\overline{\alpha} u_t^\top A_t^{1/2} u_t} + \frac{\overline{\alpha} v_t^\top A_t v_t}{1 - 2\overline{\alpha} v_t^\top A_t^{1/2} v_t}. \qquad (3.9)
\end{aligned}$$

Plugging Eq. (3.9) into Eq. (3.7), we arrive at the desired result:

$$-\sum_{t=0}^{T-1} \langle F_t, U \rangle \leq \sum_{t=0}^{T-1} (-\frac{\beta u_t^\top A_t u^t}{\beta + 2\alpha u_t^\top A_t^{1/2} u_t} + \frac{\beta v_t^\top A_t v_t}{\beta - 2\alpha v_t^\top A_t^{1/2} v_t}) + \frac{\beta}{\alpha} \Delta_\psi(A_0, U).$$

□

3.3.5 Approximate Swapping Lemma

The goal of this section is to present and prove Lemma 3.3.13. We start with a helpful definition.

Definition 3.3.12 (B functions)**.** Let α, β denote two fixed parameters. Let A denote a fixed matrix. We define function $B^+ : \mathbb{R}^d \to \mathbb{R}$ and $B^- : \mathbb{R}^d \to \mathbb{R}$ as follows:

$$B^+(x) = \frac{\langle A, xx^\top \rangle}{\beta + 2\alpha \langle A^{1/2}, xx^\top \rangle},$$
$$B^-(x) = \frac{\langle A, xx^\top \rangle}{\beta - 2\alpha \langle A^{1/2}, xx^\top \rangle}.$$

Lemma 3.3.13. *Let $\beta \in [1, \gamma - 1)$ and $\varepsilon \in (0, 1/\gamma]$. For every subset $S \subset [m]$ of cardinality n (let $\overline{S}$ denote $[m] \setminus S$), suppose $\lambda_{\min}(\sum_{i \in S} x_i x_i^\top) \leq 1 - \gamma\varepsilon$ and $A = (cI + \alpha \sum_{i \in S} x_i x_i^\top)^{-2}$, where $c \in \mathbb{R}$ is the unique number such that $A \succeq 0$ and $\mathrm{tr}[A] = 1$. For any $\alpha = \sqrt{d}\beta/\varepsilon$ and $n \geq \frac{6}{\gamma - 1 - \beta} d/\varepsilon^2$, we have*

- *Part 1. There exists $i \in S$ such that $2\alpha x_i^\top A x_i < \beta$ and $B^-(x_i) \leq \frac{1-\varepsilon}{\beta n}$,*
- *Part 2. There exists $j \in \overline{S}$ such that $B^+(x_j) \geq \frac{1}{\beta n}$.*

Proof. In this proof, we will extensively use Claim 3.3.10, therefore, we pre-compute the value $d + \alpha\sqrt{d}$ and $\sqrt{d}/\alpha$ here for references. By the choice of our α, we have

$$d + \alpha\sqrt{d} = (1 + \frac{\beta}{\varepsilon}) d \ , \sqrt{d}/\alpha = \frac{\varepsilon}{\beta}. \qquad (3.10)$$

We also define the quantity $\nu := \min_{i \in S, 2\alpha x_i^\top A x_i < \beta} B^-(x_i)$ which will be used throughout our proof.

The proof directly follows from combining Claim 3.3.14 and Claim 3.3.15. □

3.3.6 Approximate Swapping Lemma, Part 1

In this section, we will prove that as long as we enter the main while loop of the algorithm, we can always find an index $i \in S$ such that $B^-(x_i)$ is small.

Claim 3.3.14 (Part 1 of Lemma 3.3.13)**.** *There exists $i \in S$ such that $2\alpha x_i^\top A x_i < \beta$ and $B^-(x_i) \le \frac{1-\varepsilon}{\beta n}$.*

Proof. To demonstrate the existence of such an i, it suffices to show that $\min_{i\in S, 2\alpha x_i^\top A x_i<\beta} B^-(x_i) \le \frac{1-\varepsilon}{\beta n}$, we use ν to denote this minimum value. Note that $\nu > 0$, due to the fact $2\alpha x_i^\top A x_i < \beta$ and A is positive definite. To start off, we first show that there always exists an i such that $2\alpha\langle A^{1/2}, x_i x_i^\top\rangle < 1$. Define $Z = \sum_{i\in S} x_i x_i^\top$, and by definition $A = (cI + \alpha\sum_{i\in S} x_i x_i^\top)^{-2} = (\alpha Z + cI)^{-2}$. Assume for the sake of contradiction that such i does not exists. We have

$$\sum_{i\in S} 2\alpha\langle A^{1/2}, x_i x_i^\top\rangle = 2\alpha\langle A^{1/2}, Z\rangle \ge |S| = n. \tag{3.11}$$

On the other hand, because $Z \succeq 0$ and $\lambda_{\min}(Z) < 1$, invoking Claim 3.3.10 we get

$$2\alpha\langle A^{1/2}, Z\rangle \le 2d + 2\alpha\sqrt{d},$$

which contradicts Eq. (3.11) given the choice of α and $n > 4d/\varepsilon$. Thus, there must exist $i \in S$ such that $2\alpha\langle A^{1/2}, x_i x_i^\top\rangle < 1$. Since we set $\beta \ge 1$, this means we can always find an index i such that $2\alpha\langle A^{1/2}, x_i x_i^\top\rangle < \beta$ holds. By the same token, we also have $\sum_{i\in S}(\beta - 2\alpha\langle A^{1/2}, x_i x_i^\top\rangle) \ge 0$. We claim that

$$(\beta - 2\alpha\langle A^{1/2}, x_i x_i^\top\rangle)\nu \le \langle A, x_i x_i^\top\rangle, \text{ for all } i \in S,$$

because if $2\alpha\langle A^{1/2}, x_i x_i^\top\rangle \ge \beta$ the LHS is non-positive while the RHS is always non-negative due to the positive semi-definiteness of A. Subsequently,

$$\begin{aligned}
\nu &\le \frac{\sum_{i\in S}\langle A, x_i x_i^\top\rangle}{\sum_{i\in S}(\beta - 2\alpha\langle A^{1/2}, x_i x_i^\top\rangle)} \\
&\le \frac{\sqrt{d}/\alpha + \lambda_{\min}(\sum_{i\in S} x_i x_i^\top)}{\beta n - 2d - 2\alpha\sqrt{d}} \\
&\le \frac{\varepsilon/\beta + 1 - \gamma\varepsilon}{\beta n(1 - \beta\varepsilon/3)} \\
&\le \frac{1-\varepsilon}{\beta n}
\end{aligned}$$

where the first step holds because the denominator is strictly positive as we have shown; the second step is due to Claim 3.3.10; the third step has used our choices α and n and our assumption $\lambda_{\min}(\sum_{i\in S} x_i x_i^\top) \le 1 - \gamma\varepsilon$; and the forth step has used $1 - \beta\varepsilon/3 < 1$. We have thus proved that $\nu \le (1-\varepsilon)/(\beta n)$. This proves the existence of the i we want.

□

3.3.7 Approximate Swapping Lemma, Part 2

In this section, we prove the other key gredient for the swapping to proceed, i.e., there exists an $j \in \overline{S}$ such that $B^+(x_j)$ is large.

Claim 3.3.15 (Part 2 of Lemma 3.3.13)**.** *There exists $j \in \overline{S}$ such that $B^+(x_j) \geq \frac{1}{\beta n}$.*

Proof. Define $t = 1/(\beta n)$. To prove Part 2 it suffices to show that

$$\sum_{j\in\overline{S}} \pi_j \langle A, x_j x_j^\top \rangle \geq t \cdot \sum_{j\in\overline{S}} \pi_j (\beta + 2\alpha \langle A^{1/2}, x_j x_j^\top \rangle), \tag{3.12}$$

because $\pi_j \geq 0$ for all $j \in [m]$. Recall that $\sum_{j=1}^m \pi_j = n$, $\sum_{j=1}^m \pi_j x_j x_j^\top = I_d$. We then have

$$\begin{aligned}
\sum_{j\in\overline{S}} \pi_j (\beta + 2\alpha \langle A^{1/2}, x_j x_j^\top \rangle) &\leq \beta(n - \sum_{j\in S} \pi_j) + 2\alpha \cdot \sum_{j\in\overline{S}} \pi_j \langle A^{1/2}, x_j x_j^\top \rangle \\
&\leq \beta(n - \sum_{j\in S} \pi_j) + 2\alpha \cdot \sum_{j=1}^m \pi_j \langle A^{1/2}, x_j x_j^\top \rangle \\
&= \beta n - \beta \sum_{j\in S} \pi_j + 2\alpha \langle I, A^{1/2} \rangle \\
&= \beta n - \beta \sum_{j\in S} \pi_j + 2\alpha \cdot \mathrm{tr}[A^{1/2}].
\end{aligned}$$

Similarly,

$$\begin{aligned}
\sum_{j\in\overline{S}} \pi_j \langle A, x_j x_j^\top \rangle &= \langle I - \sum_{j\in S} \pi_j x_j x_j^\top, A \rangle \\
&= \mathrm{tr}[A] - \sum_{j\in S} \pi_j \langle A, x_j x_j^\top \rangle
\end{aligned}$$

Subsequently,

$$\begin{aligned}
&\sum_{j\in\overline{S}} \pi_j \langle A, x_j x_j^\top \rangle - t \cdot \sum_{j\in\overline{S}} \pi_j (\beta + 2\alpha \langle A^{1/2}, x_j x_j^\top \rangle) \\
&\geq \mathrm{tr}[A] - \sum_{j\in S} \pi_j \langle A, x_j x_j^\top \rangle - t \cdot \beta \cdot (n - \sum_{j\in S} \pi_j) - 2\alpha t \cdot \mathrm{tr}[A^{1/2}] \\
&\geq 1 - \sum_{j\in S} \pi_j \langle A, x_j x_j^\top \rangle - t \cdot \beta \cdot (n - \sum_{j\in S} \pi_j) - 2\alpha t \sqrt{d} \\
&= 1 - t\beta n - 2t\alpha\sqrt{d} - \sum_{j\in S} \pi_j (\langle A, x_j x_j^\top \rangle - t\beta) \\
&\geq 1 - t\beta n - 2t\alpha\sqrt{d} - \sum_{j\in S} \max\{\langle A, x_j x_j^\top \rangle - t\beta, 0\} \\
&= 1 - t\beta n - 2t\alpha\sqrt{d} - \sum_{j\in S} (\langle A, x_j x_j^\top \rangle - t\beta) - \sum_{j\in S} \max\{(t\beta - \langle A, x_j x_j^\top \rangle), 0\}
\end{aligned}$$

$$\geq 1 - 2t\alpha\sqrt{d} - \sqrt{d}/\alpha - \lambda_{\min}(\sum_{j\in S} x_j x_j^\top) - \sum_{j\in S} \max\{(t\beta - \langle A, x_j x_j^\top\rangle), 0\}$$

$$\geq (\gamma - \beta)\varepsilon - \frac{2d}{\varepsilon n} - \sum_{j\in S} \max\{t\beta - \langle A, x_j x_j^\top\rangle, 0\} \tag{3.13}$$

where the second step follows from Fact 3.2.2 and $\mathrm{tr}[A] = 1$. The forth step follows from $\pi_j \leq 1$ for all j, the second-to-last step follows from we apply $\sum_{j\in S}\langle A, x_j x_j^\top\rangle \leq \sqrt{d}/\alpha + \lambda_{\min}(\sum_{j\in S} x_j x_j^\top)$ which comes from Claim 3.3.10. The fifth step comes from the fact that $\max\{x, 0\} - \max\{-x, 0\} = x$. Finally, the last step comes from the choices of α, t and $\lambda_{\min}(\sum_{j\in S} x_j x_j^\top) \leq 1 - \gamma\varepsilon$.

Furthermore, because $(\beta - 2\alpha\langle A^{1/2}, x_i x_i^\top\rangle)\nu \leq \langle A, x_i x_i^\top\rangle$ for all $i \in S$, using Claim 3.3.10 we have

$$\sum_{i\in S'}(\beta\nu - \langle A, x_i x_i^\top\rangle) \leq \sum_{i\in S'} 2\nu\alpha\langle A^{1/2}, x_i x_i^\top\rangle \leq 2\nu(d + \alpha\sqrt{d}),$$

for all $S' \subseteq S$.

Consider $S' = \{i \in S : \beta t - \langle A, x_i x_i^\top\rangle \geq 0\}$. We then have

$$\begin{aligned}
\sum_{j\in S'} \max\{\beta t - \langle A, x_j x_j^\top\rangle, 0\} &= \sum_{j\in S'}(\beta t - \langle A, x_j x_j^\top\rangle) \\
&= \beta(t-\nu)|S'| + \sum_{j\in S'}(\beta\nu - \langle A, x_j x_j^\top\rangle) \\
&\leq \beta(t-\nu)n + 2\nu(d + \alpha\sqrt{d}) \\
&\leq \varepsilon + \frac{4d/\varepsilon}{n}
\end{aligned} \tag{3.14}$$

where the last two inequalities hold because $t - \nu = \varepsilon/(\beta n) \geq 0$, $|S'| \leq |S| = n$, $\nu \leq 1/(\beta n)$ and the choice of α.

Combining Eqs.(3.13) and (3.14) we arrive at

$$\sum_{j\in\overline{S}} \pi_j\{\langle A, x_j x_j^\top\rangle - t(\beta + 2\alpha\langle A^{1/2}, x_j x_j^\top\rangle)\} \geq (\gamma - 1 - \beta)\varepsilon - \frac{6d}{\varepsilon n}.$$

By choice of n, the RHS of the above inequality is non-negative, which finishes the proof of Eq. (3.12) and thus also the proof of Part 2. □

3.3.8 Implication of Swapping Lemma

Note that Lemma 3.3.13 gives rise to a natural swapping algorithm: at each round, we can find an index $i \in S$ with $2\alpha x_i^\top A x_i < \beta$ and $B^-(x_i) \leq \frac{1-\varepsilon}{\beta n}$ and an index $j \in \overline{S}$ with $B^+(x_j) \geq \frac{1}{\beta n}$, then swap them. As demonstrated in Alg. 19, the task of finding i is a search for minimum inner product, which can be implemented via our data structures. On the other hand, the search for j is a Max-IP search. Though it can also be realized by our Max-IP data structure, the two

approximation factor $c_{\text{Max-IP}}$ and $c_{\text{Min-IP}}$ would impose a restriction on the relationship between them and ε. Also, due to such precision requirement, one can not use a lossy estimation for Max-IP data structure as in the BSS case. Hence, we only present an algorithmic result that deals with one-side of the sets. We will discuss how to implement the two data structures paradigm towards the end of next section.

We also remark that by considering to finding a β-approximation point instead of an point with distance exactly 1, we require the cardinality of S to be larger since $n \propto \frac{d/\varepsilon^2}{\gamma-1-\beta}$. This is an interesting trade-off compared to the approximate result we get in one-sided Kadison-Singer, where the quality of solution becomes worse when β becomes larger. Here, the quality of solution is unaffected while we have more leeway to pack vectors into S. To some extent, this makes the problem easier similar to a worse quality of solution.

3.3.9 Main Result

In this section, we present the correctness and runtime analysis of Algorithm 19. The correctness follows from the approximate regret and swap lemma, while the runtime comes from the approximate Min-IP data structure.

Theorem 3.3.16 (Formal version of Theorem 1.2.3)**.** *Let $\pi \in [0,1]^m$ with $\|\pi\|_1 \leq n$ and $\sum_{i=1}^m \pi_i x_i x_i^\top = I_d$. Let $\gamma \geq 3$ and $\varepsilon \in (0, \frac{1}{\gamma}]$. Then, there exists a subset $S \subset [m]$ with $|S| \leq n$ such that*

$$\lambda_{\min}(\sum_{i \in S} x_i x_i^\top) \geq 1 - \gamma \cdot \varepsilon.$$

Let $\tau, \delta \in (0,1)$ and $c \in (\frac{1}{\gamma-1}, 1)$. If $n \geq \frac{6d/\varepsilon^2}{\gamma-1-2/c}$ and $\alpha = \sqrt{d}/(c\varepsilon)$, then there exists a randomized algorithm with success probability at least $1-\delta$ and running time $\mathcal{T} = \min\{\mathcal{T}_{\text{SmallIter}}, \mathcal{T}_{\text{LargeIter}}\}$ where

- *For $\mathcal{T}_{\text{SmallIter}}$, we have $c \in (\tau, \frac{1.01\tau}{0.01+\tau})$ and*

$$\mathcal{T}_{\text{SmallIter}} = \widetilde{O}(\mathcal{T}_{\text{mat}}(m,d,d) + nd^2 + \varepsilon^{-1} n \cdot (d^\omega + n + (m-n) \cdot d^2)).$$

- *For $\mathcal{T}_{\text{LargeIter}}$, we have $c \in (\tau, \frac{400\tau}{399+\tau})$ and*

$$\mathcal{T}_{\text{LargeIter}} = \widetilde{O}(\mathcal{T}_{\text{mat}}(m,d,d) + (n^{1.01} + \text{nnz}(X))d^2 + \varepsilon^{-1} n \cdot (d^\omega + (n^{0.01} + z) \cdot d^2 + (m-n) \cdot d^2)),$$

  where $z = \max_{i \in [m]} \text{nnz}(x_i)$.

Proof. We will show Alg. 19 satisfies the properties in the theorem statement. Similar to the proof of Theorem 3.2.10, we need to scale down the query point by a factor of τ. This means each query will return an index $i \in S_{t-1}$ such that

$$\frac{x_i^\top A_t x_i}{(1-\varepsilon)/n} + 2\alpha x_i^\top A_t^{1/2} x_i \leq \frac{1}{c},$$

Set $\beta = \frac{1}{c}$, note this is equivalent to find an index i satisfying $B^-(x_i) \leq \frac{1-\varepsilon}{\beta n}$.

On the other hand, we can search the index $j \in \overline{S}_{t-1}$ such that

$$B^+(x_j) \geq \frac{1}{\beta n}$$

This means that at each iteration, we either have

$$\lambda_{\min}(\sum_{i \in S} x_i x_i^\top) \geq 1 - \gamma\varepsilon,$$

which we are done, or we can find i_t and j_t such that

$$B^-(x_{i_t}) - B^+(x_{j_t}) \leq -\frac{\varepsilon}{\beta n}.$$

Combining this fact with Lemma 3.3.11 and Claim 3.3.7, we have

$$\begin{aligned} -\langle Z_0 + \sum_{t=0}^{T-1} F_t, U \rangle &\leq \sum_{t=0}^{T-1} \beta(B^-(x_{i_t}) - B^+(x_{j_t})) + \frac{2\beta\sqrt{d}}{\alpha} \\ &\leq -T \cdot \frac{\varepsilon}{\beta n} + 2\varepsilon, \end{aligned}$$

Since we can choose U such that

$$-\langle Z_0 + \sum_{t=0}^{T-1} F_t, U \rangle = -\lambda_{\min}(Z_0 + \sum_{t=0}^{T-1} F_t) = -\lambda_{\min}(\sum_{i \in S_T} x_i x_i^\top),$$

this gives a lower bound on the desired eigenvalue we want:

$$\lambda_{\min}(\sum_{i \in S_T} x_i x_i^\top) \geq T \cdot \frac{\varepsilon}{\beta n} - 2\varepsilon.$$

Since $T = \frac{\beta n}{\varepsilon}$, it is lower bounded by $1 - 2\varepsilon > 1 - \gamma\varepsilon$, and we have completed the proof of correctness.

For the running time, we separately consider initialization and cost per iteration. In initialization phase,

- Computing $X(X^\top \mathrm{diag}(\pi) X)^{-1/2}$ takes $O(\mathcal{T}_{\mathrm{mat}}(m, d, d))$ time;
- The Initialization time for data structure with n random points is either $\widetilde{O}(nd^2)$ (see Theorem 2.4.6) or $\widetilde{O}((n^{1.01} + \mathrm{nnz}(X))d^2)$ (see Theorem 2.3.24);

For each iteration, we perform the following:

- Computing eigen-decomposition of $\sum_{i \in S_{t-1}} x_i x_i^\top$ takes $O(d^\omega)$ time;
- Using binary search to finding c_t takes $O(d^\omega \log d/(c\varepsilon))$ since the searching range is $O(\alpha + \sqrt{d})$ and each search takes d^ω to form the matrix and compute its trace;
- The time of querying data structure is either $\widetilde{O}(n + d^2)$ (see Theorem 2.4.6) or $\widetilde{O}(n^{0.01})$ (see Theorem 2.3.24);
- The brute force search for j takes $O((m - n) \cdot d^2)$ if we pre-compute A_t and $A_t^{1/2}$;
- The insertion and deletion of point x_{j_t} takes either $\widetilde{O}(d^2)$ (see Theorem 2.4.6) or $\widetilde{O}((n^{0.01} + \mathrm{nnz}(x_{j_t}))d^2)$ (see Theorem 2.3.24) time.

This concludes the proof of running time. □

Chapter 4

Faster Training of Deep, Over-parametrized Neural Networks

As machine learning grows in its popularity and practical applications, it is more and more important to develop algorithms that can train deep neural network efficiently. In this thesis, we focus on neural networks that are both deep and over-parametrized: most practical networks are deep, and provable convergence results have been obtained for over-parametrized networks.

Let n denotes the number of data points, d denotes the data dimension, m denotes the width of the network and L denotes the number of layers, we obtain an algorithm that runs in time $O(m^{2-\Omega(1)})$ per training iteration, breaking the quadratic barrier for training deep and over-parametrized networks. Different from the standard first order methods, we use second order methods, which are considered much harder to compute and implement efficiently. One of the big advantages of second order methods is its adaptive adjustment of step size, which significantly reduces the hyperparameters one needs to tune.

Techniques centered around our algorithms include a shifted-ReLU sparsifier, the low rank maintenance data structure for Task 1.1.10 and use fast tensor sketching techniques as a good preconditioner to compute the Gram matrix.

This chapter is based on the following arXiv document: https://arxiv.org/pdf/2112.07628.pdf coauthored by the thesis author.

4.1 Problem Setup

Let $X \in \mathbb{R}^{m_0 \times n}$ denote the data matrix with n data points and m_0 features. By proper re-scaling, we have $\|x_i\|_2 = 1$ for all $i \in [n]$. Consider an L layer neural network with one vector $a \in \mathbb{R}^{m_L}$ and L matrices $W_L \in \mathbb{R}^{m_L \times m_{L-1}}$, $\cdots$, $W_2 \in \mathbb{R}^{m_2 \times m_1}$ and $W_1 \in \mathbb{R}^{m_1 \times m_0}$. We will use $W_\ell(t)$ to denote the weight matrix at layer ℓ at time t, and $\nabla W_\ell(t)$ to denote its gradient. We also use $W(t) = \{W_1(t), \ldots, W_L(t)\}$ to denote the collection of weight matrices at time t.

Architecture. We first describe our network architecture. The network consists of L hidden layers, each represented by a weight matrix $W_\ell \in \mathbb{R}^{m_\ell \times m_{\ell-1}}$ for any $\ell \in [L]$. The output layer consists of a vector $a \in \mathbb{R}^{m_L}$. We define the neural network prediction function $f : \mathbb{R}^{m_0} \to \mathbb{R}$ as

follows:

$$f(W, x) = a^\top \varphi(W_L(\varphi(\cdots \varphi(W_1 x)))),$$

where $\varphi : \mathbb{R} \to \mathbb{R}$ is the (shifted) ReLU activation function ($\sigma_b(x) = \max\{x - b, 0\}$) applied coordinate-wise to a vector.

We measure the loss via the squared-loss function:

$$\mathcal{L}(W) = \frac{1}{2} \sum_{i=1}^{n} (y_i - f(W, x_i))^2.$$

This is also the objective function for our training.

The prediction function $f_t : \mathbb{R}^{m_0 \times n} \to \mathbb{R}^n$ is defined as

$$f_t(X) = \begin{bmatrix} f(W(t), x_1) & f(W(t), x_2) & \cdots & f(W(t), x_n) \end{bmatrix}^\top.$$

Initialization. Our neural networks are initialized as follows:

- For each $\ell \in [L]$, the layer-ℓ's weight parameter $W_\ell(0) \in \mathbb{R}^{m_\ell \times m_{\ell-1}}$ is initialized such that each entry is sampled from $\mathcal{N}(0, \frac{2}{m_\ell})$.
- Each entry of a is an i.i.d. sample from $\{-1, +1\}$ uniformly at random.

Gradient. In order to write gradient in an elegant way, we define some artificial variables:

$$\begin{aligned}
g_{i,1} &= W_1 x_i, & h_{i,1} &= \varphi(W_1 x_i), & &\forall i \in [n] \\
g_{i,\ell} &= W_\ell h_{i,\ell-1}, & h_{i,\ell} &= \varphi(W_\ell h_{i,\ell-1}), & &\forall i \in [n], \forall \ell \in [L] \backslash \{1\} \\
D_{i,1} &= \mathrm{diag}\big(\varphi'(W_1 x_i)\big), & & & &\forall i \in [n] \\
D_{i,\ell} &= \mathrm{diag}\big(\varphi'(W_\ell h_{i,\ell-1})\big), & & & &\forall i \in [n], \forall \ell \in [L] \backslash \{1\}
\end{aligned} \tag{4.1}$$

Using the definitions of f and h, we have

$$f(W, x_i) = a^\top h_{i,L}, \quad \in \mathbb{R}, \quad \forall i \in [n]$$

We can compute the gradient of $\mathcal{L}$ in terms of $W_\ell \in \mathbb{R}^{m_\ell \times m_{\ell-1}}$, for all $\ell \geq 2$

$$\frac{\partial \mathcal{L}(W)}{\partial W_\ell} = \sum_{i=1}^{n} (f(W, x_i) - y_i) \underbrace{D_{i,\ell}}_{m_\ell \times m_\ell} \left(\prod_{k=\ell+1}^{L} \underbrace{W_k^\top}_{m_{k-1} \times m_k} \underbrace{D_{i,k}}_{m_k \times m_k} \right) \underbrace{a}_{m_L \times 1} \underbrace{h_{i,\ell-1}^\top}_{1 \times m_{\ell-1}} \tag{4.2}$$

Note that the gradient for $W_1 \in \mathbb{R}^{m_1 \times m_0}$ (recall that $m_0 = d$) is slightly different and can not be written by general form. By the chain rule, the gradient of the variables in W_1 can be expressed as:

$$\frac{\partial \mathcal{L}(W)}{\partial W_1} = \sum_{i=1}^{n} (f(W, x_i) - y_i) \underbrace{D_{i,1}}_{m_1 \times m_1} \left(\prod_{k=2}^{L} \underbrace{W_k^\top}_{m_{k-1} \times m_k} \underbrace{D_{i,k}}_{m_k \times m_k} \right) \underbrace{a}_{m_L \times 1} \underbrace{x_i^\top}_{1 \times m_0} \tag{4.3}$$

It is worth noting that the gradient matrix is of rank n, since it's a sum of n rank-1 matrices.

Jacobian. For each layer $\ell \in [L]$ and time $t \in [T]$, we define the Jacobian matrix $J_{\ell,t} \in \mathbb{R}^{n \times m_\ell m_{\ell-1}}$ via the following formulation:

$$J_{\ell,t} := \left[\mathrm{vec}(\tfrac{\partial f(W(t),x_1)}{\partial W_\ell(t)}) \quad \mathrm{vec}(\tfrac{\partial f(W(t),x_2)}{\partial W_\ell(t)}) \quad \cdots \quad \mathrm{vec}(\tfrac{\partial f(W(t),x_n)}{\partial W_\ell(t)}) \right]^\top.$$

The Gram matrix at layer ℓ and time t is then defined as $G_{\ell,t} = J_{\ell,t} J_{\ell,t}^\top \in \mathbb{R}^{n \times n}$ whose (i,j)-th entry is

$$\Big\langle \frac{\partial f(W(t), x_i)}{\partial W_\ell}, \frac{\partial f(W(t), x_j)}{\partial W_\ell} \Big\rangle.$$

Remark 4.1.1. *For simplicity, we assume for any $i \in [L]$, $m_i = m$ for some m. Our methods can be generalized to different m_i's by designing a more general type of sketching matrices.*

4.2 Probability Tools

Lemma 4.2.1 (Chernoff bound [24])**.** *Let $X = \sum_{i=1}^n X_i$, where $X_1, \ldots, X_n$ are n independent 0/1 Bernoulli random variables with $\Pr[X_i = 1] = p_i$ for $i \in [n]$. Let $\mu = \mathbb{E}[X]$. Then,*
1. $\Pr[X \geq (1+\varepsilon)\mu] \leq \exp(-\varepsilon^2 \mu/3)$, $\forall\, \varepsilon > 0$ *;*
2. $\Pr[X \leq (1-\varepsilon)\mu] \leq \exp(-\varepsilon^2 \mu/2)$, $\forall\, \varepsilon \in (0,1)$.

Lemma 4.2.2 (Hoeffding bound [37])**.** *Let $Y_1, \cdots, Y_n$ denote n independent bounded variables in $[\alpha_i, \beta_i]$. For $Y = \sum_{i=1}^n Y_i$ and $\tau > 0$, we have*

$$\Pr[|Y - \mathbb{E}[Y]| \geq \tau] \leq 2 \exp\left(- \frac{2\tau^2}{\sum_{i=1}^n (\beta_i - \alpha_i)^2} \right).$$

Lemma 4.2.3 (Bernstein inequality [14])**.** *Let $Z_1, \cdots, Z_n$ denote n independent mean-zero random variables. Suppose that $|Z_i| \leq B$ almost surely, for all $i \in [n]$. Let $\sigma^2 := \sum_{i \in [n]} \mathbb{E}[Z_i^2]$. Then, for all $\tau > 0$,*

$$\Pr\left[\sum_{i=1}^n Z_i > \tau \right] \leq \exp\left(- \frac{\tau^2/2}{\sigma^2 + B\tau/3} \right).$$

Lemma 4.2.4 (Anti-concentration of Gaussian distribution)**.** *Let $X \sim \mathcal{N}(0, \sigma^2)$, then*

$$\Pr[|X| \leq t] = \Theta(t/\sigma).$$

Lemma 4.2.5 (Concentration of subgaussian random variables)**.** *Let $a \in \mathbb{R}^n$ be a vector where each coordinate of a is an independent subgaussian random variable with parameter σ^2. Then, for any vector $x \in \mathbb{R}^n$,*

$$\Pr[|\langle a, x \rangle| \geq t \cdot \|x\|_2] \leq 2 \exp\left(- \frac{t^2}{2\sigma^2} \right).$$

Lemma 4.2.6 (Small ball probability, [66]). *Let $h \in \mathbb{R}^n$ be a vector such that $|h_i| \geq \delta$ for all $i \in [n]$. Let $a \in \{-1,1\}^n$ be a random vector such that each coordinate is an independent Rademacher random variable. Then, for some absolute constants C_1, C_2, we have for any $t > 0$,*

$$\Pr[|\langle h, a\rangle| \leq t] \leq \min\left\{\frac{C_1 t}{\|h\|_2}, \frac{C_2 t}{\delta\sqrt{n}}\right\}.$$

Fact 4.2.7 (Minimum eigenvalue of Hadamard product matrices, [68]). *Let $A, B \in \mathbb{R}^{n\times n}$ be two PSD matrices. Then, we have*

$$\lambda_{\min}(A \odot B) \geq \min_{i\in[n]} (B)_{i,i} \cdot \lambda_{\min}(A).$$

4.3 Complete Algorithm and its Runtime Analysis

In this section, we first present our complete algorithm, then we analyze its running time. We show that as long as we use the shifted ReLU activation so that the number of activated neurons is sparse, then all our operations can be realized in subquadratic time.

Algorithm 20 Training last layer.

procedure CompleteAlgorithm($X \in \mathbb{R}^{d\times n}, y \in \mathbb{R}^n$) ▷ Theorem 4.3.1
  /*Initialization*/
  Initialize $W_\ell(0), \forall \ell \in [L]$
  Compute $h_{i,\ell}$ for $\ell \in [L-1]$ ▷ Takes $O(nm^2L)$ time
  Store $h_{i,L-1}$ in memory, $\forall i \in [n]$
  LowRankMaintenance LMR ▷ Algorithm 11
  LMR.Init($\{W_1(0) \dots, W_L(0)\}$)
  for $t = 0 \to T$ **do**
    /*Forward computation*/
    $v_{i,L} \leftarrow h_{i,L-1}, \forall i \in [n]$
    $g_{i,L} \leftarrow$ LMR.Query$(L, h_{i,L-1}), \forall i \in [n]$ ▷ Takes $o(nm^2)$ time
    $h_{i,L} \leftarrow \varphi(g_{i,L}), \forall i \in [n]$ ▷ $h_{i,L}$ is sparse
    $D_{i,L} \leftarrow \mathrm{diag}(\varphi'(g_{i,L})), \forall i \in [n]$ ▷ $D_{i,L}$ is sparse
    $f_t \leftarrow [a^\top h_{1,L}, \dots, a^\top h_{n,L}]^\top$ ▷ Takes $O(nm)$ time
    /*Backward computation*/
    $u_{i,L} \leftarrow a^\top D_{i,L}, \forall i \in [n]$ ▷ Takes $o(nm)$ time
    $g_L \leftarrow$ FastTensorRegression$(\{u_{i,L}\}_{i=1}^n, \{v_{i,L}\}_{i=1}^n, f_t - y)$ with precision $\sqrt{\lambda_L/n}$
    LMR.Update$(\{g_{L,i}u_{i,L}\}_{i=1}^n, \{v_{i,L}\}_{i=1}^n)$
  end for
end procedure

Theorem 4.3.1 (Formal version of Theorem 1.2.4). *Let $X \in \mathbb{R}^{d\times n}$ and $y \in \mathbb{R}^n$, and let k denote the sparsity of $D_{i,\ell}$ and s denote the sparsity of $\Delta D_{i,\ell}$, $\forall \ell \in [L], i \in [n]$. Let m denote the width*

of neural network, L denote the number of layers and α denote the dual matrix multiplication exponent (Def. 2.5.1),then the running time of Algorithm 20 is

$$O(\mathcal{T}_{\mathrm{init}} + T \cdot \mathcal{T}_{\mathrm{iter}}),$$

where

$$\begin{aligned}\mathcal{T}_{\mathrm{init}} &= O(m^2 nL),\\ \mathcal{T}_{\mathrm{iter}} &= \widetilde{O}(n \cdot (m^{2-\alpha} + m \cdot (s+k))).\end{aligned}$$

Therefore, the cost per iteration *of Algorithm 20 is*

$$\widetilde{O}(n \cdot (m^{2-\alpha} + m \cdot (s+k))).$$

Proof. We analyze $\mathcal{T}_{\mathrm{init}}$ and $\mathcal{T}_{\mathrm{iter}}$ separately.

Initialization time. We will first initialize $(L-1)$ $m \times m$ matrices and one $m \times d$ matrix, which takes $O(m^2 L)$ time. Compute $h_{i,L-1}$ for all $i \in [n]$ takes $O(m^2 nL)$ time. Finally, initialize the data structure takes $O(m^2 L)$ time. Hence, $\mathcal{T}_{\mathrm{init}} = O(m^2 nL)$.

Cost per iteration. For each iteration, we perform one forward computation from layer 1 to L, then we train the last layer via solving a regression based on its Jacobian matrix.

- **Forward computation:** In forward computation, we first compute $g_{i,L} \in \mathbb{R}^m$, which involves using the QUERY procedure of LMR data structure, hence by Lemma 2.5.2, it takes $O(m \cdot (s + k + m^\alpha))$ time. Compute $h_{i,L}$ and $D_{i,L}$ takes $O(m)$ time. Hence the overall runtime of forward computation is $O(nm \cdot (s + k + m^\alpha))$.
- **Backward computation:** In backward computation, we first compute $u_{i,L} \in \mathbb{R}^m$, which takes $O(m(s+k))$ time owing to the sparsity of $D_{i,L}$. Then, we call Algorithm 22 to solve the Gram regression problem, which due to Theorem 4.5.14 takes $\widetilde{O}(mn + n^\omega)$ time. Note that even we want a high probability version of the solver with $e^{-\log^2 nL}$ failure probability, we only pay extra $\log^2 nL$ term in running time, which is absorbed by the $\widetilde{O}(\cdot)$ notation. Finally, the update takes $O(m^{2-\alpha} n)$ amortized time owing to Lemma 2.5.2. Put things together, we get an overall running time of $\widetilde{O}(n(m(s+k) + m^{2-\alpha}))$ time.

This concludes the proof of our Theorem. □

Corollary 4.3.2. *Suppose the network width m is chosen as in 4.7.20 and the shift parameter b is chosen as in 4.7.7, then the cost per iteration of Algorithm 20 is*

$$\widetilde{O}(m^{2-\alpha} n).$$

Remark 4.3.3. *As long as the neural network is wide enough, as in 4.7.20 and we choose the shift threshold properly, as in 4.7.7, then we can make sure that both sparsity parameters k and s to be $o(m)$, and we achieve subquadratic cost per iteration.*

We also compare our result with our approaches. Note that a naive implementation of variants of gradient descent will take $O(nm^2)$ time, namely, one evaluates the gradient with respect to each data point and sum them up. By batching the n data points and use fast rectangular

matrix multiplication, the running time can be improved to $\mathcal{T}_{\mathrm{mat}}(m, n, m)$, in the setting where $n \leq m^{\alpha}$, this will only take $O(m^{2+o(1)})$ time.

In the specific parameter set we choose, we need that $m^{2-\alpha}n < m^2$ to truly beat the quadratic barrier, which implies that $n < m^{\alpha}$. As we will later see the choice of m (Def. 4.7.20), we will have $n \leq m^{1/4}$, which means that we get a truly subquadratic time *in m.*

4.4 Efficient Computation of Rank-1 Decompositions

We note that to compute the vectors $u_{i,L}$, we need to compute the vector $g_{i,L}$ on line 11 of Algorithm 20, which is a matrix-vector product query in which the matrix is updated in a low rank fashion. This is Task 1.1.10, and we will use the low rank maintenance data structure to speed up this step.

We illustrate the method to compute the vectors $u_{i,\ell}, v_{i,\ell} \in \mathbb{R}^m$ using the low rank maintenance data structure. Recall the definition of these vectors:

$$\begin{aligned} u_{i,\ell}(t)^\top &= a^\top D_{i,L}(t) W_L(t) \dots D_{i,\ell+1}(t) W_{\ell+1}(t) D_{i,\ell}(t) \in \mathbb{R}^{1 \times m}, \\ v_{i,\ell}(t) &= h_{i,\ell-1}(t) \in \mathbb{R}^m. \end{aligned}$$

Before proceeding, we list the assumptions we will be using:

- For any $\ell \in [L]$, $D_{i,\ell}(t)$ is s_D-sparse, where $s_D := k + s$, k is the sparsity of $D_{i,\ell}(0)$ and s is the sparsity of $D_{i,\ell}(t) - D_{i,\ell}(0)$.
- For any $\ell \in [L]$, the change of the weight matrix W_ℓ, $\Delta W_\ell(t) := W_\ell(t) - W_\ell(0)$, is of low-rank. That is, $\Delta W_\ell(t) = \sum_{j=1}^{r_t} y_{\ell,j} z_{\ell,j}^\top$.
- For any $i \in [n]$, $W_1(0)x_i$ is pre-computed.

We first note that as a direct consequence of $D_{i,\ell}(0)$ is k-sparse, $h_{i,\ell}(0)$ is k-sparse as well. Similarly, $h_{i,\ell}(t) - h_{i,\ell}(0)$ has sparsity s. Hence $h_{i,\ell}(t)$ has sparsity bounded by s_D.

Compute $u_{i,\ell}(t)$. Compute $u_{i,\ell}(t)$ is equivalent to compute the following vector:

$$D_{i,\ell}(t)(W_{\ell+1}(0) + \Delta W_{\ell+1}(t))^\top D_{i,\ell+1}(t) \cdots (W_L(0) + \Delta W_L(t))^\top D_{i,L}(t) a.$$

First, we know that $D_{i,L}(t)a \in \mathbb{R}^m$ is an s_D-sparse vector, and it takes $O(s_D)$ time. The next matrix is $(W_L(0)+\Delta W_L(t))^\top$, which gives two terms: $W_L(0)^\top (D_{i,L}(t)a)$ and $\Delta W_L(t)^\top (D_{i,L}(t)a)$. For the first term, since $D_{i,L}(t)a$ is s_D-sparse, it takes $O(ms_D)$-time. For the second term, we have

$$\begin{aligned} \Delta W_L(t)^\top (D_{i,L}(t)a) &= \sum_{j=1}^{r_t} z_{L,j} y_{L,j}^\top (D_{i,L}(t)a) \\ &= \sum_{j=1}^{r_t} z_{L,j} \cdot \langle y_{L,j}, D_{i,L}(t)a \rangle. \end{aligned}$$

Each inner-product takes $O(s_D)$-time and it takes $O(mr_t + s_D r_t) = O(mr_t)$-time in total. Hence, in $O(m(s_D + r_t))$-time, we compute the vector $W_L(t)^\top D_{i,L}(t)a$. Note that we do not assume the sparsity of a.

Thus, by repeating this process for the $L - \ell$ intermediate matrices $W_j^\top(t) D_{i,j}(t)$, we can obtain the vector

$$\left(\prod_{j=\ell+1}^{L} W_j^\top(t) D_{i,j}(t) \right) a$$

in time $O((L-\ell)m(s_D + r_t))$. Finally, by multiplying a sparse diagonal matrix $D_{i,\ell}(t)$, we get the desired vector $u_{i,\ell}(t)$.

Compute $v_{i,\ell}(t)$. Note that $v_{i,\ell}(t)$ is essentially $h_{i,\ell-1}(t)$, so we consider how to compute $h_{i,\ell}(t)$ for general $\ell \in [L]$. Recall that

$$h_{i,\ell}(t) = \varphi((W_\ell(0) + \Delta W_\ell(t)) h_{i,\ell-1}(t)),$$

since $h_{i,\ell-1}(t)$ is s_D-sparse, the product $W_\ell(0) h_{i,\ell-1}(t)$ can be computed in $O(m s_D)$ time. For the product $\Delta W_\ell(t) h_{i,\ell-1}(t)$ can be computed use the low rank decomposition, which takes $O(m r_t)$ time. Apply the shifted ReLU takes $O(m)$ time. Hence, the total time is $O(m(r_t + s_D))$ time.

The running time results are summarized in the following lemma:

Lemma 4.4.1. *For $\ell \in [L]$ and $i \in [n]$, suppose $\|D_{i,\ell}(0)\|_0 \leq k$. Let $t > 0$. Suppose the change of $D_{i,\ell}$ is sparse, i.e., $\|D_{i,\ell}(t) - D_{i,\ell}(0)\|_0 \leq s$. For $\ell \in [L]$, $i \in [n]$, for any $t > 0$, suppose the change of W_ℓ is of low-rank, i.e., $\Delta W_\ell(t) = \sum_{j=1}^{r_t} y_{\ell,j} z_{\ell,j}^\top$. We further assume that $\{y_{\ell,j}, z_{\ell,j}\}_{\ell \in [L], j \in [r_t]}$ and $\{W_1(0) x_i\}_{i \in [n]}$ are pre-computed.*

Then, for any $\ell \in [L]$ and $i \in [n]$, the vectors $u_{\ell,i}(t), v_{\ell,i}(t) \in \mathbb{R}^m$ can be computed in

$$O(mL(s + k + r_t))$$

time.

As a direct consequence, if we combine Lemma 2.5.2 and Lemma 4.4.1, then we get the following corollary:

Corollary 4.4.2. *For $\ell \in [L]$ and $i \in [n]$, we can compute $v_{i,\ell}(t), u_{i,\ell}(t) \in \mathbb{R}^m$ as in Algorithm 20 with the following time bound:*

- *Compute $u_{i,\ell}(t)$ in time $O(mL(s + k + r_\ell))$.*
- *Compute $v_{i,\ell}(t)$ in time $O(m(s + k + r_\ell))$.*

Remark 4.4.3. *We note that the result we present is more general than needed for our algorithm, since it can handle the updates across all layers. This means we can use it to implement a subquadratic cost per iteration algorithm for gradient descent algorithm over all layers. In this work, we focus our attention to the training of last layer, since the step size of that algorithm is chosen adaptively.*

4.5 Fast Tensor Product Regression

In this section, we show how to solve a specific type of regression fast using both tensor-based sketching matrices for approximation, and sketching-based preconditioner for high precision solution.

Consider the following problem: Given two matrices $U = [u_1^\top, \ldots, u_n^\top]^\top, V = [v_1^\top, \ldots, v_n^\top] \in \mathbb{R}^{m \times n}$ with $m \gg n$, consider the matrix $J \in \mathbb{R}^{n \times m^2}$ formed by

$$J = \begin{bmatrix} \mathrm{vec}(u_1 v_1^\top)^\top \\ \mathrm{vec}(u_2 v_2^\top)^\top \\ \vdots \\ \mathrm{vec}(u_n v_n^\top)^\top \end{bmatrix}.$$

We are also given a vector c in n dimension, our task is to find a solution to the following regression problem:

$$\min_{x \in \mathbb{R}^n} \|JJ^\top x - c\|_2^2.$$

Our main theorem for this section is as follows:

Theorem 4.5.1 (Restatement of Theorem 4.5.14)**.** *Given two $n \times m$ matrices U and V, and a target vector $c \in \mathbb{R}^n$. Let $J = [\mathrm{vec}(u_1 v_1^\top)^\top, \ldots, \mathrm{vec}(u_n v_n^\top)^\top] \in \mathbb{R}^{n \times m^2}$. There is an algorithm (Algorithm 22) takes $\widetilde{O}(nm + n^2(\log(\kappa/\varepsilon) + \log(m/\varepsilon\delta)\varepsilon^{-2}) + n^\omega)$ time and outputs a vector $\widehat{x} \in \mathbb{R}^n$ such that*

$$\|JJ^\top \widehat{x} - c\|_2 \leq \varepsilon \|c\|_2$$

holds with probability at least $1 - \delta$, and κ is the condition number of J.

4.5.1 Approximate J via TensorSketch

We introduce the notion of TensorSketch for two vectors:

Definition 4.5.2. Let $h_1, h_2 : [m] \to [s]$ be 3-wise independent hash functions, also let $\sigma : [m] \to \{\pm 1\}$ be a 4-wise independent random sign function. The degree two TensorSketch transform, $S : \mathbb{R}^m \times \mathbb{R}^m \to \mathbb{R}^s$ is defined as follows: for any $i, j \in [m]$ and $r \in [s]$,

$$S_{r,(i,j)} = \sigma_1(i) \cdot \sigma_2(j) \cdot \mathbf{1}[h_1(i) + h_2(j) = r \bmod s].$$

Remark 4.5.3. *Apply S to two vectors $x, y \in \mathbb{R}^m$ can be implemented in time $O(s \log s + \mathrm{nnz}(x) + \mathrm{nnz}(y))$.*

We introduce one key technical lemma from [11]:

Lemma 4.5.4 (Theorem 1 of [11])**.** *Let $S \in \mathbb{R}^{s \times m^2}$ be the* TensorSketch *matrix, consider a fixed n-dimensional subspace V. If $s = \Omega(n^2/(\varepsilon^2\delta))$, then with probability at least $1 - \delta$, $\|Sx\|_2 = (1 \pm \varepsilon)\|x\|_2$ simultaneously for all $x \in V$.*

Now we are ready to prove the main lemma of this section:

Lemma 4.5.5. *Let $\varepsilon, \delta \in (0, 1)$ denote two parameters. Let $J \in \mathbb{R}^{n \times m^2}$ represent a matrix such that the i-th row of J is equal to $\mathrm{vec}(u_i v_i^\top)$ for some $u_i, v_i \in \mathbb{R}^m$. Then, we can compute a matrix $\widetilde{J} \in \mathbb{R}^{n \times s}$ such that for any vector $x \in \mathbb{R}^n$, with probability at least $1 - \delta$, we have*

$$\|\widetilde{J}^\top x\|_2 = (1 \pm \varepsilon)\|J^\top x\|_2,$$

where $s = \Omega(n^2/(\varepsilon^2\delta))$. The time to compute $\widetilde{J}$ is $O(ns \log s + \mathrm{nnz}(U) + \mathrm{nnz}(V))$.

Proof. Notice that the row space of matrix J can be viewed as an n-dimensional subspace, hence, by Lemma 4.5.4, the TensorSketch matrix S with $s = \Omega(n^2/(\varepsilon^2\delta))$ can preserve the length of all vectors in the subspace generated by $J^\top$ with probability $1-\delta$, to a multiplicative factor of $1 \pm \varepsilon$.

The running time part is to apply the FFT algorithm to each row of J with a total of n rows. For each row, it takes $O(s \log s + m)$ time, hence the overall running time is $O(n(s \log s + m))$. □

4.5.2 Approximate J via TensorSRHT

We note that the dependence on the target dimension of sketching is $O(1/\delta)$ for TensorSketch. We introduce another kind of sketching technique for tensor, called TensorSRHT. The tradeoff is we lose input sparsity runtime of matrices U and V.

Definition 4.5.6. We define the TensorSRHT $S : \mathbb{R}^m \times \mathbb{R}^m \to \mathbb{R}^s$ as $S = \frac{1}{\sqrt{s}} P \cdot (HD_1 \times HD_2)$, where each row of $P \in \{0,1\}^{s \times m^2}$ contains only one 1 at a random coordinate, one can view P as a sampling matrix. H is a $m \times m$ Hadamard matrix, and D_1, D_2 are two $m \times m$ independent diagonal matrices with diagonals that are each independently set to be a Rademacher random variable (uniform in $\{-1, 1\}$).

Remark 4.5.7. *By using FFT algorithm, apply S to two vectors $x, y \in \mathbb{R}^m$ takes time $O(m \log m + s)$.*

We again introduce a technical lemma for TensorSRHT.

Lemma 4.5.8 (Theorem 3 of [2])**.** *Let $S \in \mathbb{R}^{s \times m^2}$ be the* TensorSRHT *matrix, consider a fixed n-dimensional subspace V. If $s = \Omega(n \log^3(nm/\varepsilon\delta)\varepsilon^{-2})$, then with probability at least $1-\delta$, $\|Sx\|_2 = (1 \pm \varepsilon)\|x\|_2$ simultaneously for all $x \in V$.*

Lemma 4.5.9. *Let $\varepsilon, \delta \in (0,1)$ denote two parameters. Given a list of vectors $u_1, \cdots, u_m, v_1, \cdots, v_m \in \mathbb{R}^m$. Let $J \in \mathbb{R}^{n \times m^2}$ represent a matrix where the i-th row of J is equal to $\mathrm{vec}(u_i v_i^\top)$. Then, we can compute a matrix $\widetilde{J} \in \mathbb{R}^{n \times s}$ such that for any vector $x \in \mathbb{R}^n$, with probability at least $1-\delta$, we have*

$$\|\widetilde{J}^\top x\|_2 = (1 \pm \varepsilon)\|J^\top x\|_2,$$

where $s = \Omega(n \log^3(nm/(\varepsilon\delta))\varepsilon^{-2})$. The time to compute $\widetilde{J}$ is $O(n(m \log m + s))$.

Proof. The correctness follows directly from Lemma 4.5.8. The running time follows from the FFT algorithm to each row of J, each application takes $O(m \log m + s)$ time, and we need to apply it to n rows. □

4.5.3 Sketching-based Preconditioner

In this section, we first use TensorSketch and TensorSRHT to approximate J, then use a general sketching matrix as a preconditioner to solve a regression task involving $JJ^\top$.

Before proceeding, we introduce the notion of *subspace embedding*:

Definition 4.5.10 (Subspace Embedding, [67])**.** Let $A \in \mathbb{R}^{N\times k}$, we say a matrix $S \in \mathbb{R}^{s\times N}$ is a $(1\pm\varepsilon)-\ell_2$ subspace embedding for A if for any $x \in \mathbb{R}^k$, we have $\|SAx\|_2^2 = (1\pm\varepsilon)\|Ax\|_2^2$. Equivalently, $\|I - U^\top S^\top SU\| \le \varepsilon$ where U is an orthonormal basis for the column space of A.

We will mainly utilize efficient subspace embedding.

Definition 4.5.11 ([57, 88])**.** Given a matrix $A \in \mathbb{R}^{N\times k}$ with $N = \mathrm{poly}(k)$, then we can compute an $S \in \mathbb{R}^{k\mathrm{poly}(\log(k/\delta))/\varepsilon^2\times N}$ such that with probability at least $1-\delta$, we have

$$\|SAx\|_2 = (1\pm\varepsilon)\|Ax\|_2$$

hols for all $x \in \mathbb{R}^k$. Moreover, SA can be computed in $O(Nk\log((k\log N)/\varepsilon))$ time.

Algorithm 21 Fast Regression algorithm of [17]]

procedure FASTREGRESSION(A, y, ε) ▷ Lemma 4.5.12
▷ $A \in \mathbb{R}^{N\times k}$ is full rank, $\varepsilon \in (0, 1/2)$
    Compute a subspace embedding SA ▷ $S \in \mathbb{R}^{k\mathrm{poly}(\log k)} \times N$
    Compute R such that SAR has orthonormal columns via QR decomposition ▷ $R \in \mathbb{R}^{k\times k}$
    $z_0 = \mathbf{0}_k \in \mathbb{R}^k$
    $t \leftarrow 0$
    while $\|A^\top ARz_t - y\|_2 \ge \varepsilon$ **do**
        $z_{t+1} \leftarrow z_t - (R^\top A^\top AR)^\top (R^\top A^\top ARz_t - R^\top y)$
        $t \leftarrow t+1$
    end while
    return Rz_t
end procedure

Lemma 4.5.12 (Lemma 4.2 of [17])**.** *Let $N = \Omega(k\mathrm{poly}(\log k))$. Given a matrix $A \in \mathbb{R}^{N\times k}$, let κ denote its condition number. Consider the following regression task:*

$$\min_{x\in\mathbb{R}^k} \|A^\top Ax - y\|_2.$$

Using the procedure FASTREGRESSION *(Algorithm 21), with probability at least $1-\delta$, we can compute an ε-approximate solution $\widehat{x}$ satisfying*

$$\|A^\top A\widehat{x} - y\|_2 \le \varepsilon\|y\|_2$$

in time $\widetilde{O}(Nk\log(\kappa/\varepsilon) + k^\omega)$.

Our algorithm is similar to the ridge regression procedure in [73], where they first apply their sketching algorithm as a bootstrapping to reduce the dimension of the original matrix, then use another subspace embedding to proceed and get stronger guarantee.

We shall first prove a useful lemma.

Lemma 4.5.13. *Let $A \in \mathbb{R}^{N\times k}$, suppose SA is a subspace embedding for A (Def. 4.5.11), then we have for any $x \in \mathbb{R}^k$, with probability at least $1-\delta$,*

$$\|(SA)^\top SAx - b\|_2 = (1\pm\varepsilon)\|A^\top Ax - b\|_2.$$

Proof. Throughout the proof, we condition on the event that S preserves the length of all vectors in the column space of A.

Note that

$$\|(SA)^\top SAx - b\|_2^2 = \|(SA)^\top SAx\|_2^2 + \|b\|_2^2 - 2\langle (SA)^\top SAx, b\rangle.$$

We will first bound the norm of $(SA)^\top SAx$, then the inner product term.

Bounding $\|(SA)^\top SAx\|_2^2$.

Let $U \in \mathbb{R}^{N\times k}$ represent an orthonormal basis of A, then use an alternative definition of subspace embedding, we have $\|U^\top S^\top SU - I\| \leq \varepsilon$, this means all the eigenvalues of $U^\top S^\top SU$ lie in the range of of $[(1-\varepsilon)^2, (1+\varepsilon)^2]$. Let V denote the matrix $U^\top S^\top SU$, then we know that all eigenvalues of $V^\top V$ lie in range $[(1-\varepsilon)^4, (1+\varepsilon)^4]$. Setting ε as $\varepsilon/4$, we arrive at $\|V^\top V - I\| \leq \varepsilon$. This shows that for any $x \in \mathbb{R}^k$, we have $\|(SA)^\top SAx\|_2 = (1\pm\varepsilon)\|A^\top Ax\|_2$.

Bounding $\langle (SA)^\top SAx, b\rangle$.

Note that

$$\begin{aligned}\langle (SA)^\top SAx, b\rangle &= \langle SAx, SAb\rangle \\ &= 1/2\cdot(\|SAx\|_2^2 + \|SAb\|_2^2 - \|SA(x-b)\|_2^2) \\ &= 1/2\cdot(1\pm\varepsilon)(\|Ax\|_2^2 + \|Ab\|_2^2 - \|A(x-b)\|_2^2) \\ &= (1\pm\varepsilon)\langle A^\top Ax, b\rangle.\end{aligned}$$

Combining these two terms, we conclude that, with probability at least $1-\delta$,

$$\|(SA)^\top SAx - b\|_2 = (1\pm\varepsilon)\|A^\top Ax - b\|_2.$$

□

Algorithm 22 Fast Regression via tensor trick

procedure FASTTENSORREGRESSION($\{u_i\}_{i=1}^n \in \mathbb{R}^{m\times n}, \{v_i\}_{i=1}^n \in \mathbb{R}^{m\times n}, c \in \mathbb{R}^n$) ▷ Theorem 4.5.14
  ▷ $J = [\mathrm{vec}(u_1 v_1^\top)^\top,\ \mathrm{vec}(u_2 v_2^\top)^\top, \ldots,\ \mathrm{vec}(u_n v_n^\top)^\top]^\top \in \mathbb{R}^{n\times m^2}$
  $s_1 \leftarrow \Theta(n \log^3(nm/\delta))$
  $s_2 \leftarrow \Theta((n + \log m)\log n)$
  Let $S_1 \in \mathbb{R}^{s_1 \times m^2}$ be a sketching matrix ▷ S_1 can be TensorSketch or TensorSRHT
  Compute $\widetilde{J} = JS_1^\top$ via FFT algorithm ▷ $\widetilde{J} \in \mathbb{R}^{n\times s_1}$
  Choose $S_2 \in \mathbb{R}^{s_2 \times s_1}$ to be a sketching matrix (see Def. 4.5.11)
  Compute a subspace embedding $S_2\widetilde{J}^\top$
  Compute R such that $S_2\widetilde{J}^\top R$ has orthonormal columns via QR decomposition ▷ $R \in \mathbb{R}^{n\times n}$
  $z_0 \leftarrow \mathbf{0}_k \in \mathbb{R}^k$
  $t \leftarrow 0$
  while $\|\widetilde{J}\widetilde{J}^\top R z_t - c\|_2 \geq \varepsilon$ **do**
    $z_{t+1} \leftarrow z_t - (R^\top \widetilde{J}\widetilde{J}^\top R)^\top (R^\top \widetilde{J}\widetilde{J}^\top R z_t - R^\top c)$
    $t \leftarrow t + 1$
  end while
  return Rz_t
end procedure

Theorem 4.5.14. *Given two $n \times m$ matrices U and V, and a target vector $c \in \mathbb{R}^n$. Let $J = [\mathrm{vec}(u_1 v_1^\top)^\top, \ldots, \mathrm{vec}(u_n v_n^\top)^\top] \in \mathbb{R}^{n\times m^2}$. There is an algorithm (Algorithm 22) takes $\widetilde{O}(nm + n^2(\log(\kappa/\varepsilon) + \log(m/\delta)) + n^\omega)$ time and outputs a vector $\widehat{x} \in \mathbb{R}^n$ such that*

$$\|JJ^\top \widehat{x} - c\|_2 \leq \varepsilon \|c\|_2$$

holds with probability at least $1 - \delta$, and κ is the condition number of J.

Proof. We can decompose Algorithm 22 into two parts:

- Applying S_1 to efficiently form matrix $\widetilde{J}$ to approximate J and reduce its dimension, notice here we only need ε for this part to be a small constant, pick $\varepsilon = 0.1$ suffices.
- Using S_2 as a preconditioner and solve the regression problem iteratively.

Let $\widehat{x}$ denote the solution found by the iterative regime. We will prove this statement in two-folds:

- First, we will show that $\|\widetilde{J}\widetilde{J}^\top \widehat{x} - c\|_2 \leq \varepsilon \|c\|_2$ with probability at least $1 - \delta$;
- Then, we will show that $\|JJ^\top \widehat{x} - c\|_2 = (1 \pm 0.1)\|\widetilde{J}\widetilde{J}^\top \widehat{x} - c\|_2$ with probability at least $1 - \delta$.

Combining these two statements, we can show that

$$\begin{aligned}\|JJ^\top \widehat{x} - c\|_2 &= (1 \pm 0.1)\|\widetilde{J}\widetilde{J}^\top \widehat{x} - c\|_2 \\ &\leq 1.1\varepsilon \|c\|_2\end{aligned}$$

Setting ε to $\varepsilon/1.1$ and δ to $\delta/2$, we conclude our proof. It remains to prove these two parts.

Part 1 $\|\widetilde{J}\widetilde{J}^\top \widehat{x} - c\|_2 \leq \varepsilon \|c\|_2$**.** We observe the iterative procedure is essentially the same as running FASTREGRESSION on input $\widetilde{J}^\top, y, \varepsilon$, hence by Lemma 4.5.12, we know

$$\Pr\left[\|\widetilde{J}\widetilde{J}^\top \widehat{x} - c\|_2 \leq \varepsilon \|c\|_2\right] \geq 1 - \delta.$$

Part 2 $\|JJ^\top \widehat{x} - c\|_2 = (1 \pm 0.1)\|\widetilde{J}\widetilde{J}^\top \widehat{x} - c\|_2$**.** To prove this part, note that by Lemma 4.5.8, we know that $\widetilde{J}^\top$ is a subspace embedding for $J^\top$. Hence, we can utilize Lemma 4.5.13 and get that,

$$\Pr\left[\|JJ^\top \widehat{x} - c\|_2 = (1 \pm 0.1)\|\widetilde{J}\widetilde{J}^\top \widehat{x} - c\|_2\right] \geq 1 - \delta.$$

Combining these two parts, we have proven the correctness of the theorem. It remains to justify the running time. Note that running time can be decomposed into two parts: 1). The time to generate $\widetilde{J}$, 2). The time to compute $\widehat{x}$ via iterative scheme.

Part 1 Generate $\widetilde{J}$**.** To generate $\widetilde{J}$, we apply $S_1 \in \mathbb{R}^{s_1 \times m^2}$ which is a TensorSRHT. By Lemma 4.5.9, it takes $O(n(m \log m + s_1))$ time to compute $\widetilde{J}$, plug in $s_1 = \Theta(n \log^3(nm/\delta))$, the time is $\widetilde{O}(nm)$.

Part 2 Compute $\widehat{x}$**.** To compute $\widehat{x}$, essentially we run FASTREGRESSION on $\widetilde{J}^\top, c, \varepsilon$, hence by Lemma 4.5.12, it takes $\widetilde{O}(s_2 n \log(\kappa/\varepsilon) + n^\omega)$ time, with $s_2 = \Theta((n + \log m)\log n)$ and κ is the condition number of $\widetilde{J}$, which has the guarantee $\kappa = (1 \pm \varepsilon)\kappa(J)$. Hence, the overall running time of this part is $\widetilde{O}(n^2 \log(\kappa/\varepsilon) + n^\omega)$.

Put things together, the overall running time is $\widetilde{O}(nm + n^2 \log(\kappa/\varepsilon) + n^\omega)$. □

Remark 4.5.15. *Due to the probability requirement (union bounding over all data points), here we only prove by using* TensorSRHT*. One can use similar strategy to obtain an input sparsity time version using* TensorSketch*. We remark that this framework is similar to the approach [73] takes to solve kernel ridge regression, where one first uses a shallow but fast sketch to bootstrap, then use another sketching to proceed with the main task.*

We also point out that in a more general neural network architecture, the network width between different layers might differ, i.e., we are in fact dealing with the tensor product $u \otimes v$ where $u \in \mathbb{R}^{m_\ell}$ and $v \in \mathbb{R}^{m_{\ell-1}}$. Our sketching can be modified to handle this kind of inputs. Specifically, for TensorSketch*, it is defined by a pair of hash functions and sign functions, we can change their domain to handle different input dimensions. For* TensorSRHT*, it's more tricky, however, we note that the Hadamard matrix is merely for speeding up computation via FFT algorithm, hence we can differ the size of D_1 and D_2, and change the size of sampling matrix P accordingly.*

4.6 Spectral Properties of Over-parametrized Deep Neural Network

In this section, we study the spectral structure of our Gram matrix and its connection to the multi-layer NTK matrix. We show two different results regarding the minimum eigenvalue of the last layer and the intermediate layers. We also show that as long as the weight matrix does not move too far, the eigenvalue of the Gram matrix is relatively stable around its initialization.

4.6.1 Bounds on the Least Eigenvalue of Kernel at Initialization

The following fact gives the exact form of the Gram matrix of the ℓ-th layer of the neural network.

Fact 4.6.1 (Multi-layer Gram matrix)**.** *For any $\ell \in [L]$, let $G_\ell = J_\ell J_\ell^\top \in \mathbb{R}^{n \times n}$ be the layer-ℓ's Gram matrix. Then, for any $i, j \in [n]$,*

$$(G_\ell)_{i,j} = h_{i,\ell-1}^\top h_{j,\ell-1} \cdot a^\top \left(D_{i,\ell} \prod_{k=\ell+1}^{L} W_k^\top D_{i,k} \right)^\top \left(D_{j,\ell} \prod_{k=\ell+1}^{L} W_k^\top D_{j,k} \right) a,$$

where $h_{i,\ell-1} = \prod_{k=1}^{\ell-1} D_{i,k} W_k x_i$.

Proof. For for any $i, j \in [n]$, by definition,

$$\begin{aligned}
(G_\ell)_{i,j} = & \operatorname{vec}(\frac{\partial f(W, x_i)}{\partial W_\ell})^\top \operatorname{vec}(\frac{\partial f(W, x_j)}{\partial W_\ell}) \\
= & \operatorname{vec}\left(D_{i,\ell} \prod_{k=\ell+1}^{L} W_k^\top D_{i,k} a h_{i,\ell-1}^\top \right)^\top \operatorname{vec}\left(D_{j,\ell} \prod_{k=\ell+1}^{L} W_k^\top D_{j,k} a h_{j,\ell-1}^\top \right) \\
= & \left((h_{i,\ell-1} \otimes I_{m_\ell}) \left(D_{i,\ell} \prod_{k=\ell+1}^{L} W_k^\top D_{i,k} a \right) \right)^\top (h_{j,\ell-1} \otimes I_{m_\ell}) \left(D_{j,\ell} \prod_{k=\ell+1}^{L} W_k^\top D_{j,k} a \right) \\
= & \left(D_{i,\ell} \prod_{k=\ell+1}^{L} W_k^\top D_{i,k} a \right)^\top (h_{i,\ell-1}^\top \otimes I_{m_\ell})(h_{j,\ell-1} \otimes I_{m_\ell}) \left(D_{j,\ell} \prod_{k=\ell+1}^{L} W_k^\top D_{j,k} a \right) \\
= & a^\top \left(D_{i,\ell} \prod_{k=\ell+1}^{L} W_k^\top D_{i,k} \right)^\top \left(D_{j,\ell} \prod_{k=\ell+1}^{L} W_k^\top D_{j,k} \right) a \cdot h_{i,\ell-1}^\top h_{j,\ell-1}.
\end{aligned}$$

□

The following lemma handles the least eigenvalue for all intermediate layers $\ell \in [L-1]$.

Lemma 4.6.2 (Bounds on the least eigenvalue at initialization for layer 1 to $L-1$)**.** *Let $\lambda := \min_{\ell \in [L-1]} \lambda_\ell$. Then, for all $\ell \in [L-1]$, with probability at least $1 - \delta$, we have*

$$\lambda_{\min}(G_\ell) \geq \Omega(\lambda \delta^2 n^{-2} L^{-1}).$$

Proof. Let $\ell \in [L]$ and $i, j \in [n]$. By Fact 4.6.1, the (i,j)-th entry of the layer-ℓ Gram matrix can be expressed as

$$\begin{aligned}
(G_\ell)_{i,j} = & h_{i,\ell-1}^\top h_{j,\ell-1} \cdot a^\top \left(D_{i,\ell} \prod_{k=\ell+1}^{L} W_k^\top D_{i,k} \right)^\top \left(D_{j,\ell} \prod_{k=\ell+1}^{L} W_k^\top D_{j,k} \right) a \\
= & (H_{\ell-1})_{i,j} \cdot (A_\ell)_{i,j},
\end{aligned}$$

where $(H_{\ell-1})_{i,j} = h_{i,\ell-1}^\top h_{j,\ell-1}$ and $(A_\ell)_{i,j} = a^\top (D_{i,\ell} \prod_{k=\ell+1}^{L} W_k^\top D_{i,k})^\top (D_{j,\ell} \prod_{k=\ell+1}^{L} W_k^\top D_{j,k}) a$. Hence, we can write G_ℓ as

$$G_\ell = H_{\ell-1} \odot A_\ell,$$

where $\odot$ represents the Hadamard product.

By Fact 4.2.7, we have

$$\lambda_{\min}(G_\ell) \geq \min_{i\in[n]} (A_\ell)_{i,i} \cdot \lambda_{\min}(H_\ell)$$

Note that for $i \in [n]$,

$$\begin{aligned}
(A_\ell)_{i,i} &= a^\top (D_{i,\ell} \prod_{k=\ell+1}^{L} W_k^\top D_{i,k})^\top (D_{i,\ell} \prod_{k=\ell+1}^{L} W_k^\top D_{i,k}) a \\
&= \Big\| D_{i,\ell} \prod_{k=\ell+1}^{L} W_k^\top D_{i,k} a \Big\|_2^2 \\
&\geq \frac{\Big\langle W_\ell \prod_{k=1}^{\ell-1} D_{i,k} W_k x_i, D_{i,\ell} \prod_{k=\ell+1}^{L} W_k^\top D_{i,k} a \Big\rangle^2}{\|W_\ell \prod_{k=1}^{\ell-1} D_{i,k} W_k x_i\|_2^2} \\
&= \frac{\langle a, h_{i,L}\rangle^2}{\|W_\ell \prod_{k=1}^{\ell-1} D_{i,k} W_k x_i\|_2^2}.
\end{aligned}$$

By Lemma 4.7.9 part (a), we have

$$\Big\| W_\ell \prod_{k=1}^{\ell-1} D_{i,k} W_k x_i \Big\|_2^2 \leq O(\sqrt{L})$$

with probability $1 - e^{-\Omega(k/L^2)}$ for all $i \in [n]$, where $k = m\exp(-b^2 m/4) = m^{0.8}$ by our choice of parameters.

By Lemma 4.7.8,

$$\Pr[\forall i \in [n], \quad \|h_{i,L}\|_2 \in 1 \pm \varepsilon] \geq 1 - O(nL) \cdot \exp(-\Omega(k\varepsilon^2/L^2)).$$

Conditioning on this event, let $h_{i,L}$ be a fixed vector h with length $1 \pm \varepsilon$ and consider the randomness of the Rademacher vector a.

Note that $\langle a, h\rangle$ can be written as $a^\top (hh^\top) a = a^\top B a$, where $B := hh^\top$ satisfies $\|B\| = \|h\|_2^2$ and $\|B\|_{\mathrm{HS}}^2 = \|h\|_2^4$. For $r \in [m_L]$, we know that a_r is a centered subgaussian random variable with $\|a_r\|_{\psi_2} = 1$.

By Lemma 4.2.6, we have

$$\Pr[|\langle a, h\rangle| \leq t] \leq \min\left\{ \frac{C_1 t}{\|h\|_2}, \frac{C_2 t}{\widetilde{O}(m^{-0.2})\sqrt{k}} \right\} \leq \widetilde{O}(t).$$

By taking t to be $O(\delta/n)$, we have

$$\Pr[\forall i \in [n],\ \langle a, h_{i,L}\rangle^2 = \Omega(\delta^2/n^2)] \geq 1 - O(\delta).$$

Applying a union bound gives

$$\Pr[\min_{i\in[n]} (A_\ell)_{i,i} = \Omega(\delta^2 n^{-2} L^{-1})] \geq 1 - \delta/2.$$

By Lemma 4.6.4, with probability at least $1 - \delta/2$, we have:

$$\lambda_{\min}(H_{\ell-1}) \geq \Omega(\lambda)$$

for all $\ell \in [L]$.

Combine them together, we get that

$$\lambda_{\min}(G_\ell) \geq \Omega(\delta^2 \lambda n^{-2} L^{-1})$$

for all $\ell \in [L]$ with probability $1 - \delta$, which completes the proof of the lemma. □

In order to bound $\lambda_{\min}(H_\ell)$, we first define the NTK kernel for multiple layer neural network.

Definition 4.6.3 (Multiple layer NTK kernel)**.** The NTK kernel $\mathbf{K}_\ell \in \mathbb{R}^{n \times n}$ for $\ell \in \{0, \ldots, L\}$ of an L-layer neural network are defined as follows:

- $(\mathbf{K}_0)_{i,j} := x_i^\top x_j$
- For $\ell > 0$, let $\Sigma_{\ell,i,j} := \begin{bmatrix} (\mathbf{K}_{\ell-1})_{i,i} & (\mathbf{K}_{\ell-1})_{i,j} \\ (\mathbf{K}_{\ell-1})_{j,i} & (\mathbf{K}_{\ell-1})_{j,j} \end{bmatrix} \in \mathbb{R}^{2\times 2}$ for any $(i,j) \in [n] \times [n]$. Then,

$$\begin{aligned} (\mathbf{K}_\ell)_{i,j} &:= \mathbb{E}_{(x_1,x_2)\sim\mathcal{N}(\mathbf{0},2\Sigma_{\ell-1,i,j})}[\varphi(x_1)\varphi(x_2)] \quad \forall \ell \in [L-1], \\ (\mathbf{K}_L)_{i,j} &:= \mathbb{E}_{(x_1,x_2)\sim\mathcal{N}(\mathbf{0},2\Sigma_{L-1,i,j})}[\varphi'(x_1)\varphi'(x_2)] \end{aligned}$$

Let $\lambda_\ell := \lambda_{\min}(\mathbf{K}_\ell)$ to be the minimum eigenvalue of the NTK kernel $\mathbf{K}_\ell$.

In the following lemma, we generalize Lemma C.3 in [17] (also Lemma 3 in [18]) into multiple layer neural networks.

Lemma 4.6.4. *For $\ell \in [L-1]$, let λ_ℓ denote the minimum eigenvalue of NTK defined for ℓ-th layer of neural networks. Suppose $m_\ell = \Omega(\lambda_\ell^{-2} n^2 \log(n/\delta))$, then with probability at least $1-\delta$, we have*

$$\lambda_{\min}(H_\ell) \geq \frac{3}{4}\lambda_\ell, \quad \forall \ell \in [L]$$

Proof. We will prove that $\|H_\ell - \mathbf{K}_\ell\|_\infty$ is small, which implies that $\lambda_{\min}(H_\ell)$ is close to λ_ℓ. The proof idea is similar to [31] via induction on ℓ.

For $\ell = 1$, recall $(g_{1,i})_k = \sum_{b\in[m]} (W_1)_{k,b}(x_i)_b$ for $k \in [m]$. Hence, for any $k \in [m]$,

$$\begin{aligned} \mathbb{E}[(g_{i,1})_k (g_{j,1})_k] &= \sum_{b,b'\in[m]} \mathbb{E}[(W_1)_{k,b}(W_1)_{k,b'}(x_i)_b (x_j)_{b'}] \\ &= \sum_{b\in[m]} \mathbb{E}[(W_1)_{k,b}^2] \cdot (x_i)_b (x_j)_b \qquad ((W_1)_{k,b} \sim \mathcal{N}(0, \tfrac{2}{m}).) \\ &= \frac{2}{m} \sum_{b\in[m]} (x_i)_b (x_j)_b \\ &= \frac{2}{m} x_i^\top x_j. \end{aligned}$$

Then, we have

$$\begin{aligned}
\mathbb{E}[h_{i,1}^\top h_{j,1}] &= \sum_{k\in[m]} \mathbb{E}[(h_{i,1})_k (h_{j,1})_k] \\
&= \sum_{k\in[m]} \mathbb{E}[\varphi((g_{i,1})_k)\varphi((g_{j,1})_k)] \\
&= \sum_{k\in[m]} \mathbb{E}_{(u,v)\sim\mathcal{N}(0,\frac{2}{m}\Sigma_{1,i,j})}[\varphi(u)\varphi(v)] \\
&= \mathbb{E}_{(u,v)\sim\mathcal{N}(0,\frac{2}{m}\Sigma_{1,i,j})}[m\varphi(u)\varphi(v)] \\
&= \mathbb{E}_{(u',v')\sim\mathcal{N}(0,2\Sigma_{1,i,j})}[\varphi(u')\varphi(v')] \\
&= (\mathbf{K}_1)_{i,j}.
\end{aligned}$$

Next, we will show that $h_{i,1}^\top h_{j,1}$ concentrates around its expectation. First, for any $k \in [m]$,

$$|(h_{i,1})_k (h_{j,1})_k| \le |(g_{i,1})_k (g_{j,1})_k| \le |\langle (W_1)_{k,*}, x_i\rangle| \cdot |\langle (W_1)_{k,*}, x_j\rangle|.$$

Since $\langle (W_1)_{k,*}, x_i\rangle \sim \mathcal{N}(0, \frac{2\|x_i\|_2^2}{m})$, by the concentration of Gaussian distribution,

$$|\langle (W_1)_{k,*}, x_i\rangle| \le \sqrt{c} \quad \forall k \in [m], i \in [n]$$

holds with probability at least $1 - mne^{-cm/4}$.

Conditioning on the above event, we have $|(h_{i,1})_k (h_{j,1})_k| \le c$ for all $i, j \in [n]$ and $k \in [m]$. Then, by Hoeffding's inequality, we have for any $(i, j) \in [n] \times [n]$,

$$\begin{aligned}
\Pr\left[|h_{i,1}^\top h_{j,1} - (\mathbf{K}_1)_{i,j}| \ge t\right] &\le \exp\left(-\frac{t^2}{2m\cdot(2c)^2}\right) \\
&= \exp(-\Omega(t^2/(mc^2))).
\end{aligned}$$

Hence, by union bound, we get that

$$\Pr[\max_{(i,j)\in[n]\times[n]} |h_{i,1}^\top h_{j,1} - (\mathbf{K}_1)_{i,j}| \le t] \ge 1 - mn\cdot\exp(-\Omega(mc)) - n^2\cdot\exp(-\Omega(t^2/(mc^2))).$$

If we choose $c := \frac{\log(mnL/\delta)}{m}$ and $t := m^{-1/2}\cdot \mathrm{polylog}(nL/\delta)$, we have with probability at least $1 - \frac{\delta}{L}$,

$$\max_{(i,j)\in[n]\times[n]} |h_{i,1}^\top h_{j,1} - (\mathbf{K}_1)_{i,j}| \le \widetilde{O}(m^{-1/2}).$$

Let $h < L$. Suppose that for $\ell = 1, \dots h$,

$$\max_{(i,j)\in[n]\times[n]} |h_{i,\ell}^\top h_{j,\ell} - (\mathbf{K}_\ell)_{i,j}| \le \widetilde{O}(m^{-1/2}).$$

Consider $\ell = h + 1$. By a similar computation, we have

$$\mathbb{E}_{W_\ell}[(g_{i,\ell})_k (g_{j,\ell})_k] = \frac{2}{m} h_{i,\ell-1}^\top h_{j,\ell-1}.$$

Define a new covariance matrix

$$\widehat{\Sigma}_{\ell,i,j} := \begin{bmatrix} h_{i,\ell-1}^\top h_{i,\ell-1} & h_{i,\ell-1}^\top h_{j,\ell-1} \\ h_{j,\ell-1}^\top h_{i,\ell-1} & h_{j,\ell-1}^\top h_{j,\ell-1} \end{bmatrix} \quad \forall (i,j) \in [n] \times [n].$$

We have

$$\begin{aligned} \mathbb{E}_{W_\ell}[h_{i,\ell}^\top h_{j,\ell}] &= \sum_{k \in [m]} \mathbb{E}_{(u,v) \sim \mathcal{N}(0, \frac{2}{m}\widehat{\Sigma}_{\ell,i,j})}[\varphi(u)\varphi(v)] \\ &= \mathbb{E}_{(u',v') \sim \mathcal{N}(0, 2\widehat{\Sigma}_{\ell,i,j})}[\varphi(u')\varphi(v')] \\ &:= (\widehat{\mathbf{K}}_\ell)_{i,j}. \end{aligned}$$

Hence, we have with probability at least $1 - \frac{\delta}{L}$,

$$\max_{(i,j) \in [n] \times [n]} \left| h_{i,\ell}^\top h_{j,\ell} - (\widehat{\mathbf{K}}_\ell)_{i,j} \right| \leq \widetilde{O}(m^{-1/2}). \tag{4.4}$$

It remains to upper bound the difference $\|\widehat{\mathbf{K}}_\ell - \mathbf{K}_\ell\|_\infty$.

$$\left\| \widehat{\mathbf{K}}_\ell - \mathbf{K}_\ell \right\|_\infty = \max_{(i,j) \in [n] \times [n]} \left| \mathbb{E}_{(u,v) \sim \mathcal{N}(0, 2\widehat{\Sigma}_{\ell,i,j})}[\varphi(u)\varphi(v)] - \mathbb{E}_{(u,v) \sim \mathcal{N}(0, 2\Sigma_{\ell,i,j})}[\varphi(u)\varphi(v)] \right|.$$

Recall that

$$\Sigma_{\ell,i,j} := \begin{bmatrix} (\mathbf{K}_{\ell-1})_{i,i} & (\mathbf{K}_{\ell-1})_{i,j} \\ (\mathbf{K}_{\ell-1})_{j,i} & (\mathbf{K}_{\ell-1})_{j,j} \end{bmatrix} \quad \forall (i,j) \in [n] \times [n],$$

and hence, by the induction hypothesis, we have

$$\|\widehat{\Sigma}_{\ell,i,j} - \Sigma_{\ell,i,j}\|_\infty \leq \max_{(i,j) \in [n] \times [n]} \left| h_{i,\ell-1}^\top h_{j,\ell-1} - (\mathbf{K}_{\ell-1})_{i,j} \right| = \widetilde{O}(m^{-1/2}).$$

Notice that $\widehat{\Sigma}_{\ell,i,j}$ can be written as

$$\begin{bmatrix} \|h_{i,\ell-1}\|_2^2 & \cos(\theta_{\ell,i,j})\|h_{i,\ell-1}\|_2\|h_{j,\ell-1}\|_2 \\ \cos(\theta_{\ell,i,j})\|h_{i,\ell-1}\|_2\|h_{j,\ell-1}\|_2 & \|h_{j,\ell-1}\|_2^2 \end{bmatrix}.$$

Moreover, when φ is the ReLU function, we have

$$\mathbb{E}_{(u,v) \sim \mathcal{N}(0, 2\widehat{\Sigma}_{\ell,i,j})}[\varphi(u)\varphi(v)] = 2\|h_{i,\ell-1}\|_2\|h_{j,\ell-1}\|_2 \cdot F(\theta_{\ell,i,j}),$$

where

$$F(\theta) := \mathbb{E}_{(u,v) \sim \mathcal{N}(0, \Sigma(\theta))}[\varphi(u)\varphi(v)] \quad \text{with} \quad \Sigma(\theta) := \begin{bmatrix} 1 & \cos(\theta) \\ \cos(\theta) & 1 \end{bmatrix}.$$

We note that $F(\theta)$ has the following analytic form:

$$F(\theta) = \frac{1}{2\pi}(\sin(\theta) + (\pi - \theta)\cos(\theta)) \in [0, 1/2]. \tag{4.5}$$

Similarly,

$$\mathbb{E}_{(u,v)\sim\mathcal{N}(0,2\Sigma_{\ell,i,j})}[\varphi(u)\varphi(v)] = 2\sqrt{(\mathbf{K}_{\ell-1})_{i,i}(\mathbf{K}_{\ell-1})_{j,j}}\cdot F(\tau_{\ell,i,j}),$$

where $\tau_{\ell,i,j} := \cos^{-1}\left(\frac{(\mathbf{K}_{\ell-1})_{i,j}}{\sqrt{(\mathbf{K}_{\ell-1})_{i,i}(\mathbf{K}_{\ell-1})_{j,j}}}\right)$. By the induction hypothesis, we have $(\mathbf{K}_\ell)_{i,j} \in h_{\ell,i}^\top h_{\ell,j} \pm \widetilde{O}(m^{-1/2})$ for all $i,j \in [n]$. By Lemma 4.7.8, we also have $\|h_{\ell,i}\|_2 \in 1\pm\varepsilon$ for all $\ell \in [L]$ and $i \in [n]$ with probability $1 - O(nL)\cdot e^{-\Omega(m\varepsilon^2/L)}$. They implies that $\cos(\tau_{\ell,i,j}) \in \cos(\theta) \pm \widetilde{O}(m^{-1/2})$. Thus, by Taylor's theorem, it gives us

$$|F(\theta_{\ell,i,j}) - F(\tau_{\ell,i,j})| \le \widetilde{O}(m^{-1/2}).$$

Therefore, we have

$$\begin{aligned}
&\left|\mathbb{E}_{(u,v)\sim\mathcal{N}(0,2\widehat{\Sigma}_{\ell,i,j})}[\varphi(u)\varphi(v)] - \mathbb{E}_{(u,v)\sim\mathcal{N}(0,2\Sigma_{\ell,i,j})}[\varphi(u)\varphi(v)]\right| \\
&= 2\left|\|h_{i,\ell-1}\|_2\|h_{j,\ell-1}\|_2 F(\theta_{\ell,i,j}) - \sqrt{(\mathbf{K}_{\ell-1})_{i,i}(\mathbf{K}_{\ell-1})_{j,j}}F(\tau_{\ell,i,j})\right| \\
&\le \widetilde{O}(m^{-1/2}).
\end{aligned}$$

That is,

$$\|\widehat{\mathbf{K}_\ell} - \mathbf{K}_\ell\|_\infty \le \widetilde{O}(m^{-1/2}). \tag{4.6}$$

Combining Eqs. (4.4) and (4.6) together, we get that

$$\max_{(i,j)\in[n]\times[n]} |h_{\ell,i}^\top h_{\ell,j} - (\mathbf{K}_\ell)_{i,j}| \le \widetilde{O}(m^{-1/2})$$

holds with probability at least $1 - \frac{\delta}{L}$ for $\ell = h+1$.

By induction, we have proved that for the first $L-1$ layers, the intermediate correlation $h_{\ell,i}^\top h_{\ell,j}$ is close to the intermediate Gram matrix $(\mathbf{K}_\ell)_{i,j}$, i.e.,

$$\|H_\ell - K_\ell\| \le \frac{\lambda_\ell}{4} \quad \forall \ell \in [L-1].$$

Hence, we get that for all $\ell \in [L-1]$,

$$\lambda_{\min}(H_\ell) \ge \frac{3}{4}\lambda_\ell$$

The lemma is then proved. □

Lemma 4.6.5 (Bounds on the least eigenvalue at initialization for layer L)**.** *Suppose* $m = \Omega(\lambda_L^{-2} n^2 \log(n/\delta))$*, then we have*

$$\Pr[\lambda_{\min}(G_L) \ge \frac{3}{4}\lambda_L] \ge 1 - \delta.$$

Proof. Recall G_L is defined as

$$\begin{aligned}(G_L)_{i,j} &= \mathrm{vec}(\frac{\partial f(W,x_i)}{\partial W_L})^\top \mathrm{vec}(\frac{\partial f(W,x_j)}{\partial W_L}) \\ &= \mathrm{vec}(D_{i,L} a h_{i,L-1}^\top)^\top \mathrm{vec}(D_{j,L} a h_{j,L-1}^\top) \\ &= a^\top D_{i,L} D_{j,L} a \cdot h_{i,L-1}^\top h_{j,L-1},\end{aligned}$$

which has the same form as the correlation matrix of a two-layer over-parameterized neural network with input data $\{h_{L-1,i}\}_{i\in[n]}$. Define

$$(\widehat{\mathbf{K}}_L)_{i,j} := h_{i,L-1}^\top h_{j,L-1} \cdot \mathbb{E}_{w\sim\mathcal{N}(0,2I_m)} \left[\varphi'(w^\top h_{i,L-1})\varphi'(w^\top h_{j,L-1})\right].$$

Then, by the analysis of the two-layer case (see for example [32, 71]), we have

$$\|G_L - \widehat{\mathbf{K}}_L\| \le \frac{\lambda_L}{8},$$

if $m = \Omega(\lambda_L^{-2} n^2 \log(n/\delta))$, where $\lambda_L := \lambda_{\min}(\mathbf{K}_L)$. It remains to bound $\|\widehat{\mathbf{K}}_L - \mathbf{K}_L\|_\infty$. Equivalently, for any $(i,j) \in [n]\times[n]$,

$$\max_{(i,j)\in[n]\times[n]} \left|\mathbb{E}_{(u,v)\sim\mathcal{N}(0,2\widehat{\Sigma}_{L,i,j})}[\varphi'(u)\varphi'(v)] - \mathbb{E}_{(u,v)\sim\mathcal{N}(0,2\Sigma_{L,i,j})}[\varphi'(u)\varphi'(v)]\right|.$$

The expectation has the following analytic form:

$$\mathbb{E}_{(z_1,z_2)\sim\mathcal{N}(0,\Sigma)}[\varphi'(z_1)\varphi'(z_2)] = \frac{1}{4} + \frac{\sin^{-1}(\rho)}{2\pi} \quad \text{with} \quad \Sigma = \begin{bmatrix} p^2 & \rho pq \\ \rho pq & q^2 \end{bmatrix}.$$

By the analysis of the $(L-1)$-layer, we know that $|\rho_{L,i,j} - \widehat{\rho}_{L,i,j}| \le \widetilde{O}(m^{-1/2})$, where $\rho_{L,i,j} := \cos(\tau_{L,i,j})$ and $\widehat{\rho}_{L,i,j} := \cos(\theta_{L,i,j})$. Also, notice that $\cos(\tau_{L,i,j}) = F(\tau_{L-1,i,j}) \in [0, 1/2]$ by Eq. (4.5). Hence, the derivative of the expectation is bounded, and by Taylor's theorem, we have

$$\|\widehat{\mathbf{K}}_L - \mathbf{K}_L\|_\infty \le \widetilde{O}(m^{-1/2}).$$

It implies that $\|\widehat{\mathbf{K}}_L - \mathbf{K}_L\| \le \frac{\lambda_L}{8}$, which further implies that

$$\|G_L - \mathbf{K}_L\| \le \frac{\lambda_L}{4}.$$

Equivalently, we get that

$$\lambda_{\min}(G_L) \ge \frac{3}{4}\lambda_L$$

with probability at least $1-\delta$. □

Remark 4.6.6. *We observe a discrepancy of eigenvalue in our analysis: For last layer, the eigenvalue of our Gram matrix and the NTK is almost the same, while for intermediate layers, we can only provide a much weaker lower bound for Gram matrix. The main reason is by definition, the NTK for last layer is defined as the product of two derivatives of ReLU, which always have value 0 or 1. On the other hand, the NTKs for intermediate layers are defined in terms of the product of two ReLU's, which can have much larger magnitudes.*

Due to such eigenvalue discrepancy, our algorithm focuses on training the last layer*, since the training dynamic on intermediate layers has a much smaller magnitude. Hence, we present an algorithm that only trains the last layer while obtaining a good convergence result.*

4.6.2 Bounds on the Least Eigenvalue during Optimization

In this section, we adapt the Lemma C.5 in [17] into the last layer of a multi-layer neural network. We make use of the result proved in [75].

Lemma 4.6.7 (Lemma C.2 in [75]). *Let $b > 0$ and $\widetilde{R} \leq 1/b$. Let $c > 0$ and $c' > 0$ denote two fixed constants. Suppose we have*

$$\|W_L - W_L(0)\| \leq \widetilde{R},$$

then we have

- $\|G_L(W) - G_L(W(0))\|_F \leq n\alpha$ *holds with probability at least* $1 - n^2\beta$.
- $\lambda_{\min}(G_L(W)) \geq \frac{3}{4}\lambda_L - n\alpha$ *holds with probability at least* $1 - n^2\beta - \delta$,

where $\alpha = \min\{c \cdot \exp(-b^2/2), 3\widetilde{R}\}$ *and* $\beta = \exp(-m \cdot \min\{c' \cdot \exp(-b^2/2), \widetilde{R}/10\})$.

Corollary 4.6.8. *Suppose we have*

- $\alpha = 3\widetilde{R}$ *and* $\widetilde{R} \leq O(\frac{\lambda_L}{n})$.
- $\alpha = c \cdot \exp(-b^2/2)$ *and* $\exp(-b^2/2) \leq O(\frac{\lambda_L}{n})$.

then we have $\lambda_{\min}(G_L(W)) \geq \frac{\lambda_L}{2}$.

Proof. We first note that to prove the corollary, it suffices to show that $n\alpha \leq \frac{\lambda_L}{4}$. We analyze two cases.

Case 1: $\alpha = 3R$**.** Suppose $\alpha = 3R$, then the condition translates to $3n\widetilde{R} \leq \frac{\lambda_L}{4}$ which indicates $\widetilde{R} \leq O(\frac{\lambda_L}{n})$.

Case 2: $\alpha = c \cdot \exp(-b^2/2)$**.** Suppose $\alpha = c \cdot \exp(-b^2/2)$, then we have $cn \cdot \exp(-b^2/2) \leq \frac{\lambda_L}{4}$ and $\exp(-b^2/2) \leq O(\frac{\lambda_L}{n})$. □

Remark 4.6.9. *We note that the analysis of [75] focuses on the standard two-layer case of NTK, the reason we can leverage their result is that we can treat the NTK for last layer as a two-layer neural network where the inputs are $h_{i,L-1} \in \mathbb{R}^m$. One can also give include a direct proof of the multi-layer version, which agrees the above lemma and corollary.*

4.7 Convergence Analysis of Our Algorithm

In this section, we present a convergence analysis of Algorithm 20. We show that as long as the neural network width is large enough, the convergence of Algorithm 20 is linear, and the weight matrix does not change too much.

4.7.1 Preliminary

We recall the initialization of our neural network.

Definition 4.7.1 (Initialization)**.** Let $m = m_\ell$ for all $\ell \in [L]$. Let $m_0 = d$. We assume weights are initialized as

- Each entry of weight vector $a \in \mathbb{R}^m$ is i.i.d. sampled from $\{-1, +1\}$ uniformly at random.
- Each entry of weight matrices $W_\ell \in \mathbb{R}^{m \times m}$ sampled from $\mathcal{N}(0, 2/m)$.

Remark 4.7.2. *Later, we will also interpret W_L as sampled from $\mathcal{N}(0,1)$ and then being scaled by $\sqrt{\frac{2}{m}}$.*

We also restate the architecture of our neural network here.

Definition 4.7.3 (Architecture)**.** Our neural network is a standard L-layer feed-forward neural network, with the activation functions defined as a scaled version of shifted ReLU activation: $\varphi(x) = \sqrt{c_b}\mathbf{1}[x > \sqrt{2/m}b]x$, where $c_b := (2(1 - \Phi(b) + b\varphi(b)))^{-1/2}$. Here b is a threshold value we will pick later. At last layer, we use a scaled version of a vector with its entry being Rademacher random variables. We define the neural network function $f : \mathbb{R}^{m_0} \to \mathbb{R}$ as

$$f(W, x_i) = a^\top \varphi(W_L \varphi(W_{L-1}\varphi(\dots \varphi(W_1 x_i)))).$$

We measure the loss of the neural network via squared-loss function:

$$\mathcal{L}(W) = \frac{1}{2}\sum_{i=1}^{n}(f(x_i) - y_i)^2.$$

We use $f_t : \mathbb{R}^{d\times n} \to \mathbb{R}^n$ denote the prediction of our network:

$$f_t(X) = [f(W(t), x_1), \dots, f(W(t), x_n)]^\top.$$

We state two assumptions here.

Assumption 4.7.4 (Small Row Norm)**.** *Let $t \in \{0, \dots, T\}$ and let $\widetilde{R} \leq 1$ be a parameter. We assume*

$$\|W_{L,r}(t) - W_{L,r}(0)\|_2 \leq \widetilde{R}/\sqrt{m}, \quad \forall r \in [m].$$

Here, $W_{L,r} \in \mathbb{R}^m$ means the r-th row of matrix W_L.

Later, we will invoke this assumption by specifying the choice of $\widetilde{R}$.

Assumption 4.7.5 (Sparsity)**.** *Let $t \in \{0, \dots, T\}$ and let $s \geq 1$ be an integer parameter. We assume*

$$\|\Delta D_{i,\ell}\|_0 \leq s, \forall \ell \in [L], i \in [n].$$

Later, we will invoke this assumption by specifying the choice of s.

4.7.2 Technical Lemmas

We first show that during initialization, by using our shifted ReLU activation, the vector $h_{i,\ell}$ is sparse. Hence, the diagonal matrix $D_{i,\ell}$ is sparse as well.

Lemma 4.7.6 (Sparse initialization)**.** *Let $\sigma_b(x) = \max\{x - b, 0\}$ be the shifted ReLU activation with threshold $b > 0$. After initialization, with probability*

$$1 - nL \cdot e^{-\Omega(me^{-b^2 m/4})},$$

it holds for all $i \in [n]$ and $\ell \in [L]$,

$$\|h_{i,\ell}\|_0 \leq O(m \cdot e^{-b^2 m/4}).$$

Proof. We fix $i \in [n]$ and $\ell \in [L]$, since we will union bound over all i and ℓ at last. Let $u_i \in \mathbb{R}^m$ be a fixed vector and $W_{\ell,r}$ to denote the r-th row of W_ℓ, then by the concentration of Gaussian, we have

$$\Pr[\sigma_b(\langle W_{\ell,r}, u_i\rangle) > 0] = \Pr_{z\sim\mathcal{N}(0,\frac{2}{m})}[z > b] \leq \exp(-b^2 m/4).$$

Let S be the following index set $S := \{r \in [m] : \langle W_{\ell,r}, u_i\rangle > b\}$, the above reasoning means that for the indicator random variable $\mathbf{1}[r \in S]$, we have

$$\mathbb{E}[\mathbf{1}[r \in S]] \leq \exp(-b^2 m/4).$$

Use Bernstein's inequality (Lemma 4.2.3) we have that for all $t > 0$,

$$\Pr[|S| > k + t] \leq \exp(-\frac{t^2/2}{k + t/3}),$$

where $k := m \cdot \exp(-b^2 m/4)$. By picking $t = k$, we have

$$\Pr[|S| > 2k] \leq \exp(\frac{-3k}{8}).$$

Note that $|S|$ is essentially the quantity $\|h_{i,\ell}\|_0$, hence we can union bound over all ℓ and i and with probability at least

$$1 - nL \cdot \exp(-\Omega(m \cdot \exp(-b^2 m/4))),$$

we have $\|h_{i,\ell}\|_0 \leq 2m \cdot \exp(-b^2 m/4)$. □

Remark 4.7.7. *The above lemma shows that by using the shifted ReLU activation, we make sure that all $h_{i,\ell}$ are sparse after initialization. Specifically, we use $k := m \cdot \exp(-b^2 m/4)$ as a sparsity parameter. Later, we might rescale b so that the probability becomes $\exp(-b^2/2)$. We stress that such rescaling does not affect the sparsity of our initial vectors. If we rescale b and choose it as $\sqrt{2\alpha \log m}$, then $k = m^{1-\alpha}$ and hence with high probability, $\|h_{i,\ell}\|_0 \leq O(m^{1-\alpha})$.*

As a direct consequence, we note that all initial $D_{i,\ell}$ are k-sparse as well.

We state a lemma that handles the ℓ_2 norm of $h_{i,\ell}$ when one uses *truncated Gaussian distribution* instead. Due to the length and the delicacy of the proof, we defer it to Section 4.8.

Lemma 4.7.8 (Restatement of Lemma 4.8.6)**.** *Let $b > 0$ be a fixed scalar. Let the activation function $\varphi(x) := \sqrt{c_b}\mathbf{1}[x > \sqrt{2/m}b]x$, where $c_b := (2(1 - \Phi(b) + b\varphi(b)))^{-1/2}$. Let $\varepsilon \in (0, 1)$, then over the randomness of $W(0)$, with probability at least*

$$1 - O(nL) \cdot \exp(-\Omega(m \exp(-b^2/2)\varepsilon^2/L^2)),$$

we have

$$\|h_{i,\ell}\|_2 \in [1 - \varepsilon, 1 + \varepsilon], \forall i \in [n], \ell \in [L].$$

The second lemma handles the consecutive product that appears naturally in the gradient computation. It is useful in analyzing the spectral property of the Gram matrix.

Lemma 4.7.9 (Variant of Lemma 7.3 in [5]). *Suppose $m \geq \Omega(nL\log(nL))$, then over the randomness of initializations $W_1(0), \ldots, W_L(0) \in \mathbb{R}^{m\times m}$, for all $i \in [n]$ and $1 \leq a \leq b \leq L$,*

$$\Pr[\|W_b D_{i,b-1} W_{b-1} \ldots D_{i,a} W_a\| \leq O(\sqrt{L})] \geq 1 - e^{-\Omega(k/L^2)}.$$

The proof is similarly to the original proof of the corresponding lemma in [5], however we replace the bound on $h_{i,\ell}$ with our Lemma 4.7.8. We highlight this does not change the bound, merely in expense of a worse probability.

The next several lemmas bound norms after small perturbation.

Lemma 4.7.10 (Lemma 8.2 in [5]). *Suppose Assumption 4.7.4 is satisfied with $\widetilde{R} \leq O(\frac{1}{L^{9/2}\log^3 m})$. With probability at least $1 - e^{-\Omega(m\widetilde{R}^{2/3}L)}$,*

(a) $\Delta g_{i,\ell}$ can be written as $\Delta g_{i,\ell} = \Delta g_{i,\ell,1} + \Delta g_{i,\ell,2}$ where
- *$\|\Delta g_{i,\ell,1}\|_2 \leq O(\widetilde{R}L^{3/2})$*
- *$\|\Delta g_{i,\ell,2}\|_\infty \leq O(\frac{\widetilde{R}L^{5/2}\sqrt{\log m}}{\sqrt{m}})$*

(b) $\|\Delta D_{i,\ell}\|_0 \leq O(m\widetilde{R}^{2/3}L)$ and $\|(\Delta D_{i,\ell})g_{i,\ell}\|_2 \leq O(\widetilde{R}L^{3/2})$.

(c) $\|\Delta g_{i,\ell}\|_2, \|\Delta h_{i,\ell}\|_2 \leq O(\widetilde{R}L^{5/2}\sqrt{\log m})$.

Remark 4.7.11. *Lemma 4.7.10 establishes the connection between parameter $\widetilde{R}$ and s of Assumption 4.7.4 and 4.7.5. As long as $\widetilde{R}$ is small, then we have $s = O(m\widetilde{R}^{2/3}L)$. Such a relation enables us to pick R to our advantage and ensure the sparsity of $\Delta D_{i,\ell}$ is sublinear in m, and hence the update time per iteration is subquadratic in m.*

4.7.3 Bounds on Initialization

In the following lemma, we generalize the Lemma C.2 in [17] into multiple layer neural networks.

Lemma 4.7.12 (Bounds on initialization, multiple layer version of Lemma C.2 in [17]). *Suppose $m = \Omega(nL\log(nL))$, then we have the following*
- $\Pr[f(W, x_i) = \widetilde{O}(1),\ \forall i \in [n]] \geq 1 - e^{-\Omega(\log^2 n)}$.
- $\Pr[\|J_{L,0,i}\| = O(1),\ \forall i \in [n]] \geq 1 - O(nL)\cdot e^{-\Omega(k/L^2)}$.

Proof. We will prove the two parts of the statement separately.

Part 1: By definition, for any $i \in [n]$, we have

$$f(W, x_i) = a^\top \varphi(W_L(\varphi(\cdots \varphi(W_1 x_i)))).$$

We shall make use of Lemma 4.7.8 here:

$$\Pr\left[\|h_{i,L}\|_2 \in [0.9, 1.1], \forall i \in [n]\right] \geq 1 - O(nL)\cdot \exp(-\Omega(k/L^2)).$$

Recall that $a \in \mathbb{R}^m$ has each of its entry being a Rademacher random variable, hence it's 1-subgaussian. Use the concentration of subgaussian (Lemma 4.2.5), we know that

$$\Pr[|a^\top h_{i,L}| \geq 1.1t] \leq 2\exp(-\frac{t^2}{2}),$$

setting $t = O(\log^2 n)$, and union bound over all $i \in [n]$, we conclude our desired result.

Part 2: For the last layer, we consider W_L is initialized as follows: each entry is first sampled from $\mathcal{N}(0,1)$, then we scale down W_L by $\frac{\sqrt{2}}{\sqrt{m}}$. This means we can write the output of last layer as $\frac{\sqrt{2}}{\sqrt{m}} W_L h_{i,L-1}$, and therefore, the gradient is $\frac{\sqrt{2}}{\sqrt{m}} D_{i,L} h_{i,L-1} a^\top$. Hence,

$$\begin{aligned}\|J_{L,0,i}\| &= \frac{1}{\sqrt{m}} \|h_{i,L-1} a^\top D_{i,L}\| \\ &\leq \frac{\sqrt{2}}{\sqrt{m}} \|h_{i,L-1}\|_2 \cdot \|D_{i,L} a\|_2 \\ &= O(1).\end{aligned}$$

The last step follows from the fact that $\|D_{i,L} a\|_2 \leq O(\sqrt{m})$ and $\|h_{i,L-1}\|_2 \leq 1.1$ with probability at least $1 - O(nL) \cdot \exp(-\Omega(k/L^2))$. □

4.7.4 Bounds on Small Perturbation

In the following, we generalize the Lemma C.4 in [17] into multiple layer neural network. We use the interpretation that W_L is generated from $\mathcal{N}(0,1)$ and scaled by $\sqrt{\frac{2}{m}}$ in our proof.

Lemma 4.7.13 (multiple layer version of Lemma C.4 in [17])**.** *Suppose $m = \Omega(nL\log(nL))$, then over the random initialization of*

$$W(0) = \{W_1(0), W_2(0), \cdots W_L(0)\},$$

the following holds with probability at least $1 - nL \cdot e^{-\log^2 m}$, for any set of weight W_L satisfying for each $r \in [m]$,

$$\|W_{L,r} - W_{L,r}(0)\|_2 \leq R/\sqrt{m},$$

- $\|W_L - W_L(0)\|_F \leq R$.
- $\|J_{W_L,x_i} - J_{W_L(0),x_i}\|_2 = \widetilde{O}(R^{1/2}/m^{1/4})$.
- $\|J_{W_L} - J_{W_L(0)}\|_F = \widetilde{O}(n^{1/2} R^{1/2}/m^{1/4})$.
- $\|J_{W_L}\|_F = \widetilde{O}(n^{1/2})$.

Proof. **Part 1.** Note that

$$\begin{aligned}\|W_L - W_L(0)\|_F^2 &= \sum_{r=1}^m \|W_{L,r} - W_{L,r}(0)\|_2^2 \\ &\leq m \cdot R^2/m \\ &= R^2.\end{aligned}$$

Taking square root yields our desired result.

Part 2. To simplify the notation, we ignore the subscripts i below. We have

$$\|J_{W_L,x} - J_{W_L(0),x}\|^2 = \frac{2}{m} \|(D_L(0) + \Delta D_L) a h_L^\top - D_L(0) a h_L^\top\|^2$$

$$
\begin{aligned}
&= \frac{2}{m} \|\Delta D_L a h_L^\top\|^2 \\
&= \frac{2}{m} \|\Delta D_L a h_L^\top\|_F^2 \\
&= \frac{2}{m} \sum_{r\in[m]} a_r^2 \cdot h_{L,r}^2 \cdot |\mathbf{1}[\langle W_{L,r}, h_L\rangle \geq b] - \mathbf{1}[\langle W_{L,r}(0), h_L\rangle \geq b]| \\
&= O(\frac{1}{m}) \sum_{r\in[m]} |\mathbf{1}[\langle W_{L,r}, h_L\rangle \geq b] - \mathbf{1}[\langle W_{L,r}(0), h_L\rangle \geq b]|.
\end{aligned}
$$

where we use $\Delta D_L a h_L^\top$ is a rank 1 matrix in the third step.

Let $s_r := |\mathbf{1}[\langle W_{L,r}, h_L\rangle \geq b] - \mathbf{1}[\langle W_{L,r}(0), h_L\rangle \geq b]|$ and define the event E_r as

$$
E_r = \Big\{ \|W_{L,r} - W_{L,r}(0)\|_2 \leq R/\sqrt{m}, \quad \mathbf{1}[\langle W_{L,r}, h_L\rangle \geq b] \neq \mathbf{1}[\langle W_{L,r}(0), h_L\rangle \geq b] \Big\}.
$$

It is not hard to see that event E_r happens if and only if

$$
W_{L,r}(0)^\top h_L \in [b - \|h_L\|_2 R/\sqrt{m}, b + \|h_L\|_2 R/\sqrt{m}].
$$

By the anti-concentration of Gaussian distribution (Lemma 4.2.4), we have

$$
\mathbb{E}[s_r] = \Pr[E_r = 1] \leq \frac{4}{5} R/\sqrt{m}.
$$

We have

$$
\begin{aligned}
\Pr\Big[\sum_{r=1}^m s_r \geq (t + \frac{4}{5})\|h_L\| R\sqrt{m}\Big] \leq\ & \Pr\Big[\sum_{r=1}^m (s_r - \mathbb{E}[s_r]) \geq t\|h_L\|_2 R\sqrt{m}\Big] \\
&\leq 2\exp(-\frac{2t^2R^2\|h_L\|_2^2 m^2}{m^2}) \\
&= 2\exp(-2t^2R^2\|h_L\|_2^2) \\
&\leq 2\exp(-t^2),
\end{aligned}
$$

where the first step follows from our above analysis, we use Lemma 4.2.2 in the second step, and we use both $\|h_L\|_2^2 \geq 0.5$ and $R \geq 1$ in the final step.

Set $t = \log m$ and by union bound over i, we have with probability at least $1 - n \cdot e^{-\log^2 m}$,

$$
\begin{aligned}
\|J_{W_L,x_i} - J_{W_L(0),x_i}\|^2 &= \frac{1}{m}\sum_{r=1}^m s_r \\
&\leq \frac{1}{m}\widetilde{O}(R\sqrt{m}) \\
&= \widetilde{O}(\frac{R}{\sqrt{m}}).
\end{aligned}
$$

Taking square root yields our desired result.

Part 3. Note that the squared Frobenious norm is just the sum of all squared ℓ_2 norm of rows, hence

$$\|J_{W_L} - J_{W_L(0)}\|_F \leq \widetilde{O}(n^{1/2}R^{1/2}/m^{1/4}).$$

Part 4. We will prove by triangle inequality:

$$\begin{aligned}\|J_{W_L}\|_F &\leq \|J_{W_L(0)}\|_F + \|J_{W_L} - J_{W_L(0)}\|_F \\ &\leq \widetilde{O}(n^{1/2}) + \widetilde{O}(n^{1/2}R^{1/2}/m^{1/4}) \\ &= \widetilde{O}(n^{1/2}).\end{aligned}$$

Note that, in the final step, we use both the choice of R (see Def. 4.7.18) and m (see Def. 4.7.20). □

4.7.5 Putting It All Together

In this section, we will prove the following core theorem that analyzes the convergence behavior of Algorithm 20:

Theorem 4.7.14 (Formal version of Theorem 1.2.5)**.** *Suppose the neural network width satisfies $m = \Omega(\lambda_L^{-2}n^2L^2)$, then over the randomness of the initialization of the neural network and the randomness of the algorithm, Algorithm 20 satisfies*

$$\Pr[\|f_{t+1} - y\|_2 \leq \frac{1}{3}\|f_t - y\|_2] \geq 1 - \exp(-\Omega(\log^2 n)).$$

Before moving on, we introduce several definitions and prove some useful facts related to them.

Definition 4.7.15 (function J)**.** We define

$$\mathsf{J}_\ell(Z_1, \ldots, Z_L)_i := D_{i,\ell}(Z_\ell) \prod_{k=\ell+1}^{L} Z_k^\top D_{i,k}(Z_k) a(h_i(Z_1, \ldots, Z_{\ell-1}))^\top \quad \in \mathbb{R}^{m_\ell \times m_{\ell-1}}$$

where

$$\begin{aligned} D_{i,\ell}(Z_\ell) &:= \mathrm{diag}(\varphi'(Z_\ell h_i(Z_1, \ldots, Z_{\ell-1}))), && \in \mathbb{R}^{m_\ell \times m_\ell} \\ h_i(Z_1, \ldots, Z_{\ell-1}) &:= \varphi(Z_{\ell-1}(\varphi(Z_{\ell-2} \cdots (\varphi(Z_1 x_i))))) && \in \mathbb{R}^{m_{\ell-1}} \end{aligned}$$

Fact 4.7.16. *Let J_ℓ denote the function be defined as Definition 4.7.15. For any $t \in \{0, \ldots, T\}$, we have*

$$f_{t+1} - f_t = \left(\int_0^1 \mathsf{J}_L((1-s)W(t) + sW(t+1)) \mathrm{d}s \right)^\top \cdot \mathrm{vec}(W_L(t+1) - W_L(t)),$$

Proof. For $i \in [n]$, consider the i-th coordinate.

$$(f_{t+1} - f_t)_i = \int_0^1 f((1-s)W(t) + sW(t+1), x_i)' \mathrm{d}s$$

$$= \int_0^1 \left(\frac{\partial f}{\partial W_L}((1-s)W(t) + sW(t+1), x_i) \right)^\top \cdot \mathrm{vec}(W_L(t+1) - W_L(t)) \mathrm{d}s$$
$$= \left(\int_0^1 \mathsf{J}_L((1-s)W(t) + sW(t+1))_i \mathrm{d}s \right)^\top \cdot \mathrm{vec}(W_L(t+1) - W_L(t)),$$

Thus, we complete the proof. □

Fact 4.7.17. *For any $t \in \{0, \dots, T\}$, we have $\mathsf{J}_\ell(W_1(t), \dots, W_L(t)) = J_{\ell,t}$.*

Proof. In order to simplify the notation, we drop the term t below.

We note that for $i \in [n]$, the i-th row of matrix $J_{\ell,t}$ is defined as

$$D_{i,\ell}(\prod_{k=\ell+1}^{L} W_k^\top D_{i,k}) a h_{i,\ell-1}^\top,$$

where

$$D_{i,\ell} = \mathrm{diag}(\varphi'(W_\ell h_{i,\ell-1})),$$
$$h_{i,\ell-1} = \varphi(W_{\ell-1}(\varphi(W_{\ell-2} \dots (\varphi(W_1 x_i))))),$$

this is essentially the same as $h_i(W_1, \dots, W_{\ell-1})$ and $D_{i,\ell}(W_\ell)$. This completes the proof. □

We state the range we require for parameter $\widetilde{R}$ and R:

Definition 4.7.18. We choose $\widetilde{R}$ so that

$$\frac{2}{\sqrt{m}} \cdot \frac{n}{\lambda_L} \leq \widetilde{R} \leq \min\{\frac{1}{L^{4.5} \log^3 m}, \frac{\lambda_L}{n}\}.$$

Recall that R is the scale-up version of $\widetilde{R}$, hence

$$\frac{n}{\lambda_L} \leq R \leq \min\{\frac{1}{L^{4.5} \log^3 m}, \frac{\lambda_L}{n}\} \cdot \sqrt{m}.$$

Remark 4.7.19. *Recall that the sparsity parameter s is directly related to $\widetilde{R}$: $s = O(m\widetilde{R}^{2/3}L)$, hence to ensure the sparsity is small, we shall pick $\widetilde{R}$ as small as possible.*

Next, we pick the value of m:

Definition 4.7.20. We choose m to be

$$m \geq \Omega(n^4 L \lambda_L^{-4}).$$

We use induction to prove the following two claims recursively.

Definition 4.7.21 (Induction hypothesis 1)**.** Let $t \in [T]$ be a fixed integer. We have

$$\|W_{L,r}(t) - W_{L,r}(0)\|_2 \leq R/\sqrt{m}$$

holds for any $r \in [m]$.

Definition 4.7.22 (Induction Hypothesis 2)**.** Let $t \in [T]$ be a fixed integer. We have

$$\|f_t - y\|_2 \leq \frac{1}{3}\|f_{t-1} - y\|_2.$$

Suppose the above two claims hold up to t, we prove they continue to hold for time $t+1$. The second claim is more delicate, we are going to prove it first and we define

$$J_{\ell,t,t+1} := \int_0^1 \mathsf{J}_\ell\Big((1-s)W_t + sW_{t+1}\Big)\mathrm{d}s,$$

where J_ℓ is defined as Definition 4.7.15.

Lemma 4.7.23. *Let $g_L^\star := (J_{L,t}J_{L,t}^\top)^{-1}(f_t - y)$. We have*

$$\begin{aligned}\|f_{t+1} - y\|_2 \leq & \|f_t - y - J_{L,t}J_{L,t}^\top g_{L,t}\|_2 \\ & + \|(J_{L,t} - J_{L,t,t+1})J_{L,t}^\top g_L^\star\|_2 \\ & + \|(J_{L,t} - J_{L,t,t+1})J_{\ell,t}^\top (g_{L,t} - g_L^\star)\|_2. \end{aligned} \tag{4.7}$$

Proof. Consider the following computation:

$$\begin{aligned}
& \|f_{t+1} - y\|_2 \\
= & \|f_t - y + (f_{t+1} - f_t)\|_2 \\
= & \|f_t - y + J_{L,t,t+1} \cdot \mathrm{vec}(W_{L,t+1} - W_{L,t})\|_2 \\
= & \|f_t - y - J_{L,t,t+1} \cdot J_{L,t}^\top g_{L,t}\|_2 \\
= & \|f_t - y - J_{L,t}J_{L,t}^\top g_{L,t} + J_{L,t}J_{L,t}^\top g_{L,t} - J_{L,t,t+1}J_{L,t}^\top g_{L,t}\|_2 \\
\leq & \|f_t - y - J_{L,t}J_{L,t}^\top g_{L,t}\|_2 + \|(J_{L,t} - J_{L,t,t+1})J_{L,t}^\top g_{L,t}\|_2 \\
\leq & \|f_t - y - J_{L,t}J_{L,t}^\top g_{L,t}\|_2 + \|(J_{L,t} - J_{L,t,t+1})J_{L,t}^\top g_L^\star\|_2 + \|(J_{L,t} - J_{L,t,t+1})J_{L,t}^\top (g_{L,t} - g_L^\star)\|_2,
\end{aligned}$$

The second step follows from the definition of $J_{L,t,t+1}$ and simple calculus. □

Claim 4.7.24 (1st term in Eq. (4.7))**.** *We have*

$$\|f_t - y - J_{L,t}J_{L,t}^\top g_{L,t}\|_2 \leq \frac{1}{9}\|f_t - y\|_2.$$

Proof. We have

$$\begin{aligned}\|f_t - y - J_{L,t}J_{L,t}^\top g_{L,t}\|_2 \leq & \varepsilon_0 \|f_t - y\|_2 \\ \leq & \frac{1}{9}\|f_t - y\|_2, \end{aligned} \tag{4.8}$$

since $g_{L,t}$ is an ε_0 ($\varepsilon_0 \leq \frac{1}{9}$) approximate solution to the regression problem

$$\min_g \|J_{L,t}J_{L,t}^\top g - (f_t - y)\|_2.$$

□

Claim 4.7.25 (2nd term in Eq. (4.7))**.** *We have*

$$\|(J_{L,t} - J_{L,t,t+1})J_{L,t}^\top g_L^\star\|_2 \leq \frac{1}{9}\|f_t - y\|_2.$$

Proof. We bound the second term in Eq. (4.7) as follows:

$$\begin{aligned}
\|(J_{L,t} - J_{L,t,t+1})J_{L,t}^\top g_L^\star\|_2 &\leq \|J_{L,t} - J_{L,t,t+1}\| \cdot \|J_{L,t}^\top g_L^\star\|_2 \\
&= \|J_{L,t} - J_{L,t,t+1}\| \cdot \|J_{L,t}^\top (J_{L,t}J_{L,t}^\top)^{-1} \cdot (f_t - y)\|_2 \\
&\leq \|J_{L,t} - J_{L,t,t+1}\| \cdot \|J_{L,t}^\top (J_{L,t}J_{L,t}^\top)^{-1}\| \cdot \|f_t - y\|_2. \qquad (4.9)
\end{aligned}$$

We bound these term separately.

For the first term in Eq. (4.9),

$$\begin{aligned}
\|J_{L,t} - J_{L,t,t+1}\| &= \left\| \mathsf{J}_L(W_t) - \int_0^1 \mathsf{J}_L((1-s)W_t + sW_{t+1}) \mathrm{d}s \right\| \\
&\leq \int_0^1 \|\mathsf{J}_L(W_t) - \mathsf{J}_L((1-s)W_t + sW_{t+1})\| \,\mathrm{d}s \\
&\leq \int_0^1 \|\mathsf{J}_L(W_t) - \mathsf{J}_L(W_0)\| + \|\mathsf{J}_L(W_0) - \mathsf{J}_L((1-s)W_t + sW_{t+1})\| \,\mathrm{d}s \\
&\leq \|\mathsf{J}_L(W_t) - \mathsf{J}_L(W_0)\| + \int_0^1 \|\mathsf{J}_L(W_0) - \mathsf{J}_L((1-s)W_t + sW_{t+1})\| \,\mathrm{d}s \\
&\leq \widetilde{O}(n^{1/2}R^{1/2}/m^{1/4}), \qquad (4.10)
\end{aligned}$$

where by Fact 4.7.17, we know $\|\mathsf{J}_L(W_t) - \mathsf{J}_L(W_0)\| = \|J_{W_L(t)} - J_{W_L(0)}\| \leq \widetilde{O}(n^{1/2}R^{1/2}/m^{1/4})$, we use Lemma 4.7.13 in the last inequality.

For the term $\int_0^1 \|\mathsf{J}_L(W_0) - \mathsf{J}_L((1-s)W_t + sW_{t+1})\| \,\mathrm{d}s$, we analyze the following:

$$\begin{aligned}
&\|(1-s)\cdot \mathrm{vec}(W_L(t)) + s\cdot \mathrm{vec}(W_L(t+1)) - \mathrm{vec}(W_L(0))\|_2 \\
\leq\ & (1-s)\cdot \|\mathrm{vec}(W_L(t)) - \mathrm{vec}(W_L(0))\|_2 + s\cdot \|\mathrm{vec}(W_L(t+1)) - \mathrm{vec}(W_L(0))\|_2 \\
=\ & (1-s)\cdot \|W_L(t) - W_L(0)\|_F + s\cdot \|W_L(t+1) - W_L(0)\|_F \\
\leq\ & O(R).
\end{aligned}$$

This means the perturbation of $(1-s)W_L(t) + sW_L(t+1)$ with respect to $W_L(0)$ is small, hence $\|\mathsf{J}_L(W_0) - \mathsf{J}_L((1-s)W_t + sW_{t+1})\| = \widetilde{O}(n^{1/2}R^{1/2}/m^{1/4})$.

Furthermore, we have

$$\|J_{L,t}^\top (J_{L,t}J_{L,t}^\top)^{-1}\| = \frac{1}{\sigma_{\min}(J_{L,t}^\top)} \leq \sqrt{2/\lambda_L}, \qquad (4.11)$$

where the second inequality follows from $\sigma_{\min}(J_{L,t}) = \sqrt{\lambda_{\min}(J_{L,t}J_{L,t}^\top)} \geq \sqrt{\lambda_L/2}$ (see Lemma 4.6.8).

Combining Eq. (4.9), (4.10) and (4.11), we have

$$\begin{aligned}
\|(J_{L,t} - J_{L,t,t+1})J_{L,t}^\top g_L^\star\|_2 &\leq \widetilde{O}(n^{1/2}R^{1/2}/m^{1/4}) \cdot \lambda_L^{-1/2} \cdot \|f_t - y\|_2 \\
&\leq \frac{1}{9}\|f_t - y\|_2, \qquad (4.12)
\end{aligned}$$

where the last step follows from choice of m (Definition 4.7.20). □

Claim 4.7.26 (3rd term in Eq. (4.7))**.** *We have*

$$\|(J_{L,t} - J_{L,t,t+1})J_{L,t}^\top(g_{L,t} - g_L^\star)\|_2 \leq \frac{1}{9}\|f_t - y\|_2$$

Proof. We can show

$$\|(J_{L,t} - J_{L,t,t+1})J_{L,t}^\top(g_{L,t} - g_L^\star)\|_2 \leq \|J_{L,t} - J_{L,t,t+1}\| \cdot \|J_{L,t}^\top\| \cdot \|g_{L,t} - g_L^\star\|_2. \tag{4.13}$$

Moreover, one has

$$\begin{aligned}
\frac{\lambda}{2}\|g_{L,t} - g_L^\star\|_2 &\leq \lambda_{\min}(J_{L,t}J_{L,t}^\top) \cdot \|g_{L,t} - g_L^\star\|_2 \\
&\leq \|J_{L,t}J_{L,t}^\top g_{L,t} - J_{L,t}J_{L,t}^\top g_L^\star\|_2 \\
&= \|J_{L,t}J_{L,t}^\top g_{L,t} - (f_t - y)\|_2 \\
&\leq \frac{\sqrt{\lambda_L/n}}{2} \cdot \|f_t - y\|_2. \qquad (4.14)
\end{aligned}$$

The first step comes from $\lambda_{\min}(J_{L,t}J_{L,t}^\top) = \lambda_{\min}(G_{L,t}) \geq \lambda_L/2$ (see Lemma 4.6.8). The last step follows from $g_{L,t}$ is an ε_0 ($\varepsilon_0 \leq \sqrt{\lambda_L/n}$) approximate solution to the regression.

Consequently, we have

$$\begin{aligned}
\|(J_{L,t} - J_{L,t,t+1})J_{L,t}^\top(g_{L,t} - g_L^\star)\|_2 &\leq \|J_{L,t} - J_{L,t,t+1}\| \cdot \|J_{L,t}^\top\| \cdot \|g_{L,t} - g_L^\star\|_2 \\
&\leq \widetilde{O}(n^{1/2}R^{1/2}/m^{1/4}) \cdot \widetilde{O}(n^{1/2}) \cdot \frac{2}{\sqrt{n\lambda_L}} \cdot \|f_t - y\|_2 \\
&= \widetilde{O}(\frac{n^{1/2}R^{1/2}}{m^{1/4}\lambda_L^{1/2}}) \cdot \|f_t - y\|_2.
\end{aligned}$$

Note that, for the 2nd step, it follows from Eq. (4.10) and (4.14) and the fact that $\|J_{L,t}\| \leq O(\sqrt{n})$ (see Lemma 4.7.13).

Finally, we have

$$\begin{aligned}
\|(J_{L,t} - J_{L,t,t+1})J_{L,t}^\top(g_{L,t} - g_L^\star)\|_2 &\leq \widetilde{O}(\frac{n^{1/2}R^{1/2}}{m^{1/4}\lambda_L^{1/2}}) \cdot \|f_t - y\|_2 \qquad (4.15) \\
&\leq \frac{1}{9}\|f_t - y\|_2. \qquad (4.16)
\end{aligned}$$

The last step follows from choice of m (Definition 4.7.20).

□

Lemma 4.7.27 (Putting it all together)**.** *We have*

$$\|f_{t+1} - y\|_2 \leq \frac{1}{3}\|f_t - y\|_2. \tag{4.17}$$

Proof. Combining Eq. (4.7), (4.8), (4.12), and (4.15), we have proved the second claim, i.e.,

$$\|f_{t+1} - y\|_2 \leq \frac{1}{3}\|f_t - y\|_2.$$

□

4.7.6 Bounds on the Movement of Weights

Lemma 4.7.28. *Let R be chosen as in Definition 4.7.18, then the following holds:*

$$\|W_{L,r}(t+1) - W_{L,r}(0)\|_2 \leq R/\sqrt{m}.$$

Proof. First, we have

$$\begin{aligned}
\|g_{L,t}\|_2 &\leq \|g_L^\star\|_2 + \|g_{L,t} - g_L^\star\|_2 \\
&= \|(J_{L,t}J_{L,t}^\top)^{-1}(f_t - y)\|_2 + \|g_{L,t} - g_L^\star\|_2 \\
&\leq \|(J_{L,t}J_{L,t}^\top)^{-1}\| \cdot \|(f_t - y)\|_2 + \|g_{L,t} - g_L^\star\|_2 \\
&\leq \frac{1}{\lambda_L} \cdot \|f_t - y\|_2 + \frac{1}{\sqrt{n\lambda_L}} \cdot \|f_t - y\|_2 \\
&\lesssim \frac{1}{\lambda_L} \cdot \|f_t - y\|_2 \qquad (4.18)
\end{aligned}$$

where the third step is owing to Eq. (4.14), and the final step is due to the fact that $1/\sqrt{n\lambda_L} \leq 1/\lambda_L$.

Then

$$\begin{aligned}
\|W_{L,r}(k+1) - W_{L,r}(k)\|_2 &= \Big\| \sum_{i=1}^n \frac{1}{\sqrt{m}} a_r h_{i,L-1}^\top \mathbf{1}[\langle W_{L,r}(k), h_{i,L-1}\rangle \geq 0] g_{L,k,i} \Big\|_2 \\
&\leq O(\frac{1}{\sqrt{m}}) \sum_{i=1}^n |g_{L,k,i}| \\
&\leq O(\frac{\sqrt{n}}{\sqrt{m}}) \cdot \|g_{L,k}\|_2 \\
&\leq O(\frac{\sqrt{n}}{\sqrt{m}}) \cdot \frac{1}{\lambda_L} \cdot \|f_k - y\|_2 \\
&\leq O(\frac{n^{1/2}}{2^k \lambda_L m^{1/2}}) \cdot \|f_0 - y\|_2 \\
&\leq \widetilde{O}(\frac{n}{2^k \lambda_L m^{1/2}}).
\end{aligned}$$

The first step follows from the update rule, the second step is by triangle inequality, the third step uses the fact the ℓ_1 norm of a vector is upper bounded by $\sqrt{n}$ times the ℓ_2 norm, the fourth step is by Eq. (4.18), and the last step is by each entry of f_0 and y is of order $\widetilde{O}(1)$.

Consequently, we have

$$\begin{aligned}
\|W_{L,r}(t+1) - W_{L,r}(0)\|_2 &\leq \sum_{k=0}^t \|W_{L,r}(k+1) - W_{L,r}(k)\|_2 \\
&\leq \sum_{k=0}^t \widetilde{O}(\frac{n}{2^k \lambda_L m^{1/2}})
\end{aligned}$$

$$\leq \widetilde{O}(\frac{n}{\lambda_L m^{1/2}}).$$

By the choice of R (Definition 4.7.18), we know this is upper bounded by $R/\sqrt{m}$. This concludes our proof.

□

4.8 Bounds on the Intermediate Layer Output with Shifted ReLU

In this section, we prove a technical lemma (Lemma 4.7.8) involving truncated gaussian distribution, which correlates to the shifted ReLU activation we use.

Definition 4.8.1 (Truncated Gaussian distribution)**.** Suppose $X \sim \mathcal{N}(0, \sigma^2)$. Let $b \in \mathbb{R}$. Then, we say a random variable Y follows from a truncated Gaussian distribution $\mathcal{N}_b(0, \sigma^2)$ if $Y = X|X \geq b$. The probability density function for $\mathcal{N}_b(0, \sigma^2)$ is as follows:

$$f(y) = \frac{1}{\sigma(1 - \Phi(b/\sigma))} \cdot \frac{1}{\sqrt{2\pi}} e^{-y^2/(2\sigma^2)} \quad y \in [b, \infty),$$

where $\Phi(\cdot)$ is the standard Gaussian distribution's CDF.

Fact 4.8.2 (Properties of truncated Gaussian distribution)**.** *For $b \in \mathbb{R}$, suppose $X \sim \mathcal{N}_b(0, \sigma^2)$. Let $\beta := b/\sigma$. Then, we have*

- $\mathbb{E}[X] = \frac{\sigma\varphi(\beta)}{1-\Phi(\beta)}$, *where* $\varphi(x) := \frac{1}{\sqrt{2\pi}} e^{-x^2/2}$.
- $\mathbf{Var}[X] = \sigma^2(1 + \beta\varphi(\beta)/(1 - \Phi(\beta)) - (\varphi(\beta)/(1 - \Phi(\beta)))^2)$.
- $X/\sigma \sim \mathcal{N}_{b/\sigma}(0, 1)$.
- *When $\sigma = 1$, X is $C(b + 1)$-subgaussian, where $C > 0$ is an absolute constant.*

Lemma 4.8.3 (Concentration inequality for b-truncated chi-square distribution)**.** *For $b \in \mathbb{R}$, $n > 0$, let $X \sim \chi^2_{b,n}$; that is, $X = \sum_{i=1}^n Y_i^2$ where $Y_1, \ldots, Y_n \sim \mathcal{N}_b(0, 1)$ are independent b-truncated Gaussian random variables. Then, there exist two constants C_1, C_2 such that for any $t > 0$,*

$$\Pr\left[\left|X - n(1 + \frac{b\varphi(b)}{1 - \Phi(b)})\right| \geq nt\right] \leq \exp\left(-C_1 nt^2/b^4\right) + \exp\left(-C_2 nt/b^2\right).$$

In particular, we have

$$\Pr\left[|X - n(1 + b(b + 1))| \geq t\right] \leq \exp\left(-C_1 t^2/(nb^4)\right) + \exp\left(-C_2 t/b^2\right).$$

Proof. Since we know that $Y_i \sim \mathcal{N}_b(0, 1)$ is $C(b + 1)$-subgaussian, it implies that Y_i^2 is a sub-exponential random variable with parameters $(4\sqrt{2}C^2(b + 1)^2, 4C^2(b + 1)^2)$. Hence, by the standard concentration of sub-exponential random variables, we have

$$\Pr\left[\left|\sum_{i=1}^n Y_i^2 - n\mathbb{E}[Y_i^2]\right| \geq nt\right] \leq \begin{cases} 2\exp\left(-\frac{nt^2}{2\cdot 32C^4(b+1)^4}\right) & \text{if } nt \leq 8C^2(b+1)^2 \\ 2\exp\left(-\frac{nt}{2\cdot 4C^2(b+1)^2}\right) & \text{otherwise} \end{cases}$$

$$\leq 2\exp\left(-C_1 nt^2/b^4\right) + 2\exp\left(-C_2 nt/b^2\right).$$

□

Fact 4.8.4. *Let $h \in \mathbb{R}^p$ be fixed vectors and $h \neq 0$, let $b > 0$ be a fixed scalar, $W \in \mathbb{R}^{m\times p}$ be random matrix with i.i.d. entries $W_{i,j} \sim \mathcal{N}(0, \frac{2}{m})$ and vector $v \in \mathbb{R}^m$ defined as $v_i = \varphi((Wh)_i) = \mathbf{1}[(Wh)_i \geq b](Wh)_i$. Then*

- *$|v_i|$ follows i.i.d. from the following distribution: with probability $1 - e^{-b^2 m/(4\|h\|^2)}$, $|v_i| = 0$, and with probability $e^{-b^2 m/(4\|h\|^2)}$, $|v_i|$ follows from truncated Gaussian distribution $\mathcal{N}_b(0, \frac{2}{m}\|h\|_2^2)$.*
- *$\frac{m\|v\|_2^2}{2\|h\|_2^2}$ is in distribution identical to $\chi^2_{b',\omega}$ (b'-truncated chi-square distribution of order ω) where ω follows from binomial distribution $\mathcal{B}(m, e^{-b^2 m/(4\|h\|^2)})$ and $b' = \frac{\sqrt{m/2}}{\|h\|_2} b$.*

Proof. We assume each vector W_i is generated by first generating a gaussian vector $g \sim \mathcal{N}(0, \frac{2}{m}I)$ and then setting $W_i = \pm g$ where the sign is chosen with half-half probability.

Now, $|\langle W_i, h\rangle| = |\langle g, h\rangle|$ only depends on g, and is in distribution identical to $\mathcal{N}_b(0, \frac{2}{m}\|h\|_2^2)$.

Next, after the sign is determined, the indicator $\mathbf{1}[(W_i h)_i \geq b]$ is 1 with probability $e^{-b^2 m/(4\|h\|^2)}$ and 0 with probability $1 - e^{-b^2 m/(4\|h\|^2)}$.

Therefore, $|v_i|$ satisfies the aforementioned distribution.

As for $\|v\|_2^2$, letting $\omega \in \{0, 1, \ldots, m\}$ be the variable indicates how many indicators are 1, then $\omega \sim \mathcal{B}(m, e^{-b^2 m/(4\|h\|^2)})$ and $\frac{m\|v\|_2^2}{2\|h\|_2^2} \sim \chi^2_{b',\omega}$, where $b' = \frac{\sqrt{m/2}}{\|h\|_2} b$. □

Fact 4.8.5 (Gaussian tail bound)**.** *For any $b > 0$, we have*

$$\frac{e^{-b^2/2}}{C(b+1)} \leq 1 - \Phi(b) \leq e^{-b^2/2},$$

where C is an absolute constant.

We prove a truncated Gaussian version of Lemma 7.1 of [5].

Lemma 4.8.6. *Let $b > 0$ be a fixed scalar. Let the activation function be defined as*

$$\varphi(x) := \sqrt{c_b}\mathbf{1}[x > \sqrt{2/m}b]x,$$

where

$$c_b := (2(1 - \Phi(b) + b\varphi(b)))^{-1}.$$

Let $\varepsilon \in (0, 1)$, then over the randomness of $W(0)$, with probability at least

$$1 - O(nL)\cdot\exp(-\Omega(m\exp(-b^2/2)\varepsilon^2/L^2)),$$

we have

$$\|h_{i,\ell}\|_2 \in [1-\varepsilon, 1+\varepsilon], \forall i \in [n], \ell \in [L].$$

Proof. We only prove for a fixed $i \in [n]$ and $\ell \in \{0, 1, 2, \ldots, L\}$ because we can apply union bound at the end. Below, we drop the subscript i for notational convenience, and write $h_{i,\ell}$ and x_i as h_ℓ and x respectively.

According to Fact 4.8.4, fixing any $h_{\ell-1} \neq 0$ and letting W_ℓ be the only source of randomness, we have

$$\frac{m}{2}\|h_\ell\|_2^2 \sim \chi^2_{b/\|h\|_2,\omega}, \quad \text{with} \quad \omega \sim \mathcal{B}(m, 1-\Phi(b')),$$

where $b' := b/\|h_{\ell-1}\|_2$.

We first consider the $\ell = 1$ case. Then, we have $\|h_{\ell-1}\|_2 = 1$, and $b' = b$. Let $P_b := 1-\Phi(b)$. By Chernoff bound, for any $\delta \in (0,1)$, we have

$$\Pr[\omega \in (1 \pm \delta)mP_b] \geq 1 - \exp(-\Omega(\delta^2 P_b m)).$$

In the following proof, we condition on this event. By Fact 4.8.5,

$$\omega \in (1 \pm \delta)P_b m \iff \omega \in \left[(1-\delta)\frac{e^{-b^2/2}}{C(b+1)}m, (1+\delta)\exp(-b^2/2)m\right].$$

By Lemma 4.8.3, we have

$$\Pr\left[\left|\frac{m}{2}\|h_1\|_2^2 - \omega\left(1 + \frac{b\varphi(b)}{P_b}\right)\right| > t\right] \leq \exp\left(-\Omega(t^2/(\omega b^4))\right) + \exp\left(-\Omega(t/b^2)\right)$$

Note that

$$\omega\left(1 + \frac{b\varphi(b)}{P_b}\right) \in (1 \pm \delta)mP_b + (1 \pm \delta)mP_b \cdot \frac{b\varphi(b)}{P_b} = (1 \pm \delta)(P_b + b\varphi(b)) \cdot m.$$

Let $c_b^{-1} := 2(P_b + b\varphi(b))$ be the normalization constant. Then, we have

$$\Pr[|c_b\|h_1\|_2^2 - (1 \pm \delta)| > 2tc_b/m] \leq \exp\left(-\Omega(t^2/(\omega b^4))\right) + \exp\left(-\Omega(t/b^2)\right).$$

We want $2tc_b/m = O(\delta)$, i.e., $t = O(\delta c_b^{-1} m)$. Then, we have $\omega t = m^{\Omega(1)} > b^2$. Hence, by Lemma 4.8.3, we actually have

$$\Pr[|c_b\|h_1\|_2^2 - (1 \pm \delta)| > O(\delta)] \leq \exp\left(-\Omega(\delta m/(c_b b^2))\right).$$

By taking $\delta = \varepsilon/L$, we get that

$$\|h_1\|_2^2 \in [1 - \varepsilon/L, 1 + \varepsilon/L]$$

holds with probability at least

$$1 - \exp(-\Omega(\varepsilon^2 P_b m/L^2)) - \exp\left(-\Omega(\varepsilon m/(c_b b^2 L))\right) \geq 1 - \exp(-\Omega(\varepsilon^2 P_b m/L^2)),$$

where the last step follows from $\frac{1}{c_b b^2} = \frac{P_b + b\varphi(b)}{b^2} = \Theta(P_b)$.

We can inductively prove the $\ell > 1$ case. Since the blowup of the norm of h_1 is $1 \pm \varepsilon/L$, the concentration bound is roughly the same for h_ℓ for $\ell \geq 2$. Thus, by carefully choosing the parameters, we can achieve $\|h_\ell\|_2^2 \in [(1-\varepsilon/L)^\ell, (1+\varepsilon/L)^\ell]$ with high probability.

In this end, by a union bound over all the layers $\ell \in [L]$ and all the input data $i \in [n]$, we get that

$$\|h_{i,\ell}\|_2 \in [1-\varepsilon, 1+\varepsilon]$$

with probability at least

$$1 - O(nL)\exp(-\Omega(\varepsilon^2 P_b m/L^2)),$$

which completes the proof of the lemma.

□

Appendix A

The AFN Data Structure

In this section, we include the algorithm and correctness analysis of the AFN data structure.

Throughout this section, we use n to denote the number of data points, and d denote the dimension of the data.

A.1 Algorithm

The AFN data structure we are going to use has similar high-level idea as that of Indyk [39], but we give an improved analysis on the overall running time.

In this section, we present our algorithm that solves approximate Min-IP efficiently. We start with presenting the SORTEDLIST data structure in Alg. 23.

Algorithm 23 Helper data structure SORTEDLIST

data structure SORTEDLIST ▷ This data structure can be implemented via various self-balancing binary search trees
  INIT($P \in (\mathbb{R} \times \mathbb{R}^d)^n$) ▷ n points each has a real key and d dimensional data points, $O(n \log n)$ time
  INSERT($p \in \mathbb{R} \times \mathbb{R}^d$) ▷ Insert a single key-value pair, $O(\log n)$ time
  DELETE($p \in \mathbb{R} \times \mathbb{R}^d$) ▷ Remove a single key-value pair, $O(\log n)$ time
  SEARCHLEQ($T \in \mathbb{R}$) ▷ Output a subtree with key less than or equal to T, $O(\log n)$ time
  SEARCHGEQ($T \in \mathbb{R}$) ▷ Output a subtree with key greater than or equal to T, $O(\log n)$ time
  MAX() ▷ Return max key-value pair, $O(\log n)$ time
  MIN() ▷ Return min key-value pair, $O(\log n)$ time
end data structure

Next, we introduce a task called $(\overline{c}, r)$-DFN defined in Task A.1.1.

Task A.1.1. Let $P \subset \mathbb{R}^d$ be an n-point dataset. Let $\overline{c} > 1$ We define the $(\overline{c}, r)$-DFN problem as follows: given a point $q \in \mathbb{R}^d$ and $r > 0$, if there exists a point $p \in P$ such that, if $\|p - q\|_2 \geq r$, then the data structure reports a point $\widehat{p} \in P$ such that $\|\widehat{p} - q\|_2 \geq r/\overline{c}$, otherwise, it reports "Fail".

The data structure for $(\overline{c}, r)$-DFN shown in Alg. 24 and Alg. 25 is the building block of our approximate Min-IP algorithm.

Algorithm 24 Data structure DFN: members, init, insert and delete

data structure DFN ▷ Theorem A.3.1

members
- $\ell \in \mathbb{N}_+$ ▷ Number of random directions
- $G \in \mathbb{R}^{\ell \times d}$ ▷ Random Gaussian vectors
- SORTEDLIST $L_1, \ldots, L_\ell$ ▷ ℓ sorted lists, Alg. 23
- $t \in \mathbb{R}_+$ ▷ Threshold parameter
- $\overline{c} \in (1, \infty)$ ▷ Approximation parameter

end members

procedure INIT($A \in \mathbb{R}^{n \times d}, \overline{c} \in (1, \infty)$)
- $\overline{c} \leftarrow \overline{c}$
- $\ell \leftarrow \Theta(n^{1/\overline{c}^2} \log^{(1-1/\overline{c})/2} n)$
- $t \leftarrow \Theta(\sqrt{\log n})$ ▷ t is the solution to $e^{t^2/2}/t = 2n$
- $G_{i,j} \sim \mathcal{N}(0,1), \forall i \in [\ell], \forall j \in [d]$ ▷ Each entry is a standard Gaussian
- SORTEDLIST $L_1, \ldots, L_\ell$ ▷ Alg. 23
- Let g_i denote the i-th row of G and A_i denote the i-th row of A
- **for** $i = 1 \to \ell$ **do**
  - $P_i \leftarrow \{(\langle g_i, a_j \rangle, a_j) : j \in [n]\}$
  - L_i.INIT(P_i) ▷ Alg. 23
- **end for**

end procedure

procedure INSERT($p \in \mathbb{R}^d$)
- **for** $i = 1 \to \ell$ **do**
  - $k \leftarrow \langle g_i, p \rangle$
  - L_i.INSERT((k, p)) ▷ Alg. 23
- **end for**

end procedure

procedure DELETE($p \in \mathbb{R}^d$)
- **for** $i = 1 \to \ell$ **do**
  - $k \leftarrow \langle g_i, p \rangle$
  - L_i.DELETE((k, p)) ▷ Alg. 23
- **end for**

end procedure

end data structure

Algorithm 25 Data structure DFN: query

data structure DFN ▷ Theorem A.3.1

procedure QUERY($q \in \mathbb{R}^d, r \in \mathbb{R}_+$)
    $T \leftarrow rt/\overline{c}$
    $i \leftarrow 1, m \leftarrow 0$
    $S \leftarrow \emptyset$
    while $i \leq \ell$ and $m \leq 2\ell + 1$ **do**
        $\mathrm{dist} \leftarrow \langle g_i, q \rangle$
        $T_1 \leftarrow L_i$.SEARCHLEQ$(\mathrm{dist} - T)$
        $T_2 \leftarrow L_i$.SEARCHGEQ$(T + \mathrm{dist})$
                ▷ Search for the subtree such that $|\langle g_i, q - p \rangle| \geq T$
        if $m + |T_1| + |T_2| \leq 2\ell + 1$ **then**
            $S \leftarrow S \cup T_1 \cup T_2$
            $m \leftarrow m + |T_1| + |T_2|$
        else
            Add points from T_1 and T_2 to S until $|S| = 2\ell + 1$
            $m \leftarrow 2\ell + 1$
        end if
        $i \leftarrow i + 1$
    end while
    for $p \in S$ **do**
        if $\|p - q\|_2 \geq r/\overline{c}$ **then**
            return p
        else
            return "Fail"
        end if
    end for
end procedure

end data structure

Finally, in Alg. 26 and Alg. 27, we present our algorithm that solves AFN (see Definition 2.3.5). As AFN is the dual problem of approximate Min-IP, this algorithm could be used to solve approximate Min-IP.

Algorithm 26 AFN Algorithm: members, init, insert and delete

data structure AFN ▷ Theorem A.4.2

members
  $\varepsilon \in (0,1)$
  $\delta \in (0,1)$
  $s \in \mathcal{N}_+$ ▷ Number of data structures
  DFN $\text{dfn}_1, \text{dfn}_2, \ldots, \text{dfn}_s$
  $\text{bw} \in \mathbb{R}_+$ ▷ Boxwidth of all points
  SORTEDLIST $T_1, \ldots, T_d$ ▷ Max/min value for each dimension
end members

procedure INIT($A \in \mathbb{R}^{n \times d}, \overline{c} \in (1, \infty), \delta \in (0,1)$)
  $\varepsilon \leftarrow \overline{c} - 1$
  $s \leftarrow \Theta(\log \log(d/\delta))$
  dfn_i.INIT$(A, \overline{c})$ for all $i \in [s]$ ▷ Alg. 24
  for $j = 1 \to d$ **do**
    T_j.INIT$(A_{*,j})$ ▷ Alg. 23
  end for
  $\text{bw} \leftarrow \max_{j \in [d]} |T_j.\textsc{Max}() - T_j.\textsc{Min}()|$ ▷ 1d boxwidth
end procedure

procedure INSERT($p \in \mathbb{R}^d$)
  dfn_i.INSERT(p) for all $i \in [s]$ ▷ Alg. 24
  for $j = 1 \to d$ **do**
    T_j.INSERT(p_j) ▷ Alg. 23
  end for
  $\text{bw} \leftarrow \max_{j \in [d]} |T_j.\textsc{Max}() - T_j.\textsc{Min}()|$
end procedure

procedure DELETE($p \in \mathbb{R}^d$)
  dfn_i.DELETE(p) for all $i \in [s]$ ▷ Alg. 24
  for $j = 1 \to d$ **do**
    T_j.DELETE(p_j) ▷ Alg. 23
  end for
  $\text{bw} \leftarrow \max_{j \in [d]} |T_j.\textsc{Max}() - T_j.\textsc{Min}()|$
end procedure

end data structure

Algorithm 27 AFN Algorithm: query

data structure AFN ▷ Theorem A.4.2

procedure QUERY($q \in \mathbb{R}^d$)
  $\text{lo} \leftarrow \text{bw}/2$
  $\text{hi} \leftarrow \sqrt{d}/\varepsilon \cdot \text{bw}$
  Binary search over the range $[\text{lo}, \text{hi}]$ to search for $r \in \mathbb{R}_+$,
  with the predicate dfn_i.QUERY(q, r) for $i \in [s]$
  ▷ $\Theta(\log(d/\varepsilon\delta))$ rounds
  for $i = 1 \to s$ **do**
    $p \leftarrow \text{dfn}_i$.QUERY$(q, r)$
    if $p \neq$ "Fail" **then**
      return p
    end if
  end for
  return "Fail"
end procedure

end data structure

A.2 Success and Failure Probability of Random Projection

In this section, we analyze both the success and failure probability of random projection. To start with, we supply a technical lemma that upper bounds the failure probability that two points are far in the random direction but close in the original space.

Lemma A.2.1. *Let t be the solution to $e^{t^2/2}/t = 2n$ and $T = rt/\overline{c}$. Let $\widehat{p}$ be a point such that $\|\widehat{p} - q\|_2 < r/\overline{c}$. Then*

$$\Pr_{g \sim \mathcal{N}(0,I)}[|\langle g, \widehat{p}\rangle - \langle g, q\rangle| \geq T] \leq 1/n.$$

Proof. Observe that

$$\begin{aligned}
\Pr[|\langle g, \widehat{p}\rangle - \langle g, q\rangle| \geq T] &= \Pr\left[\frac{|\langle g, \widehat{p} - q\rangle|}{\|\widehat{p} - q\|_2} \geq T/\|\widehat{p} - q\|_2\right] \\
&\leq \Pr\left[\frac{|\langle g, \widehat{p} - q\rangle|}{\|\widehat{p} - q\|_2} \geq \frac{T}{r/\overline{c}}\right] \\
&= \Pr\left[\frac{|\langle g, \widehat{p} - q\rangle|}{\|\widehat{p} - q\|_2} \geq t\right] \\
&\leq 2\exp(-t^2/2)/t \\
&\leq 1/n.
\end{aligned}$$

The second step follows from $\|\widehat{p} - q\|_2 < r/\overline{c}$. For the fourth step, note that since standard Gaussian is 2-stable (Fact 2.1.1), we know that $\frac{\langle g, \widehat{p} - q\rangle}{\|\widehat{p} - q\|_2}$ follows a standard Gaussian distribution, so we can apply part 1 of the Gaussian concentration bound (Fact 2.1.2). □

Next, we provide a lower bound on the success probability that when two points are far away from each other in the random direction, then they are far away in the original space.

Lemma A.2.2. *Let t be the solution to $e^{t^2/2}/t = 2n$ and $T = rt/\overline{c}$. Let $\ell = O(n^{1/\overline{c}^2} \log^{(1-1/\overline{c})/2} n)$. Let p be a point such that $\|p - q\|_2 \geq r$. Then*

$$\Pr_{g \sim N(0,I)}[|\langle g, p\rangle - \langle g, q\rangle| \geq T] \geq 1/\ell.$$

Proof. The proof is similar to Lemma A.2.1, consider

$$\begin{aligned}
\Pr[|\langle g, p\rangle - \langle g, q\rangle| \geq T] = & \Pr\left[\frac{|\langle g, p-q\rangle|}{\|p-q\|_2} \geq \frac{T}{\|p-q\|_2}\right] \\
\geq & \Pr\left[\frac{|\langle g, p-q\rangle|}{\|p-q\|_2} \geq \frac{T}{r}\right] \\
= & \Pr\left[\frac{|\langle g, p-q\rangle|}{\|p-q\|_2} \geq \frac{t}{\overline{c}}\right] \\
\geq & 2B \cdot \exp(-(t/\overline{c})^2/2)/(t/\overline{c}) \\
= & \frac{2B \cdot \overline{c}(\exp(-t^2/2)/t)^{1/\overline{c}^2}}{t^{1-1/\overline{c}^2}} \\
= & \frac{2B\overline{c}}{n^{1/\overline{c}^2} t^{1-1/\overline{c}^2}}.
\end{aligned}$$

The second step follows from $\|p - q\|_2 \geq r$, the fourth step follows from Fact 2.1.2. Note that by picking $\ell = O(n^{1/\overline{c}^2} \log^{(1-1/\overline{c}^2)/2} n)$, we get our desired result. □

A.3 Guarantees of DFN Data Structure

In this section, we setup the theoretical guarantees of Alg. 24. We state and prove the following theorem regarding our DFN data structure.

Theorem A.3.1. *Let $P \subset \mathbb{R}^d$ be an n-point dataset and $\overline{c} > 1$. There exists a randomized dynamic data structure (Alg. 24, 25) that solves $\overline{c}$-DFN task (see Task A.1.1) using $O(n^{1+1/\overline{c}^2} \log n + dn^{1/\overline{c}^2} \log n)$ space with the following operations:*

- INIT*: Preprocess P in $O(n^{1+1/\overline{c}^2} \log^2 n + dn^{1/\overline{c}^2} \log n)$ time;*
- QUERY*: Given a point $q \in \mathbb{R}^d$ and $r > 0$, either outputs a point $\widehat{p} \in P$ such that $\|\widehat{p} - q\|_2 \geq r/\overline{c}$ with constant probability or outputs "Fail" in $O(n^{1/\overline{c}^2}(d + \log n) \log n)$ time;*
- INSERT*: Insert a point $p \in \mathbb{R}^d$ into the data structure in $O(n^{1/\overline{c}^2} \log^2 n)$ time;*
- DELETE*: Delete a point $p \in P$ in $O(n^{1/\overline{c}^2} \log^2 n)$ time.*

Proof. We prove four corresponding parts of Theorem A.3.1 accordingly.

Space: Storing the $\ell \times d$ standard Gaussian matrix takes $O(\ell d)$ space. Maintaining ℓ sorted list takes $O(\ell n)$ space. Thus, the total space is

$$O(\ell(d + n)) = O(n^{1+1/\overline{c}^2} \log^{(1-1/\overline{c})/2} n + dn^{1/\overline{c}^2} \log^{(1-1/\overline{c})/2} n).$$

Procedure INIT: By Alg. 24, the initiation needs to initialize an $\ell \times d$ standard Gaussian matrix, which takes $O(\ell d)$ time, processing all points into sorted lists takes $O(\ell(d + n \log n))$ time. Thus, the total time for INIT is

$$O(\ell(d + n \log n)) = O(n^{1+1/\overline{c}^2} \log n \log^{(1-1/\overline{c})/2} n + dn^{1/\overline{c}^2} \log^{(1-1/\overline{c})/2} n).$$

Procedure QUERY: We first show the correctness. Our goal is to prove that with constant probability, our data structure retrieves a pair (p, i) among the first $2\ell + 1$ pairs where each point p satisfies $|\langle g_i, p\rangle - \langle g_i, q\rangle| \geq T$ and at least one of the point p has the guarantee that $\|p - q\|_2 \geq r/\overline{c}$.

We first justify that picking $2\ell + 1$ pairs suffices for at least one point has desired distance guarantee, with constant probability. Let $Y_{\widehat{p},i}$ denote the event that a pair $(\widehat{p}, i) \in P \times [\ell]$ has the property that $\|\widehat{p} - q\|_2 < r/\overline{c}$ and $|\langle g_i, q\rangle - \langle g_i, \widehat{p}\rangle| \geq T$. Then

$$\begin{aligned}
\mathbb{E}[\sum_{(\widehat{p},i)} Y_{\widehat{p},i}] &= n\ell \cdot \Pr[Y_{\widehat{p},i}] \\
&\leq n\ell \cdot \frac{1}{n} \\
&= \ell.
\end{aligned}$$

The second step follows from Lemma A.2.1. Note that $\mathbb{E}[\sum_{(\widehat{p},i)} Y_{\widehat{p},i}]$ is the expected total number of such pairs, this means via a Markov bound, with the probability at least 1/2, there are no more than 2ℓ such pairs. Thus, if we retrieve exactly $2\ell + 1$ such pairs, there must be at least one pair (p, i) with $\|p - q\|_2 \geq r/\overline{c}$. Next, we analyze the failure probability when picking $2\ell + 1$ pairs. Note that for a point $p \in P$ with $|\langle g_i, p\rangle - \langle g_i, q\rangle| \geq T$, the probability that $\|p - q\|_2 < r/\overline{c}$ is at most $1 - 1/\ell$, due to Lemma A.2.2. This means the probability that *some i* among the first $2\ell + 1$ pairs has the property that $\|p - q\|_2 \geq r/\overline{c}$ is at least

$$1 - (1 - 1/\ell)^{2\ell+1} \geq 1 - 1/e,$$

this means we have a constant probability of success. Thus, our DFN data structure has a constant probability to output a point which is *not* within the distance of $r/\overline{c}$ from q.

For the running time, note that we do at most ℓ rounds of search, at each round, we search the sorted lists, so we pay a total of $O(\ell \log n)$ for searching the lists. Finally, we need to examine these $2\ell + 1$ pairs for their distances, this takes $O(\ell d)$ time. Therefore, the total running time is

$$O(n^{1/\overline{c}^2} \log n \log^{(1-1/\overline{c}^2)/2} n + dn^{1/\overline{c}^2} \log^{(1-1/\overline{c}^2)/2} n).$$

Procedure INSERT and DELETE: It is obvious that the running time of both procedures is $O(\ell(d + \log n))$, which is the same as the time of procedure QUERY. □

A.4 Guarantees of AFN Data Structure

In this section we provide an analysis for an AFN data structure implemented via DFN data structure. The idea is to use binary search to find the correct distance r. The search range is determined via the notion of *box width*.

Definition A.4.1. Given a dataset $P \subset \mathbb{R}^d$, we define the box width of P, denoted as $\mathrm{bw}(P)$ or bw if P is clear from context as

$$\mathrm{bw}(P) := \max_{i \in [d]} |\max_{p \in P}(p_i) - \min_{p \in P}(p_i)|.$$

Note that p_i denotes the i-th coordinate of point p.

We now proceed with the formal statement and proof.

Theorem A.4.2. *Let $P \subset \mathbb{R}^d$ be an n-point dataset, $\overline{c} > 1$, $r > 0$ and $\delta > 0$. Let $\varepsilon = \overline{c} - 1$. There exists a randomized dynamic data structure (Alg. 26, 27) that solves $(\overline{c} + \delta, r)$-AFN task using space $O((n^{1+1/\overline{c}^2} \log n + dn^{1/\overline{c}^2} \log n) \log\log(d/\varepsilon\delta) + dn)$ with the following operations:*

- INIT*: Preprocess P in $O((n^{1+1/\overline{c}^2} \log^2 n + dn^{1/\overline{c}^2} \log n) \log\log(d/\varepsilon\delta))$ time;*
- QUERY*: Given a point $q \in \mathbb{R}^d$, returns a $(\overline{c} + \delta)$-approximate furthest neighbor $p \in P$ with constant probability in $O(n^{1/\overline{c}^2}(d + \log n) \log n \log(d/\varepsilon\delta) \log\log(d/\varepsilon\delta))$ time;*
- INSERT*: Insert a point $p \in \mathbb{R}^d$ into the data structure in $O(n^{1/\overline{c}^2} \log^2 n \log\log(d/\varepsilon\delta) + d \log n)$ time;*
- DELETE*: Delete a point $p \in \mathbb{R}^d$ from the data structure in $O(n^{1/\overline{c}^2} \log^2 n \log\log(d/\varepsilon\delta) + d \log n)$ time.*

Proof. We start with the space complexity

Space: We note that there are $s = O(\log\log(d/\varepsilon\delta))$ DFN data structures to initialize, each data structure takes $O(n^{1+1/\overline{c}^2} \log n + dn^{1/\overline{c}^2} \log n)$ space. Moreover. the d different sorted lists for each dimension takes $O(dn)$ space. Therefore, the final space is

$$O((n^{1+1/\overline{c}^2} \log n + dn^{1/\overline{c}^2} \log n) \log\log(d/\varepsilon\delta) + dn)$$

Next, we prove four parts separately.

Procedure INIT: We note that there are $s = O(\log\log(d/\varepsilon\delta))$ DFN data structures to initialize, each data structure takes $O(n^{1+1/\overline{c}^2} \log^2 n + dn^{1/\overline{c}^2} \log n)$ time. To initialize d different sorted lists for each dimension, it takes $O(dn \log n)$ time. Finally, computing boxwidth bw takes $O(\log n)$ time. Thus, the total time in initialization phase is

$$O((n^{1+1/\overline{c}^2} \log^2 n + dn^{1/\overline{c}^2} \log n) \log\log(d/\varepsilon\delta)).$$

Procedure QUERY: We need to prove the runtime and correctness of the procedure. For the runtime, we note that QUERY makes $O(\log(d/\varepsilon\delta))$ calls to binary search with $O(\log\log(d/\varepsilon\delta))$ different data structures. Each call takes $O(n^{1/\overline{c}^2}(d + \log n) \log n)$ time by Theorem A.3.1. This completes the proof of runtime.

For correctness, note that for any query $q \in \mathbb{R}^d$, if p is its furthest neighbor then $\|p - q\|_2 \geq \mathrm{bw}/2$ since q must be further from one point defining boxwidth. On the other hand, if the distance from q to the center of box is at least $2\sqrt{d}/\varepsilon \cdot \mathrm{bw}$, then any point in P is a $(1 + \varepsilon)$-approximate furthest neighbor. To see this, note that any point $p \in P$ is at most $\sqrt{d}/2 \cdot \mathrm{bw}$ away from the center, so the nearest point from the box to the center is at least $(\frac{2}{\varepsilon} - \frac{1}{2})\sqrt{d} \cdot \mathrm{bw}$. On the other

hand, the furthest point on the box to q has a distance $(\frac{2}{\varepsilon}+\frac{1}{2})\sqrt{d}\cdot \mathrm{bw}$, it suffices to show that $(\frac{2}{\varepsilon}+\frac{1}{2})\sqrt{d}\cdot \mathrm{bw}/(1+\varepsilon) \leq (\frac{2}{\varepsilon}-\frac{1}{2})\sqrt{d}\cdot \mathrm{bw}$, since the furthest neighbor to q from dataset P must have distance smaller than $(\frac{2}{\varepsilon}+\frac{1}{2})\sqrt{d}\cdot \mathrm{bw}$. Note that

$$(\frac{2}{\varepsilon}+\frac{1}{2})/(1+\varepsilon) = \frac{4+\varepsilon}{2\varepsilon(1+\varepsilon)},$$

On the other hand,

$$\begin{aligned}\frac{2}{\varepsilon}-\frac{1}{2} &= \frac{4-\varepsilon}{2\varepsilon} \\ &= \frac{4+3\varepsilon-\varepsilon^2}{2\varepsilon(1+\varepsilon)}.\end{aligned}$$

Since $\varepsilon \in (0,1)$, we always have $3\varepsilon-\varepsilon^2 > \varepsilon$, as desired.

This gives a lower and upper bound on binary search, namely we search the range $[\mathrm{bw}/2, 2\sqrt{d}/\varepsilon \cdot \mathrm{bw}]$, hence, we need $O(\log \frac{d}{\varepsilon\delta})$ rounds to achieve a δ-precision solution. This leads to a $\overline{c}(1+\varepsilon)(1+\delta) = (1+\varepsilon)^2(1+\delta)$-approximation furthest neighbor. By picking ε as $\varepsilon/2$ and δ as $\delta/3$, this leads to a $(1+\varepsilon+\delta) = (\overline{c}+\delta)$-approximate furthest neighbor. Finally, to amplify the success probability of each query, we need to use $O(\log\log \frac{d}{\varepsilon\delta})$ different data structures. This completes the correctness analysis.

Procedure INSERT and DELETE: Both of these procedures require to insert or delete a point to s different data structures and update the sorted list for each dimension, then compute the new boxwidth. The insert/delete point step takes $O(n^{1/\overline{c}^2}\log^2 n \log\log \frac{d}{\varepsilon\delta})$ time and update the sorted list takes $O(d\log n)$ time. This completes the proof. □

Appendix B

Inverse Maintenance: The Algorithm

In this section, we show how to maintain the vector h inside the matrix, so that a certain block of the inverse directly gives the desired matrix-vector product Ph. We also give an algorithm that updates W and h in the data structure.

B.1 Maintaining h for Sketch on the Left

We show how to maintain h inside the matrix, especially for sketching on the left.

We start with the following simple lemma implied by Schur complement.

Lemma B.1.1. *Let $N \in \mathbb{R}^{n \times n}$ and $h \in \mathbb{R}^n$, then we have*

$$\left(\begin{bmatrix} N & h \\ \mathbf{0}_{1 \times n} & -1 \end{bmatrix} \right)^{-1} = \begin{bmatrix} N^{-1} & N^{-1}h \\ \mathbf{0}_{1 \times n} & -1 \end{bmatrix}$$

Proof. By Fact 2.6.1, we have that $D - CA^{-1}B = -1$ and so is its inverse, $A^{-1}B = N^{-1}h$, $CA^{-1} = 0$. So the top left block is N^{-1}, top right block is $N^{-1}h$, bottom left block is 0 and bottom right block is -1. □

Using the L matrix we defined in Definition 2.6.5, it is not hard to say that we can augment the matrix as

$$\begin{bmatrix} & & \begin{bmatrix} 0 \\ h \\ h \end{bmatrix} \\ & L & \\ & & \\ \mathbf{0}_{1 \times 3n+d} & & -1 \end{bmatrix}$$

so that the top right block of the inverse is our desired product.

Lemma B.1.2 (Sketch on the left, with vector)**.** *Let $\mathsf{R} \in \mathbb{R}^{n \times n}$ be a collection of sketching matrices, then we have*

$$\left(\begin{bmatrix} W^{-1} & A^\top & W^{-1/2} & 0 & 0 & 0 \\ A & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & -I & 0 & 0 & h \\ W^{-1/2} & 0 & 0 & -I & 0 & h \\ 0 & 0 & 0 & \mathsf{R} & -I & 0 \end{bmatrix} \right)^{-1} = \begin{bmatrix} \star & \mathsf{R}Ph \\ \star & \star \end{bmatrix}$$

B.2 Inverse Maintenance Algorithm

In this section, we present a data structure that maintains the big matrix (possibly with h), supports efficient update, reset and query the matrix-vector product. It is a key data structure to recover the results in [27, 55, 72].

For simplicity and generality, we consider the matrix without sketching.

Algorithm 28 Projection Maintenance via Inverse Maintenance. Note that we store the inverse of M, but only update it during the procedure RESET. For UPDATE, we assume the change happens for at most n^a entries in at most n^b columns. If we are not maintaining h, we can set h to 0.

data structure PROJECTIONMAINTENANCE ▷ Theorem B.2.8
members
  $U \in \mathbb{R}^{n \times n}$
  $A \in \mathbb{R}^{d \times n}$
  $M \in \mathbb{R}^{(3n+d+1) \times (3n+d+1)}$
  $a \in (0,1)$ ▷ Entry change threshold
  $b \in (0,1)$ ▷ Column change threshold
  $X, Y \in \mathbb{R}^{(3n+d+1) \times n^b}$ ▷ Each column of Y has only one nonzero entry
  $Q \in \mathbb{R}^{n^b \times n^b}$
  $N \in \mathbb{R}^{(3n+d+1) \times (3n+d+1)}$ ▷ Inverse of M
  $h \in \mathbb{R}^n$
end members
private:
procedure INIT($b \in (0,1), u_0 \in \mathbb{R}^n, \mathsf{R} \in \mathbb{R}^{n \times n}, A \in \mathbb{R}^{d \times n}$) ▷ Lemma B.2.4
  $b \leftarrow b, A \leftarrow A$
  $U \leftarrow \mathrm{diag}(u)$
  $\mathsf{R} \leftarrow \mathsf{R}$
  $$M \leftarrow \begin{bmatrix} U^{-1} & A^\top & U^{-1/2} & 0 & 0 \\ A & 0 & 0 & 0 & 0 \\ 0 & 0 & -I & 0 & h \\ (U^{-1/2})^\top & 0 & 0 & -I & h \\ 0 & 0 & 0 & 0 & -1 \end{bmatrix}$$
  $X, Y \leftarrow \mathbf{0}_{(3n+d+1) \times n^b}$
  $N \leftarrow M^{-1}$ ▷ Takes $O(n^\omega)$ time
  $Q \leftarrow \mathbf{0}_{n^b \times n^b}$
end procedure

Lemma B.2.1 (Update correctness)**.** *The output of* UPDATE$(u^{\mathrm{new}}, h^{\mathrm{new}})$ *in Algorithm 28 satisfies:*

$$Q = (I + Y^\top M^{-1} X)^{-1}$$

Proof. This follows directly from the invariant that $N = M^{-1}$. Note that by storing the quantity $(I + Y^\top M^{-1} X)^{-1}$, we can compute the query quickly, as we will show later. □

Lemma B.2.2 (Reset correctness)**.** *The output of* RESET$(X^{\mathrm{new}}, Y^{\mathrm{new}})$ *in Algorithm 29 satisfies:*

$$N = M^{-1}$$

Proof.

$$\begin{aligned} N^{\mathrm{new}} &= N - NXQY^\top N \\ &= M^{-1} - M^{-1}X(I + Y^\top M^{-1}X)^{-1}Y^\top M^{-1} \\ &= (M + XY^\top)^{-1} \\ &= (M^{\mathrm{new}})^{-1} \end{aligned}$$

where the first step follows from $N = M^{-1}$, the second step follows the matrix Woodbury formula, and the third step follows from Line 38 in Algorithm 28.

Note at the end of RESET, N and M are updated with the new value. This completes the proof. □

Lemma B.2.3 (Query correctness)**.** *The output of* QUERY$(I \subset [n], j \in [n])$ *in Algorithm 29 satisfies:*

$$v = (M^{-1})_{I,j} - (M^{-1}X(I + Y^\top M^{-1}X)^{-1}Y^\top M^{-1}e_j)_I$$

Proof. We have

$$\begin{aligned} v &= N_{I,j} - (NX)_I QY^\top N e_j \\ &= (M^{-1})_{I,j} - (M^{-1}XQY^\top M^{-1}e_j)_I \\ &= (M^{-1})_{I,j} - (M^{-1}X(I + Y^\top M^{-1}X)^{-1}Y^\top M^{-1}e_j)_I \end{aligned}$$

where the first step follows from $N = M^{-1}$, and the second step follows from $Q = (I + Y^\top NX)^{-1} = (I + Y^\top M^{-1}X)^{-1}$.

This completes the proof. □

Initialization Time

Lemma B.2.4 (Initialization Time)**.** *Taking a threshold $a \in (0,1)$, a vector $u_0 \in \mathbb{R}^n$, a sketching matrix $\mathsf{R} \in \mathbb{R}^{n \times n}$, a matrix $A \in \mathbb{R}^{d \times n}$ and a vector $h \in \mathbb{R}^n$ as input,* INIT *(Algorithm 28) operation takes $O(n^\omega)$ time to complete.*

Proof. The running time of INIT (Algorithm 28) operation consists of the following component:

- $N \leftarrow M^{-1}$ takes $O(n^\omega)$ to compute the matrix inverse of $M \in \mathbb{R}^{(3n+d+1)\times(3n+d+1)}$.

Therefore, we complete the proof. □

Update Time

Lemma B.2.5 (Update Time)**.** *Taking a vector $u^{\mathrm{new}} \in \mathbb{R}^n$ as input, the* UPDATE *(Algorithm 28) operation takes $O(n^{a+b} + n^{b \cdot \omega})$ time to complete.*

Proof. The running time of UPDATE operation consists of the following components:

- $Y^\top N$ takes $O(n^b)$ time to compute the matrix multiplication between a sparse matrix $Y^\top \in \mathbb{R}^{n^b \times (3n+d+1)}$ where each column of Y only has one non-zero entry and matrix $N \in \mathbb{R}^{(3n+d+1) \times (3n+d+1)}$.
- $(Y^\top N) \cdot X$ takes $O(n^{a+b}))$ time to compute the matrix multiplication between $Y^\top N \in \mathbb{R}^{n^b \times (3n+d+1)}$ and $X \in \mathbb{R}^{(3n+d+1) \times n^b}$.
- $Q \leftarrow S^{-1}$ takes $o(n^{b\omega})$ time to compute the matrix inverse of $S \in \mathbb{R}^{n^b \times n^b}$.

Therefore, we have:

$$\begin{aligned} & O(n^b) + O(n^{a+b}) + O(n^{b\omega}) \\ = & O(n^{a+b}) + O(n^{b\omega}) \end{aligned}$$

This completes the proof. □

Query Time

Lemma B.2.6 (Query Time)**.** *Taking $I \subset [n]$ and an index $j \in [n]$ as input, the* QUERY *(Algorithm 29) operation takes $O(n^{2b} + |I| \cdot n^a)$ time to complete.*

Proof. The running time of QUERY operation consists of the following components:

- $v_1 \leftarrow Y^\top N e_j$ takes $O(n^b)$ time to compute $Y^\top N e_j$ where Y only contains one non-zero entry per column and e_j only has non-zero entry on jth element.
- $v_2 \leftarrow Q v_1$ takes $O(n^{2b})$ time to compute the matrix vector multiplication between $Q \in \mathbb{R}^{n^b \times n^b}$ and $v_1 \in \mathbb{R}^{n^b}$.
- $L \leftarrow (NX)_I$ takes $O(|I| \cdot n^a)$ time to compute the matrix multiplication between $N \in \mathbb{R}^{(3n+d+1) \times (3n+d+1)}$ and $X \in \mathbb{R}^{(4n+d) \times n^b}$ with n^a nonzero entries, for $|I|$ rows.
- $v_3 \leftarrow L v_2$ takes $O(|I| \cdot n^b)$ time to compute the matrix vector multiplication between $L \in \mathbb{R}^{|I| \times n^b}$ and $v_2 \in \mathbb{R}^{n^b}$.

Therefore, we have:

$$\begin{aligned} & O(n^b) + O(n^{2b}) + O(|I| \cdot n^a) + O(|I| \cdot n^b) \\ = & O(n^{2b} + |I| \cdot n^a) \end{aligned}$$

This completes our proof. □

Reset Time

Lemma B.2.7 (Reset Time)**.** *Let $X^{\mathrm{new}}, Y^{\mathrm{new}} \in \mathbb{R}^{(4n+d)\times n^c}$ be the inputs to* RESET *(Algorithm 29), then* RESET *takes $O(\mathcal{T}_{\mathrm{mat}}(n^c, n, n))$ time to complete.*

Proof. For the simplicity of notation, we use X and Y to denote X^{new} and Y^{new}.

The running time of RESET operation consists of the following components:

- $L_1 \leftarrow Y^\top N$ takes $O(\mathcal{T}_{\mathrm{mat}}(n^c, n, n))$ time to compute the multiplication between a $n^c \times (3n+d+1)$ matrix and a $(3n+d+1) \times (3n+d+1)$ matrix.
- $Q \leftarrow (I + Y^\top M^{-1} X)^{-1}$, it takes $O(\mathcal{T}_{\mathrm{mat}}(n^c, n, n))$ time to compute the product $Y^\top M^{-1} X$ and $O(n^{c\cdot\omega})$ time to compute the inverse.
- $L_2 \leftarrow QL_1$ takes $O(\mathcal{T}_{\mathrm{mat}}(n^c, n^c, n))$ time to compute the matrix multiplication between $Q \in \mathbb{R}^{n^c \times n^c}$ and $L_1 \in \mathbb{R}^{n^c \times (3n+d+1)}$.
- $L_3 \leftarrow NXL_2$ takes $O(\mathcal{T}_{\mathrm{mat}}(n^c, n, n))$ time to compute the matrix multiplication between $N \in \mathbb{R}^{(3n+d+1)\times(3n+d+1)}$ and $X \in \mathbb{R}^{(3n+d+1)\times n^c}$ and $O(\mathcal{T}_{\mathrm{mat}}(n^c, n, n))$ time to compute the matrix multiplication between $NX \in \mathbb{R}^{(3n+d+1)\times n^c}$ and $L_2 \in \mathbb{R}^{n^c \times (3n+d+1)}$. Therefore, the total time for $L_3 \leftarrow NXL_2$ is $O(\mathcal{T}_{\mathrm{mat}}(n^c, n, n))$.

Therefore, we have:

$$\begin{aligned} & O(\mathcal{T}_{\mathrm{mat}}(n^c, n, n)) + O(\mathcal{T}_{\mathrm{mat}}(n^c, n^c, n)) \\ = {} & O(\mathcal{T}_{\mathrm{mat}}(n^c, n, n)) \end{aligned}$$

This completes the proof. □

Main Result

Theorem B.2.8. *Let $T = \sqrt{n}$ and $m = 3n + d + 1$.*

Let $A \in \mathbb{R}^{d\times n}$ of rank d and let $U \in \mathbb{R}^{n\times n}$ be a diagonal matrix with non-zero diagonal entries and $h \in \mathbb{R}^n$. There exists a data structure PROJECTIONMAINTENANCE *(Algorithm 28 and 29) which supports the following operations:*

- INIT$(b \in (0,1), u_0 \in \mathbb{R}^n, A \in \mathbb{R}^{d\times n}, h \in \mathbb{R}^n)$*: Given threshold $b \in (0,1)$, a vector $u_0 \in \mathbb{R}^n$, a sketching matrix $\mathsf{R} \in \mathbb{R}^{n\times n}$, a matrix $A \in \mathbb{R}^{d\times n}$ and a vector $h \in \mathbb{R}^n$ as input,* INIT *(Algorithm 28) operation runs in $O(n^\omega)$ time.*
- UPDATE$(u^{\mathrm{new}} \in \mathbb{R}^n, h^{\mathrm{new}} \in \mathbb{R}^n)$*: Given a vector $u^{\mathrm{new}} \in \mathbb{R}^n$ and a vector $h^{\mathrm{new}} \in \mathbb{R}^n$ as input, the* UPDATE *(Algorithm 28) operation runs $O(n^{a+b} + n^{b\cdot\omega})$ time suppose u^{new} has at most n^b nonzero entries and $\|h^{\mathrm{new}}\|_0 + \|u^{\mathrm{new}}\|_0$ is at most n^a.*
- QUERY$(I \subset [n], j \in [n])$*: Given $I \subset [n]$ and an index $j \in [n]$ as input, the* QUERY *(Algorithm 29) operation runs in $O(n^{2b} + |I| \cdot n^a)$ time to return the rows of inverse in set I and column j.*
- RESET$(X^{\mathrm{new}}, Y^{\mathrm{new}})$*: Give matrices $X^{\mathrm{new}}, Y^{\mathrm{new}} \in \mathbb{R}^{n\times n^c}$,* RESET *(Algorithm 29) operations runs in $O(\mathcal{T}_{\mathrm{mat}}(n^c, n, n))$ time.*

Proof. The correctness follows from combining Lemma B.2.1, Lemma B.2.2 and Lemma B.2.3.

The running time follows from combining Lemma B.2.4, Lemma B.2.5, Lemma B.2.6 and Lemma B.2.7. □

Remark B.2.9. *Using this simple Schur complement-based inverse maintenance, we can simplify the projection maintenance a bit and unify it using the data structure. However, this is not enough, and even not the key to obtain the speed up for [27, 55, 72], rather, coordinate-wise embedding, as we have demonstrated earlier, is the most important ingredient for improving the running time.*

Algorithm 29 Query and Reset

data structure PROJECTIONMAINTENANCE ▷ Theorem B.2.8

procedure UPDATE($u^{\mathrm{new}} \in \mathbb{R}^n, h^{\mathrm{new}} \in \mathbb{R}^n$) ▷ Lemma B.2.1 and Lemma B.2.5

▷ To run this procedure, we require that u^{new} has at most n^b non-zero entries

  if $\|u^{\mathrm{new}}\|_0 > n^b$ **then**

    return error

  end if

  $a \leftarrow \|u^{\mathrm{new}}\|_0 + \|h^{\mathrm{new}}\|_0$

  $\Delta \leftarrow \begin{bmatrix} (U^{\mathrm{new}})^{-1} & 0 & (U^{\mathrm{new}})^{-1/2} & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & h^{\mathrm{new}} \\ ((U^{\mathrm{new}})^{-1/2})^\top & 0 & 0 & 0 & h^{\mathrm{new}} \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}$

  Let $\Delta = XY^\top$ where $X \in \mathbb{R}^{(3n+d+1)\times n^b}$ and $Y \in \mathbb{R}^{(3n+d+1)\times n^b}$ ▷ X consists of nonzero entries, Y is a column selection matrix

  ▷ Each row of Y has only 1 nonzero entry

  $X \leftarrow X, Y \leftarrow Y$

  $S \leftarrow I + Y^\top NX$ ▷ Compute $Y^\top N$ takes $O(n^b)$ time and $Y^\top NX$ takes $O(n^{a+b})$ time

  $Q \leftarrow S^{-1}$ ▷ Takes $O(n^{b\cdot\omega})$ time

end procedure

procedure QUERY($I \subset [n], j \in [n]$) ▷ Lemma B.2.3 and Lemma B.2.6

▷ Compute query using matrix Woodbury formula

▷ Return $(M^{-1})_{I,j} - (M^{-1}X(I + Y^\top M^{-1}X)^{-1}Y^\top M^{-1}e_j)_I$

  $v_1 \leftarrow Y^\top \cdot N \cdot e_j$ ▷ Takes $O(n^b)$ time, $v_1 \in \mathbb{R}^{n^b}$

  $v_2 \leftarrow Qv_1$ ▷ Takes $O(n^{2b})$ time

  $L \leftarrow (NX)_I$ ▷ Takes $O(|I| \cdot n^a)$ time, $L \in \mathbb{R}^{|I|\times n^b}$

  $v_3 \leftarrow Lv_2$ ▷ Takes $O(|I| \cdot n^b)$ time

  $v \leftarrow N_{I,j} - v_3$

  return v

end procedure

procedure RESET($X^{\mathrm{new}} \in \mathbb{R}^{(3n+d+1)\times k}, Y^{\mathrm{new}} \in \mathbb{R}^{(3n+d+1)\times k}$) ▷ Lemma B.2.2 and Lemma B.2.7

▷ To run this procedure, we require that $k \geq n^a$

▷ Compute M^{-1} explicitly by matrix Woodbury formula

▷ $(M + XY^\top)^{-1} = M^{-1} - M^{-1}X(I + Y^\top M^{-1}X)^{-1}Y^\top M^{-1}$

▷ Let $k = n^c$

  $L_1 \leftarrow (Y^{\mathrm{new}})^\top N$ ▷ Takes $O(\mathcal{T}_{\mathrm{mat}}(n^c, n, n))$ time

  $Q \leftarrow (I + (Y^{\mathrm{new}})^\top M^{-1}X^{\mathrm{new}})^{-1}$ ▷ Takes $O(\mathcal{T}_{\mathrm{mat}}(n, n, n^c))$ time

  $L_2 \leftarrow QL_1$ ▷ Takes $O(\mathcal{T}_{\mathrm{mat}}(n^c, n^c, n))$ time

  $L_3 \leftarrow NX^{\mathrm{new}}L_2$ ▷ Takes $O(\mathcal{T}_{\mathrm{mat}}(n, n, n^c))$ time

  $L \leftarrow N - L_3$

  $N \leftarrow L$

  $M \leftarrow M + X^{\mathrm{new}}(Y^{\mathrm{new}})^\top$

  $X \leftarrow 0, Y \leftarrow 0$

end procedure

end data structure

Appendix C

Tensor Circulant Transform

In this section, we introduce a novel sketching matrix for tensors: the Tensor Circulant Gaussian Transform.

C.1 Definitions and Basic Facts

We start with the definitions for Circulant Transform, Tensor Circulant Transform and some basic useful facts regarding tensor product of matrices.

Definition C.1.1 (Circulant Transform)**.** Let $x \in \mathbb{R}^d$ be a random vector, whose elements are i.i.d. Rademacher random variables. Also, let $P \in \mathbb{R}^{m \times d}$ be a random matrix in which each row contains a 1 at a uniformly distributed coordinate and zeros elsewhere. Let $G \in \mathbb{R}^{d \times d}$ be a circulant matrix generated by x and $D \in \mathbb{R}^{d \times d}$ be a diagonal matrix whose diagonal elements are i.i.d. Rademacher random variables. Then, the Circulant Transform is defined as follows:

$$S = PGD,$$

where $S : \mathbb{R}^d \to \mathbb{R}^m$.

Note that since G is a circulant matrix, multiplying G with vector $x \in \mathbb{R}^d$ only takes time $O(d \log d)$. Therefore, apply the matrix S to vector $x \in \mathbb{R}^d$ takes time $O(d \log d + m)$.

Definition C.1.2 (Tensor Circulant Transform)**.** Let $x \in \mathbb{R}^d$ be a random vector, whose elements are i.i.d. Rademacher random variables. Also, let $P \in \mathbb{R}^{m \times d^2}$ be a random matrix in which each row contains a 1 at a uniformly distributed coordinate and zeros elsewhere and let $G \in \mathbb{R}^{d \times d}$ be a circulant matrix generated by x. Let $D_1 \in \mathbb{R}^{d \times d}$ and $D_2 \in \mathbb{R}^{d \times d}$ be two independent diagonal matrices whose diagonal elements are i.i.d. Rademacher random variables. Then, the tensor Circulant Transform T is defined as follows:

$$T = P \cdot (GD_1 \times GD_2),$$

where $T : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}^m$.

We recall a simple fact called *mixed product property*:

Fact C.1.3. *Let A, B, C, D be conforming matrices, then we have*

$$(A \times B) \cdot (C \times D) = (AC) \times (BD).$$

Using the mixed product property, we have that, for $A, B \in \mathbb{R}^{d\times d}$ and $x, y \in \mathbb{R}^d$,

$$(A \times B)(x \otimes y) = (Ax) \otimes (By).$$

Therefore, the product $T(x \otimes y)$ can be written as

$$\begin{aligned} T(x \otimes y) &= P(GD_1 \times GD_2)(x \otimes y) \\ &= P((GD_1x) \otimes (GD_2y)), \end{aligned}$$

note that computing GD_1x and GD_2y takes $O(d \log d)$ time. After that, we don't directly form $(GD_1x) \otimes (GD_2y)$, rather, P is a selection matrix that picks out m indices, so we use P to locate the indices of the desired entries, and compute each of the entry. Hence, applying T to a tensor product $x \otimes y$ only takes time $O(d \log d + m)$.

C.2 Circulant Transform and Tensor Circulant Transform: Strong JL Moment Property

We show that both Circulant Transform and Tensor Circulant Transform satisfy the so-called *Strong JL Moment Property*. Strong JL Moment Property is one of the core properties that can show the sketching matrix has subspace embedding property [67].

Definition C.2.1 (Strong JL Moment Property [2])**.** For every $\varepsilon, \delta \in [0, 1]$, we say a distribution over random matrices $M \in \mathbb{R}^{m\times d}$ has the Strong (ε, δ)-JL Moment Property when

$$\|\|Mx\|_2^2 - 1\|_{L^t} \leq \frac{\varepsilon}{e}\sqrt{\frac{t}{\log(1/\delta)}}$$

and

$$\mathbb{E}\left[\|Mx\|_2^2\right] = 1$$

for all $x \in \mathbb{R}^d$, $\|x\|_2 = 1$ and every integer $t \leq \log(1/\delta)$.

To prove that Circulant Transform and Tensor Circulant Transform satisfy the strong JL moment property. We will do this by proving that a more general class of matrices satisfies the strong JL moment property. More precisely, let $k \in \mathbb{Z}_{>0}$ be a positive integer and $(D^{(i)})_{i\in[k]} \in \prod_{i\in[k]} \mathbb{R}^{d_i\times d_i}$ be independent matrices, each with diagonal entries given by independent Rademacher variables. Let $d = \prod_{i\in[k]} d_i$, and $P \in \{0, 1\}^{m\times d}$ be a random sampling matrix in which each row contains exactly one uniformly distributed nonzero element which has value one. Then we prove that the matrix $M = \frac{1}{\sqrt{m}}PG(D_1 \times \cdots \times D_k)$ satisfies the strong JL moment property, where G is $d \times d$ circulant matrix which is generated by a random vector whose elements are Rademacher variables. If $k = 1$ then M is just a Circulant Transform, and if $k = 2$ the M is a Tensor Circulant Transform.

In order to prove this result we need a couple of lemmas. The first lemma can be seen as a version of Khintchine's inequality for higher order chaos.

Lemma C.2.2 (Khintchine's inequality for higher order chaos)**.** *Let $t \geq 1$, $k \in \mathbb{Z}_{>0}$, and $(\sigma^{(i)})_{i\in[k]} \in \Pi_{i\in[k]}\mathbb{R}^{d_i}$ be independent vectors each satisfying the Khintchine inequality $\|\langle\sigma^{(i)}, x\rangle\|_{L^t} \leq C_t\|x\|_2$ for $t \geq 1$ and any vector $x \in \mathbb{R}^{d_i}$. Let $(a_{i_1,\ldots,i_k})_{i_1\in[d_j],\ldots,i_k\in[d_k]}$ be a tensor in $\mathbb{R}^{d_1\times\cdots\times d_k}$, then*

$$\|\sum_{i_1\in[d_1],\ldots,i_k\in[d_k]}(\prod_{j\in[k]}\sigma_{i_j}^{(j)})a_{i_1,\ldots,i_k}\|_{L^t} \leq C_t^k\cdot(\sum_{i_1\in[d_1],\ldots,i_k\in[d_k]}a_{i_1,\ldots,i_k}^2)^{1/2}$$

for $t \geq 1$. Or, considering $a \in \mathbb{R}^{d_1\ldots d_k}$ a vector, then simply $\|\langle\sigma^{(1)}\otimes\cdots\otimes\sigma^{(k)}, a\rangle\|_{L^t} \leq C_t^k\|a\|_2$, for $t \geq 1$.

Proof. The proof will be by induction on k. For $k = 1$ then the result is by assumption. So assume that the result is true for every value up to $k-1$. Let $B_{i_1,\ldots,i_{k-1}} = \sum_{i_k\in[d_k]}\sigma_{i_k}^{(k)}a_{i_1,\ldots,i_k}$. We then pull it out of the left hand term in the theorem:

$$\begin{aligned}
\|\sum_{i_1\in[d_1],\ldots,i_c\in[d_c]}(\prod_{j\in[k]}\sigma_{i_j}^{(j)})a_{i_1,\ldots,i_k}\|_{L^t} &= \|\sum_{i_1\in[d_1],\ldots,i_{k-1}\in[d_{k-1}]}(\prod_{j\in[k-1]}\sigma_{i_j}^{(j)})B_{i_1,\ldots,i_{k-1}}\|_{L^t}\\
&\leq C_t^{k-1}\cdot\|(\sum_{i_1\in[d_1],\ldots,i_{k-1}\in[d_{k-1}]}B_{i_1,\ldots,i_{k-1}}^2)^{1/2}\|_{L^t}\\
&= C_t^{k-1}\cdot\|\sum_{i_1\in[d_1],\ldots,i_{k-1}\in[d_{k-1}]}B_{i_1,\ldots,i_{k-1}}^2\|_{L^{t/2}}^{1/2}\\
&\leq C_t^{k-1}\cdot(\sum_{i_1\in[d_1],\ldots,i_{k-1}\in[d_{k-1}]}\|B_{i_1,\ldots,i_{k-1}}^2\|_{L^{t/2}})^{1/2}\\
&= C_p^{k-1}\cdot(\sum_{i_1\in[d_1],\ldots,i_{k-1}\in[d_{k-1}]}\|B_{i_1,\ldots,i_{k-1}}^2\|_{L^t}^2)^{1/2}
\end{aligned}$$

where the first step follows from the definition of $B_{i_1,\ldots,i_{k-1}}$, the second step follows from the inductive hypothesis, the third step follows from the Definition 2.1.3, the fourth step follows from the triangle inequality, and the last step follows from the Definition 2.1.3. Now $\|B_{i_1,\ldots,i_{k-1}}\|_{L^t}^2 \leq C_t^2\sum_{i_c\in[d_c]}a_{i_1,\ldots,i_c}^2$ by Khintchine's inequality, which finishes the induction step and hence the proof. □

The next lemma we will be using is a type of Rosenthal inequality, but which mixes large and small moments in a careful way. It bears similarity to the one sided bound in [15] (Theorem 15.10) derived from the Efron Stein inequality, and the literature has many similar bounds, but we still include a proof here based on first principles.

Lemma C.2.3 (properties of random variables with t-moment)**.** *There exists a universal constant L, such that, for $t \geq 1$ if $X_1,\ldots,X_k$ are independent non-negative random variables with t-moment, then*

$$\|\sum_{i\in[k]}(X_i - \mathbb{E}[X_i])\|_{L^t} \leq L\cdot\left(\sqrt{t}\cdot\|\max_{i\in[k]}X_i\|_{L^t}^{1/2}\cdot(\sum_{i\in[k]}\mathbb{E}[X_i])^{1/2} + t\cdot\|\max_{i\in[k]}X_i\|_{L^t}\right)$$

Proof. Throughout these calculations L_1, L_2 and L_3 will be universal constants.

$$\begin{aligned}
\|\sum_{i\in[k]}(X_i - \mathbb{E}[X_i])\|_{L^t} &\leq L_1\|\sum_{i\in[k]}\sigma_i X_i\|_{L^t} \\
&\leq L_2\sqrt{t}\cdot\|\sum_{i\in[k]}X_i^2\|_{L^{t/2}}^{1/2} \\
&\leq L_2\sqrt{t}\cdot\|\max_{i\in[k]}X_i\cdot\sum_{i\in[k]}X_i\|_{L^{t/2}}^{1/2} \\
&\leq L_2\sqrt{t}\cdot\|\max_{i\in[k]}X_i\|_{L^t}^{1/2}\cdot\|\sum_{i\in[k]}X_i\|_{L^t}^{1/2} \\
&\leq L_2\sqrt{t}\cdot\|\max_{i\in[k]}X_i\|_{L^t}^{1/2}\cdot\Big((\sum_{i\in[k]}\mathbb{E}[X_i])^{1/2} + L_2\|\sum_{i\in[k]}(X_i-\mathbb{E}[X_i])\|_{L^t}^{1/2}\Big)
\end{aligned}$$

where the first step follows from symmetrization of X_i, the second step follows from Khintchine's inequality, the third step follows from Non-negativity of X_i, and the fourth step follows from Cauchy-Schwartz inequality, and the last step follows from triangle inequality.

Now let C, B, A be defined as follows:

$$C = \|\sum_{i\in[k]}(X_i - \mathbb{E}[X_i])\|_{L^t}^{1/2}$$

$B = L_2(\sum_{i\in[k]}\mathbb{E}[X_i])^{1/2}$, and $A = \sqrt{t}\|\max_{i\in[k]}X_i\|_{L^t}^{1/2}$. then we have shown $C^2 \leq A(B+C)$. That implies C is smaller than the largest of the roots of the quadratic. Solving this quadratic inequality gives $C^2 \leq L_3(AB + A^2)$ which is the result. □

We can now prove that Circulant Transform and Tensor Circulant Transform have the Strong JL Moment Property.

Theorem C.2.4 (Circulant Transform satisfies the Strong JL Moment Property)**.** *There exists a universal constant L, such that, the following holds. Let $k \in \mathbb{Z}_{>0}$, and $(D^{(i)})_{i\in[k]} \in \prod_{i\in[k]}\mathbb{R}^{d_i\times d_i}$ be independent diagonal matrices with independent Rademacher variables. Define $d = \prod_{i\in[k]}d_i$ and $D = D_1 \times D_2 \times \ldots D_c \in \mathbb{R}^{d\times d}$. Let $P \in \mathbb{R}^{m\times d}$ be an independent sampling matrix which samples exactly one coordinate per row, and define $M = PGD$ where G is a $d\times d$ circulant matrix which is generated by a random vector y whose elements are i.i.d. Rademacher random variables. Let $x \in \mathbb{R}^d$ be any vector with $\|x\|_2 = 1$ and $t \geq 1$, then*

$$\left\|\frac{1}{m}\|PGD\|_2^2 - 1\right\|_{L^t} \leq L(\sqrt{\frac{tr^k}{m}} + \frac{tr^k}{m}),$$

where $r = \max\{t, \log m\}$.

There exists a universal constant L', such that, setting $m = \Omega(\varepsilon^{-2}\log(1/\delta)(L'\log(1/\varepsilon\delta)^k)$, we get that $\frac{1}{\sqrt{m}}PGD$ has Strong (ε,δ)-JL Moment Property.

Proof. Throughout the proof C_1, C_2 and C_3 will denote universal constants.

For every $i \in [m]$ we let P_i be the random variable that says which coordinate the i-th row of P samples, and we define the random variable $Z_i = M_i x = G_{P_i} D x$. We note that since the variables $(P_i)_{i\in[m]}$ are independent then the variables $(Z_i)_{i\in[m]}$ are conditionally independent given D, that is, if we fix D then $(Z_i)_{i\in[m]}$ are independent.

Then, we could get the following inequality:

$$\begin{aligned}
&\|\frac{1}{m}\sum_{i\in[m]} Z_i^2 - 1\|_{L^t} \\
&= \|(\mathbb{E}[(\frac{1}{m}\sum_{i\in[m]} Z_i^2 - 1) \mid D])^{1/t}\|_{L^t} \\
&\leq C_1\|\frac{\sqrt{t}}{m}\cdot(\mathbb{E}[(\max_{i\in[m]} Z_i^2) \mid D])^{1/(2t)}\cdot(\sum_{i\in[m]}\mathbb{E}[Z_i^2 \mid D])^{1/2} + \frac{t}{m}\cdot(\mathbb{E}[(\max_{i\in[m]} Z_i^2)^t \mid D])^{1/t}\|_{L^t} \\
&\leq C_1\frac{\sqrt{t}}{m}\cdot\|(\mathbb{E}[(\max_{i\in[m]} Z_i^2) \mid D])^{1/(2t)}\cdot(\sum_{i\in[m]}\mathbb{E}[Z_i^2 \mid D])^{1/2}\|_{L^t} + C_1\frac{t}{m}\cdot\|\max_{i\in[m]} Z_i^2\|_{L^t} \\
&\leq C_1\frac{\sqrt{t}}{m}\cdot\|\max_{i\in[m]} Z_i^2\|_{L^t}^{1/2}\cdot\|\sum_{i\in[m]}\mathbb{E}[Z_i^2 \mid D]\|_{L^t}^{1/2} + C_1\frac{t}{m}\cdot\|\max_{i\in[m]} Z_i^2\|_{L^t}
\end{aligned}$$

where the first step follows from Definition 2.1.3, the second step follows from Lemma C.2.3, the third step follows from triangle inequality, and the last step follows from Cauchy-Schwartz inequality.

Due to the reason that G is generated by a random vector y whose elements are i.i.d. Rademacher random variables, we could obtain that

$$\begin{aligned}
\mathbb{E}[Z_i^2|D] &= \sum_{\sigma\in\{-1,+1\}^d} p_\sigma\cdot(\langle x,\sigma\rangle)^2 \\
&= \frac{(x_1+x_2+\cdots+x_d)^2}{2^d} + \frac{(x_1+x_2+\cdots-x_d)^2}{2^d} + \cdots + \frac{(-x_1-x_2-\cdots-x_d)^2}{2^d} \\
&= x_1^2+x_2^2+\cdots+x_d^2 \\
&= \|x\|_2^2
\end{aligned}$$

We could get that

$$\sum_{i\in[m]}\mathbb{E}[Z_i^2 \mid D] = \sum_{i\in[m]}\|x\|_2^2 = m.$$

where the second step follows from $\|x\|_2^2 = 1$.

To bound $\|\max_{i\in[m]} Z_i^2\|_{L^t}$, we could show

$$\|Z_i^2\|_{L^r} = \|G_{P^i} D x\|_{L^{2r}}^2 = \|Dx\|_{L^{2r}}^2 \leq r^k\|x\|_2^2.$$

where the first step follows from the definition of Z_i, the second step follows from G is generated by the random vector y whose elements are i.i.d. Rademacher random variables, and the last step follows from Lemma C.2.2.

We then bound the maximum using a sufficiently high powered sum:

$$\|\max_{i\in[m]} Z_i^2\|_{L^t} \leq \|\max_{i\in[m]} Z_i^2\|_{L^r} \leq (\sum_{i\in[m]} \|Z_i^2\|_{L^r}^r)^{1/r} \leq m^{1/r} r^k \|x\|_2^2 \leq e r^k,$$

where the first step follows from Definition 2.1.3, the second step follows from Z_i^2 is non-negative, and the last inequality follows from $r \geq \log m$. This gives us that

$$\|\frac{1}{m}\sum_{i\in[m]} Z_i^2 - \|x\|_2^2\|_{L^t} \leq C_2\sqrt{\frac{tr^k}{m}} + C_2\frac{tr^k}{m}$$

which finishes the first part of the proof.

We want to choose m as follows $m = 4e^2C_2^2\varepsilon^{-2}\cdot\log(1/\delta)\cdot(C_3\log(1/(\delta\varepsilon)))^k$.

According to the above choice of m we know following condition for r is holding: $r \leq C_3\log(1/(\delta\varepsilon))$.

Hence $m \geq 4e^2C_2^2\varepsilon^{-2}\cdot\log(1/\delta)\cdot r^k$. We then get that

$$\begin{aligned}\|\|PGDx\|_2^2 - 1\|_{L^t} &\leq C_2(\frac{tr^k}{4e^2C_2^2\varepsilon^{-2}\log(1/\delta)r^k})^{1/2} + C_2\frac{tr^k}{4e^2C_2^2\varepsilon^{-2}\log(1/\delta)r^k} \\ &\leq \frac{\varepsilon}{e}\sqrt{\frac{t}{\log(1/\delta)}}\end{aligned}$$

for all $1 \leq t \leq \log(1/\delta)$ which finishes the proof. □

Bibliography

[1] Pankaj K Agarwal, Jiří Matoušek, and Subhash Suri. Farthest neighbors, maximum spanning trees and related problems in higher dimensions. *Computational Geometry*, 1(4): 189–201, 1992. 1.3, 2.3

[2] Thomas D. Ahle, Michael Kapralov, Jakob Bæk Tejs Knudsen, Rasmus Pagh, Ameya Velingker, David P. Woodruff, and Amir Zandieh. Oblivious sketching of high-degree polynomial kernels. In *Proceedings of the 2020 ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 141–160, 2020. 4.5.8, C.2.1

[3] Zeyuan Allen-Zhu, Zhenyu Liao, and Lorenzo Orecchia. Spectral sparsification and regret minimization beyond matrix multiplicative updates. STOC '15, 2015. 1, 1.2, 3.1.4, 3.1.4, 3.1.4

[4] Zeyuan Allen-Zhu, Yin Tat Lee, and Lorenzo Orecchia. Using optimization to obtain a width-independent, parallel, simpler, and faster positive sdp solver. SODA '16, 2016. 3.1.4

[5] Zeyuan Allen-Zhu, Yuanzhi Li, and Zhao Song. A convergence theory for deep learning via over-parameterization. In *ICML*, 2019. 1, 4.7.9, 4.7.2, 4.7.10, 4.8

[6] Zeyuan Allen-Zhu, Yuanzhi Li, and Zhao Song. On the convergence rate of training recurrent neural networks. In *NeurIPS*, 2019. 1

[7] Zeyuan Allen-Zhu, Yuanzhi Li, Aarti Singh, and Yining Wang. Near-optimal discrete optimization for experimental design: A regret minimization approach. *Mathematical Programming*, pages 1–40, 2020. 1, 1.1.1, 1.1.1, 1.2, 1.2, 3, 3.3, 3.3.2, 3.3.7, 3.3.8, 3.3.9, 3.3.10

[8] Josh Alman and Virginia Vassilevska Williams. A refined laser method and faster matrix multiplication. In *Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 522–539. SIAM, 2021. 1, 2.5.1

[9] Nima Anari, Shayan Oveis Gharan, Amin Saberi, and Nikhil Srivastava. Approximating the largest root and applications to interlacing families. In *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1015–1028. SIAM, 2018. 1.4

[10] Sanjeev Arora and Satyen Kale. A combinatorial, primal-dual approach to semidefinite programs. *J. ACM*, 2016. 3.1.4

[11] Haim Avron, Huy L. Nguyen, and David P. Woodruff. Subspace embeddings for the polynomial kernel. In *NeurIPS*, 2014. 2.3.3, 4.5.1, 4.5.4

[12] Joshua Batson, Daniel A Spielman, and Nikhil Srivastava. Twice-ramanujan sparsifiers. *SIAM Journal on Computing*, 41(6):1704–1721, 2012. 1, 1.1.1, 1.1.1, 1.1.1, 1.2, 1.2, 1.2, 3, 3.1.3, 3.1.4, 3.1.5, 3.1.2, 3.1.7, 3.1.4, 3.1.4, 3.1.4

[13] Jon Louis Bentley. Multidimensional binary search trees used for associative searching. *Communications of the ACM*, 18(9):509–517, 1975. 1.3

[14] Sergei Bernstein. On a modification of chebyshev's inequality and of the error formula of laplace. *Ann. Sci. Inst. Sav. Ukraine, Sect. Math*, 1(4):38–49, 1924. 4.2.3

[15] Stéphane Boucheron, Gábor Lugosi, and Pascal Massart. *Concentration inequalities: A nonasymptotic theory of independence*. Oxford university press, 2013. C.2

[16] Christos Boutsidis and David P. Woodruff. Optimal cur matrix decompositions. STOC '14, 2014. 3.1.4

[17] Jan van den Brand, Binghui Peng, Zhao Song, and Omri Weinstein. Training (overparametrized) neural networks in near-linear time. In *ITCS*, 2021. 1, 21, 4.5.12, 4.6.1, 4.6.2, 4.7.3, 4.7.12, 4.7.4, 4.7.13

[18] Tianle Cai, Ruiqi Gao, Jikai Hou, Siyu Chen, Dong Wang, Di He, Zhihua Zhang, and Liwei Wang. Gram-gauss-newton method: Learning overparameterized neural networks for regression problems. *arXiv preprint arXiv:1905.11675*, 2019. 1, 4.6.1

[19] J.Lawrence Carter and Mark N. Wegman. Universal classes of hash functions. *Journal of Computer and System Sciences*, 18(2):143–154, 1979. 2.3.3

[20] Xue Chen and Eric Price. Active regression via linear-sample sparsification. In *Conference on Learning Theory (COLT)*, pages 663–695. PMLR, 2019. 1.1.1

[21] Xue Chen, Daniel M Kane, Eric Price, and Zhao Song. Fourier-sparse interpolation without a frequency gap. In *2016 IEEE 57th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 741–750. IEEE, 2016. 5

[22] Yeshwanth Cherapanamjeri and Jelani Nelson. On adaptive distance estimation. *Advances in Neural Information Processing Systems*, 33:11178–11190, 2020. 1.1.2, 2.4

[23] Yeshwanth Cherapanamjeri and Jelani Nelson. Uniform approximations for randomized hadamard transforms with applications. In *Proceedings of the 54th Annual ACM SIGACT Symposium on Theory of Computing (STOC)*, 2022. 1.1.2, 2.4, 2.4, 2.4.2

[24] Herman Chernoff. A measure of asymptotic efficiency for tests of a hypothesis based on the sum of observations. *The Annals of Mathematical Statistics*, pages 493–507, 1952. 4.2.1

[25] Kenneth L Clarkson. Las vegas algorithms for linear and integer programming when the dimension is small. *Journal of the ACM (JACM)*, 42(2):488–499, 1995. 1

[26] Michael B Cohen, TS Jayram, and Jelani Nelson. Simple analyses of the sparse johnson-lindenstrauss transform. In *1st Symposium on Simplicity in Algorithms (SOSA 2018)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2018. 2.3.2, 2.3.3, 2.3.3

[27] Michael B Cohen, Yin Tat Lee, and Zhao Song. Solving linear programs in the current matrix multiplication time. In *STOC*, 2019. 1, 1.1.1, 1.1.1, 1.1.1, 1.1.1, 1.1.1, 1.1.2, 1.1.2, 1.1.2, 2.6, 2.6.1, 2.6.13, 2.6.2, 2.6.17, 2.6.2, B.2, B.2.9

[28] George B Dantzig. Maximization of a linear function of variables subject to linear inequalities. *Activity analysis of production and allocation*, 13:339–347, 1947. 1

[29] Anirban Dasgupta, Ravi Kumar, and Tamás Sarlos. A sparse johnson: Lindenstrauss transform. STOC '10, 2010. 2.3.2, 2.3.3

[30] Mayur Datar, Nicole Immorlica, Piotr Indyk, and Vahab S Mirrokni. Locality-sensitive hashing scheme based on p-stable distributions. In *Proceedings of the twentieth annual symposium on Computational geometry (SoCG)*, pages 253–262, 2004. 1.3, 2.1

[31] Simon S Du, Jason D Lee, Haochuan Li, Liwei Wang, and Xiyu Zhai. Gradient descent finds global minima of deep neural networks. In *International Conference on Machine Learning (ICML)*, 2019. 1, 4.6.1

[32] Simon S Du, Xiyu Zhai, Barnabas Poczos, and Aarti Singh. Gradient descent provably optimizes over-parameterized neural networks. In *ICLR*, 2019. 1, 4.6.1

[33] François Le Gall and Florent Urrutia. Improved rectangular matrix multiplication using powers of the coppersmith-winograd tensor. In *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '18, page 1029–1046, 2018. 1, 1.1.18, 2.5.1, 2.5.3

[34] Moritz Hardt and David P. Woodruff. How robust are linear sketches to adaptive inputs? In *Proceedings of the Forty-Fifth Annual ACM Symposium on Theory of Computing*, STOC '13, 2013. 3

[35] Haitham Hassanieh, Piotr Indyk, Dina Katabi, and Eric Price. Nearly optimal sparse fourier transform. In *Proceedings of the forty-fourth annual ACM symposium on Theory of computing (STOC)*, pages 563–578, 2012. 5

[36] David Haussler and Emo Welzl. ε-nets and simplex range queries. *Discrete & Computational Geometry*, 2(2):127–151, 1987. 2.1.4

[37] Wassily Hoeffding. Probability inequalities for sums of bounded random variables. *Journal of the American Statistical Association*, 58(301):13–30, 1963. 4.2.2

[38] Baihe Huang, Xiaoxiao Li, Zhao Song, and Xin Yang. Fl-ntk: A neural tangent kernel-based framework for federated learning analysis. In *International Conference on Machine Learning*, pages 4423–4434. PMLR, 2021. 1

[39] Piotr Indyk. Better algorithms for high-dimensional proximity problems via asymmetric embeddings. In *Proceedings of the fourteenth annual ACM-SIAM symposium on Discrete algorithms (SODA)*, pages 539–545, 2003. 1.1.1, 1.1.2, 2.3.1, A.1

[40] Piotr Indyk and Rajeev Motwani. Approximate nearest neighbors: towards removing the curse of dimensionality. In *Proceedings of the thirtieth annual ACM symposium on Theory of computing (STOC)*, pages 604–613, 1998. 1.3

[41] Arthur Jacot, Franck Gabriel, and Clément Hongler. Neural tangent kernel: convergence and generalization in neural networks. In *Proceedings of the 32nd International Conference on Neural Information Processing Systems (NeurIPS)*, pages 8580–8589, 2018. 1

[42] Shunhua Jiang, Zhao Song, Omri Weinstein, and Hengjie Zhang. Faster dynamic matrix inverse for faster lps. In *Proceedings of the 53rd Annual ACM SIGACT Symposium on*

Theory of Computing (STOC), 2021. 1, 1.4

[43] William B Johnson and Joram Lindenstrauss. Extensions of lipschitz mappings into a hilbert space. *Contemporary mathematics*, 26(189-206):1, 1984. 1.3, 2.3.2

[44] Richard V. Kadison and Isadore M. Singer. Extensions of pure states. *American Journal of Mathematics*, 81(2):383–400, 1959. 1.4

[45] Daniel M Kane and Jelani Nelson. A derandomized sparse johnson-lindenstrauss transform. 2010. 2.3.2, 2.3.3

[46] Daniel M Kane and Jelani Nelson. Sparser johnson-lindenstrauss transforms. *Journal of the ACM (JACM)*, 61(1):1–23, 2014. 2.3.2, 2.3.3

[47] Narendra Karmarkar. A new polynomial-time algorithm for linear programming. In *Proceedings of the sixteenth annual ACM symposium on Theory of computing*, pages 302–311, 1984. 1

[48] Jonathan A. Kelner, Lorenzo Orecchia, Aaron Sidford, and Zeyuan Allen Zhu. A simple, combinatorial algorithm for solving sdd systems in nearly-linear time. STOC '13, 2013. 1.1.2

[49] Leonid G Khachiyan. Polynomial algorithms in linear programming. *USSR Computational Mathematics and Mathematical Physics*, 20(1):53–72, 1980. 1

[50] Victor Klee and George J Minty. How good is the simplex algorithm. *Inequalities*, 3(3): 159–175, 1972. 1

[51] Lap Chi Lau and Hong Zhou. A spectral approach to network design. STOC'20, 2020. 1

[52] Jason D Lee, Ruoqi Shen, Zhao Song, Mengdi Wang, and Zheng Yu. Generalized leverage score sampling for neural networks. In *NeurIPS*, 2020. 1

[53] Yin Tat Lee and He Sun. Constructing linear-sized spectral sparsification in almost-linear time. In *IEEE 56th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 250–269, 2015. 1, 1.1.1, 1.1.1, 1.1.1, 1.2, 3.1, 3.1.4, 3.1.4, 3.1.4, 3.1.5, 3.1.5, 3.1.5, 3.1.10

[54] Yin Tat Lee and He Sun. An sdp-based algorithm for linear-sized spectral sparsification. In *Proceedings of the 49th Annual ACM SIGACT Symposium on Theory (STOC)*, pages 678–687, 2017. 1, 1.1.1, 1.1.1, 1.1.1, 3, 3.1.4, 3.1.4, 3.1.4

[55] Yin Tat Lee, Zhao Song, and Qiuyi Zhang. Solving empirical risk minimization in the current matrix multiplication time. In *COLT*, 2019. 1, 1.1.1, 1.1.1, 1.1.1, 1.1.1, 1.1.2, 1.1.2, 1.1.2, 2.6, 2.6.13, 2.6.2, B.2, B.2.9

[56] Yuanzhi Li and Yingyu Liang. Learning overparameterized neural networks via stochastic gradient descent on structured data. In *NeurIPS*, 2018. 1

[57] Yichao Lu, Paramveer Dhillon, Dean P Foster, and Lyle Ungar. Faster ridge regression via the subsampled randomized hadamard transform. In *Advances in neural information processing systems (NIPS)*, pages 369–377, 2013. 4.5.11

[58] Adam W Marcus, Daniel A Spielman, and Nikhil Srivastava. Interlacing families ii: Mixed characteristic polynomials and the Kadison—Singer problem. *Annals of Mathematics*, pages 327–350, 2015. 1.4

[59] Behnam Neyshabur and Nathan Srebro. On symmetric and asymmetric lshs for inner product search. In *International Conference on Machine Learning*, pages 1926–1934. PMLR, 2015. 1.1.1, 2.3, 2.3.1

[60] Lorenzo Orecchia. *Fast Approximation Algorithms for Graph Partitioning using Spectral and Semidefinite-Programming Techniques*. PhD thesis, EECS Department, University of California, Berkeley, 2011. 3.1.4

[61] Rasmus Pagh. Compressed matrix multiplication. *ACM Transactions on Computation Theory (TOCT)*, 5(3):1–17, 2013. 2.3.3

[62] Eric Price. Efficient sketches for the set query problem. In *Proceedings of the twenty-second annual ACM-SIAM symposium on Discrete Algorithms (SODA)*, pages 41–56. SIAM, 2011. 5

[63] Eric Price and Zhao Song. A robust sparse Fourier transform in the continuous setting. In *2015 IEEE 56th Annual Symposium on Foundations of Computer Science*, pages 583–600. IEEE, 2015. 5, 2.2.8

[64] Mihai Pundefinedtraşcu and Mikkel Thorup. The power of simple tabulation hashing. *J. ACM*, 2012. 2.3.3

[65] James Renegar. A polynomial-time algorithm, based on newton's method, for linear programming. *Mathematical programming*, 40(1):59–93, 1988. 1

[66] Mark Rudelson and Roman Vershynin. Smallest singular value of a random rectangular matrix. *Communications on Pure and Applied Mathematics: A Journal Issued by the Courant Institute of Mathematical Sciences*, 62(12):1707–1739, 2009. 4.2.6

[67] Tamas Sarlos. Improved approximation algorithms for large matrices via random projections. In *2006 47th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 143–152. IEEE, 2006. 4.5.10, C.2

[68] J. Schur. Bemerkungen zur theorie der beschränkten bilinearformen mit unendlich vielen veränderlichen. *Journal für die reine und angewandte Mathematik*, 140, 1911. 4.2.7

[69] Anshumali Shrivastava, Zhao Song, and Zhaozhuo Xu. Breaking the linear iteration cost barrier for some well-known conditional gradient methods using maxip data-structures. NeurIPS'21, 2021. 2

[70] Anshumali Shrivastava, Zhao Song, and Zhaozhuo Xu. Sublinear least-squares value iteration via locality sensitive hashing, 2021. 2

[71] Zhao Song and Xin Yang. Quadratic suffices for over-parametrization via matrix chernoff bound. *arXiv preprint arXiv:1906.03593*, 2019. 1, 4.6.1

[72] Zhao Song and Zheng Yu. Oblivious sketching-based central path method for solving linear programming problems. In *38th International Conference on Machine Learning (ICML)*, 2021. 1, 1.1.1, 1.1.1, 1.1.1, 1.1.1, 1.1.1, 1.1.2, 1.1.2, 1.1.2, 2.6, 2.6.1, 2.6.13, 2.6.2, 2.6.2, 2.6.2, B.2, B.2.9

[73] Zhao Song, David P. Woodruff, Zheng Yu, and Lichen Zhang. Fast sketching of polynomial kernels of polynomial degree. In *ICML*, 2021. 4.5.3, 4.5.15

[74] Zhao Song, Zhaozhuo Xu, and Lichen Zhang. Faster algorithm for one-sided kadison-singer via furthest-neighbor search. In *manuscript*. https://lczh.github.io/sxz21.pdf, 2021. 1.4, 3

[75] Zhao Song, Shuo Yang, and Ruizhe Zhang. Does preprocessing help training over-parameterized neural networks? In *Thirty-Fifth Conference on Neural Information Processing Systems (NeurIPS)*, 2021. 1, 1.1.1, 1.1.1, 1.1.1, 1.2, 1.2, 2.2.3, 4.6.2, 4.6.7, 4.6.9

[76] Zhao Song, Lichen Zhang, and Ruizhe Zhang. Training multi-layer over-parametrized neural network in subquadratic time. 2021. URL `https://arxiv.org/pdf/2112.07628.pdf`. 1.1.1, 1.1.1

[77] Zhao Song, Baocheng Sun, Omri Weinstein, and Ruizhe Zhang. Sparse fourier transform over lattices: A unified approach to signal reconstruction. http://arxiv.org/abs/2205.00658, 2022. 1

[78] Zhao Song, Zhaozhuo Xu, and Lichen Zhang. Speeding up sparsification with inner product search data structures. 2022. URL `https://arxiv.org/pdf/2204.03209.pdf`. 1.1.1, 1, 3

[79] Zhao Song, Yuanyuan Yang, and Lichen Zhang. Fast dynamic kronecker projection maintenance via private sketching. 2022. 4

[80] Daniel A Spielman and Nikhil Srivastava. Graph sparsification by effective resistances. *SIAM Journal on Computing*, 40(6):1913–1926, 2011. 1, 3.1.4, 3.1.4

[81] Nikhil Srivastava. *Spectral Sparsification and Restricted Invertibility*. PhD thesis, USA, 2010. 1, 1.2, 3.2.1

[82] Pravin M. Vaidya. An algorithm for linear programming which requires o(((m+n)nˆ2 + (m+n)ˆ1.5 n)l) arithmetic operations. In Alfred V. Aho, editor, *Proceedings of the 19th Annual ACM Symposium on Theory of Computing, 1987, New York, New York, USA*, pages 29–38. ACM, 1987. 1

[83] Pravin M Vaidya. A new algorithm for minimizing convex functions over convex sets. In *30th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 338–343, 1989. 1

[84] Nik Weaver. The Kadison–Singer problem in discrepancy theory, ii. *https://arxiv.org/pdf/1303.2405.pdf*, 2013. 1, 1.1.1, 1.1.1, 1.2, 1.2, 1.4, 3, 3.2, 3.2.1, 3.2.1, 3.2.3, 3.2.7, 16

[85] Virginia Vassilevska Williams. Multiplying matrices faster than coppersmith-winograd. In *Proceedings of the forty-fourth annual ACM symposium on Theory of computing (STOC)*, pages 887–898. ACM, 2012. 2.5.1

[86] Max A Woodbury. The stability of out-input matrices. *Chicago, IL*, 9, 1949. 3.2.3

[87] Max A Woodbury. Inverting modified matrices. 1950. 3.2.3

[88] David P. Woodruff. Sketching as a tool for numerical linear algebra. *Foundations and Trends in Theoretical Computer Science*, 10(1–2):1–157, 2014. 4.5.11

[89] Guodong Zhang, James Martens, and Roger B Grosse. Fast convergence of natural gra-

dient descent for over-parameterized neural networks. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2019. 1

[90] Anastasios Zouzias. A matrix hyperbolic cosine algorithm and applications. In *International Colloquium on Automata, Languages, and Programming*, pages 846–858. Springer, 2012. 1, 1.2, 1.2, 3, 3.1.4, 3.1.4, 3.1.4