A Strongly Polynomial Algorithm for Approximate Forster Transforms and its Application to Halfspace Learning*

Ilias Diakonikolas
University of Wisconsin-Madison
Madison, USA
ilias@cs.wisc.edu

Christos Tzamos
University of Wisconsin-Madison
& University of Athens
Madison, USA
tzamos@wisc.edu

Daniel M. Kane
University of California, San Diego
San Diego, USA
dakane@ucsd.edu

ABSTRACT

The Forster transform is a method of regularizing a dataset by placing it in *radial isotropic position* while maintaining some of its essential properties. Forster transforms have played a key role in a diverse range of settings spanning computer science and functional analysis. Prior work had given *weakly* polynomial time algorithms for computing Forster transforms, when they exist. Our main result is the first *strongly polynomial time* algorithm to compute an approximate Forster transform of a given dataset or certify that no such transformation exists. By leveraging our strongly polynomial Forster algorithm, we obtain the first strongly polynomial time algorithm for *distribution-free* PAC learning of halfspaces. This learning result is surprising because *proper* PAC learning of halfspaces is *equivalent* to linear programming. Our learning approach extends to give a strongly polynomial halfspace learner in the presence of random classification noise and, more generally, Massart noise.

CCS CONCEPTS

• **Theory of computation → Machine learning theory**.

KEYWORDS

PAC learning, Halfspaces, Massart Noise

ACM Reference Format:
Ilias Diakonikolas, Christos Tzamos, and Daniel M. Kane. 2023. A Strongly Polynomial Algorithm for Approximate Forster Transforms and its Application to Halfspace Learning. In *Proceedings of the 55th Annual ACM Symposium on Theory of Computing (STOC '23), June 20–23, 2023, Orlando, FL, USA.* ACM, New York, NY, USA, 14 pages. https://doi.org/10.1145/3564246.3585191

1 INTRODUCTION

The Forster transform is a method of regularizing a dataset X (in particular, by placing it in *radial* isotropic position) while maintaining some of its essential properties. Forster transforms have been an essential tool in a diverse range of settings, including functional analysis [4, 25], communication complexity [22], coding theory [21], mixed determinant/volume approximation [30], learning theory [17, 18, 32, 33] and the Paulsen problem in frame theory [31, 35]. The reader is referred to [3] for a more detailed discussion.

Known algorithms for computing (approximate) Forster transforms [3, 17, 32] rely on black-box convex optimization (e.g., the ellipsoid algorithm) and consequently have *weakly* polynomial runtimes. Here we study the question of whether Forster transforms can be computed in *strongly* polynomial time. We then leverage Forster transforms for the problem of PAC learning halfspaces (both in the realizable setting and in the presence of semi-random label noise).

Intuitively speaking, a Forster transform is a mapping that turns a dataset into one with good *anti-concentration* properties. Specifically, given a dataset $X \subset \mathbb{R}_*^d$[1], a Forster transform of X is an invertible linear transformation $A \in \mathbb{R}^{d\times d}$ such that the set of points $Y = \{Ax/\|Ax\|_2, x \in X\}$ is in isotropic position (i.e., has identity second moment matrix). Formally, we have the following more general definition allowing for *approximate* isotropic position.

Definition 1.1 (Approximate Forster Transform)**.** Let X be a set of n nonzero points in $\mathbb{R}^d$ and $0 \le \epsilon \le 1$ be an error parameter. An *ϵ-approximate Forster transform* of X is an invertible linear transformation $A \in \mathbb{R}^{d\times d}$ such that, considering the mapping $f_A : \mathbb{R}_*^d \mapsto \mathbb{S}^d$ defined by $f_A(x) \stackrel{\text{def}}{=} Ax/\|Ax\|_2$, the matrix $M_A(X) \stackrel{\text{def}}{=} (1/n)\sum_{x\in X} f_A(x)f_A(x)^\top$ satisfies $\frac{1-\epsilon}{d} I \le M_A(X) \le \frac{1+\epsilon}{d} I$.

An *exact* Forster transform (corresponding to $\epsilon = 0$ in Definition 1.1) aims to linearly transform a given dataset so that the normalizations of these points are in isotropic position. This notion is known as "Forster's isotropic position" or "radial isotropic position" and can be viewed as an outlier-robust analogue of isotropic position. As already mentioned, radial isotropy has been extensively studied in functional analysis and computer science.

Remark 1.2. At a high-level, a Forster transform aims to transform a given dataset so that it becomes "well-conditioned" in a well-defined technical sense. We note that several other such transformations have been studied in the literature, including the "outlier-removal technique" of Dunagan and Vempala [19] (improving on [7]) and the rescaling method of Dunagan and Vempala [20] for linear programming. We provide

*Author names are in randomized order.

[1] We use $\mathbb{R}_*$ to denote the set $\mathbb{R} \setminus \{\mathbf{0}\}$.

STOC '23, June 20–23, 2023, Orlando, FL, USA

ACM ISBN 978-1-4503-9913-5/23/06…$15.00
https://doi.org/10.1145/3564246.3585191

a summary of these techniques and a comparison to radial isotropy in Section 1.4.

Existence. Forster [22] showed that if the set of points X is in general position, then a Forster transform exists. Interestingly, generalizations of Forster's theorem appear implicitly in [4] and explicitly in [30]. We note that there are datasets for which a Forster transform does not exist. For example, if there is a $d/3$-dimensional subspace that contains half of the points in X, then after applying *any* such transformation to our dataset, this will still be the case; thus, there will be a $d/3$-dimensional subspace over which the trace of the second moment matrix is at least $1/2$. In a recent refinement of the aforementioned works, [33] showed that this is the only thing that can go wrong. That is, a Forster transform of a given dataset X exists unless there is a k-dimensional subspace, for some $0 < k < d$, containing at least a k/d-fraction of the points in X.

Efficient Computability. Forster's existence proof proceeds via a non-constructive iterative argument. By analyzing a convex program proposed by Barthe [4], Hardt and Moitra [32] (see also [3]) showed that the ellipsoid method yields a *weakly polynomial* time algorithm to compute an approximate Forster transform (when it exists). (More recently, [17] pointed out that a simple explicit SDP can be used to obtain a similar guarantee.) We remind the reader that the term *weakly polynomial time* algorithm refers to the fact that the *number of arithmetic operations* performed by the algorithm scales polynomially with *the bit complexity of the numbers in the input*. Specifically, in our Forster setting, the number of arithmetic operations required by the ellipsoid method is $\mathrm{poly}(n, d, b, \log(1/\epsilon))$, where ϵ is the accuracy parameter of Definition 1.1, n is the size of the dataset X, and b is the bit complexity of X.

Starting from the convex programming formulation in [4], Artstein-Avidan, Kaplan, and Sharir [3] gave an SVD-based gradient-descent method for computing approximate Forster transforms. This method incurs a $\mathrm{poly}(1/\epsilon)$ runtime dependence and is still weakly polynomial, i.e., the number of arithmetic operations scales polynomially in the bit complexity b. Finally, it is interesting to remark that Forster's rescaling is a special case of operator scaling and tensor scaling (see [24] for a survey). Efficient algorithms have been developed for these more general tasks, see, e.g., [1, 9], albeit with weakly polynomial guarantees.

Weakly versus Strongly Polynomial Time. As is standard for computational purposes, we assume that every integer or rational number appearing in the input is encoded using its binary representation. Let $N \in \mathbb{Z}_+$ denote the number of integer numbers given as input and $b \in \mathbb{Z}_+$ denote the bit complexity of the largest integer appearing in the input description. An algorithm for the underlying computational problem is called *weakly polynomial*, if its worst-case running time is bounded by a fixed-degree polynomial in the Turing machine model of computation.

The concept of *strongly polynomial* time was introduced by Megiddo [39], under the name "genuinely polynomial". A strongly polynomial time algorithm satisfies the following properties (see, e.g., Section 1.3 of [29]): (i) it uses only elementary arithmetic operations (specifically, integer addition, subtraction, multiplication, and division), (ii) the number of arithmetic operations is bounded above by a polynomial in N, and (iii) the algorithm is a polynomial space algorithm: that is, all numbers appearing in all intermediate computations are rational numbers with bit complexity bounded above by a polynomial in the input size (i.e., $\mathrm{poly}(N, b)$).

The key difference between strongly and weakly polynomial time lies in property (ii) above. In a weakly polynomial algorithm, the *number of arithmetic operations* is allowed to scale with the bit complexity of the numbers in the input. *In sharp contrast, in a strongly polynomial time algorithm no bit complexity dependence is allowed.*

Forster Transforms in Strongly *Polynomial Time?* Motivated by the fundamental nature and the varied applications of Forster transforms, here we ask the following question:

Is there a strongly *polynomial time algorithm to compute an approximate Forster transform of a given dataset (assuming one exists)?*

Our main algorithmic result (Theorem 1.5) answers this question in the affirmative by giving the first randomized strongly polynomial-time algorithm for computing approximate Forster transforms — corresponding to $\epsilon = \Omega(\mathrm{poly}(1/(n, d)))$ in Definition 1.1. Importantly, a constant value of ϵ suffices for our learning theory application to learning halfspaces. Obtaining a strongly polynomial time algorithm for inverse exponential values of ϵ is left as an interesting open problem (see Section 7 for a discussion).

1.1 Halfspaces and Efficient PAC Learnability

One of the main motivations behind this work was leveraging Forster transforms as a tool for the algorithmic problem of distribution-free PAC learning of halfspaces. We review the relevant background in the subsequent discussion.

Halfspaces. We are concerned with the efficient learnability of halfspaces in Valiant's distribution-free PAC model [53]. A *halfspace* or Linear Threshold Function (LTF) is any Boolean-valued function $f : \mathbb{R}^d \mapsto \{\pm 1\}$ of the form $f(x) = \mathrm{sign}(w \cdot x - t)$, for some $w \in \mathbb{R}^d$ (known as the weight vector) and $t \in \mathbb{R}$ (known as the threshold). (The function $\mathrm{sign} : \mathbb{R} \mapsto \{\pm 1\}$ is defined as $\mathrm{sign}(u) = 1$ if $u \geq 0$, and $\mathrm{sign}(u) = -1$ otherwise.) Halfspaces are one of the most extensively studied classes of Boolean functions due to their central role in several areas, including complexity theory, learning theory, and optimization [23, 28, 40, 42, 43, 48, 49, 54, 58].

Background on PAC Learning. The major goal of computational learning theory is to develop learning algorithms for expressive concept classes that are both statistically and computationally efficient. To facilitate the subsequent discussion, we formally define Valiant's PAC model.

Definition 1.3 (PAC Learning)**.** Let C be a class of Boolean-valued functions over $X = \mathbb{R}^d$ and $\mathcal{D}_X$ be a fixed but unknown distribution over X. Let f be an unknown target function in C. A *PAC example oracle*, $\mathrm{EX}(f, \mathcal{D}_X)$, works as follows:

Each time $\mathrm{EX}(f, \mathcal{D}_X)$ is invoked, it returns a labeled example (x, y), where $x \sim \mathcal{D}_X$ and $y = f(x)$. Let $\mathcal{D}$ denote the joint distribution on (x, y) generated by the above oracle. Given an accuracy parameter $\gamma > 0$ and access to i.i.d. samples from $\mathcal{D}$, the learner wants to output a hypothesis $h : \mathbb{R}^d \mapsto \{\pm 1\}$ such that with high probability the misclassification error of h is at most γ, i.e., we have that $\mathbf{Pr}_{(x,y)\sim\mathcal{D}}[h(x) \neq y] \leq \gamma$.

The hypothesis h in Definition 1.3 does not necessarily belong to the class C. Namely, we focus on the standard notion of *improper* learning, where the learner can output any efficiently computable hypothesis. The special case where h is required to lie in C is known as *proper* learning. While proper learning might be desirable for some applications (e.g., due to its interpretability), there exist natural concept classes for which proper learning is computationally hard and improper learning is easy (see, e.g., [34]). An improper hypothesis is as useful as a proper one for the purpose of predicting new function values.

Remark 1.4. The PAC model of Definition 1.3 is known as *realizable* because of the assumption that the labels are consistent with the target concept. While our main learning application is on the realizable learning of halfspaces in strongly polynomial time (Theorem 1.6), our positive result extends for learning halfspaces in the presence of random or semi-random label noise (Theorem 1.8).

PAC Learning Halfspaces and Linear Programming. With this terminology, we return to our discussion on halfspaces. Suppose we are given a multiset of n labeled examples, $(x^{(i)}, y^{(i)})$, with $x^{(i)} \sim \mathcal{D}_X$ and $y^{(i)} = f^*(x^{(i)})$, where $f^*(x) = \mathrm{sign}(w^* \cdot x - t^*)$ is the target halfspace. Then we can find a consistent halfspace hypothesis $h(x) = \mathrm{sign}(\widehat{w} \cdot x - \widehat{t})$ (i.e., a halfspace that agrees with the training set) via a reduction to Linear Programming (LP); see, e.g., [37]. Indeed, each example $(x^{(i)}, y^{(i)})$ gives rise to the linear inequality $(w \cdot x^{(i)} - t)y^{(i)} \geq 0$ over variables $(w, t) \in \mathbb{R}^{d+1}$. This gives us an LP with $d+1$ variables and n constraints, which is feasible (as (w^*, t^*) is a feasible solution by assumption). We can thus use any polynomial-time LP algorithm to compute a feasible solution $(\widehat{w}, \widehat{t})$. By standard VC-dimension generalization results (see, e.g., [34]), if the sample size n is sufficiently large, namely for some $n = \tilde{O}(d/\gamma)$, the halfspace hypothesis $h(x) = \mathrm{sign}(\widehat{w} \cdot x - \widehat{t})$ with high probability satisfies $\mathbf{Pr}_{(x,y)\sim\mathcal{D}}[h(x) \neq y] \leq \gamma$. This straightforward reduction gives a PAC learning algorithm for halfspaces on $\mathbb{R}^d$ with sample complexity $\tilde{O}(d/\gamma)$ and running time polynomial in the input size. Formally speaking, the running time of such an algorithm is *weakly polynomial*, i.e., its worst-case number of arithmetic operations scales with the bit complexity of the input examples.

Interestingly, the aforementioned reduction can be reversed. That is, one can use any PAC learner that outputs a halfspace hypothesis as a black-box to solve the linear feasibility problem $Aw \geq 0$, $w \neq 0$, where $A \in \mathbb{R}^{n\times d}$ and $w \in \mathbb{R}^d$, by considering each linear constraint as an example. Intuitively, the vector w can be viewed as the weight vector defining the target halfspace.

Learning Halfspaces in Strongly *Polynomial Time?* All known polynomial time algorithms for LP, including the ellipsoid algorithm and interior-point methods, are weakly polynomial. The existence of a strongly polynomial LP algorithm is a major open question in computer science, famously highlighted by Smale [50]. The straightforward reduction of PAC learning halfspaces to LP leads to a *weakly* polynomial learner. Interestingly, the reduction in the opposite direction has lead various authors (see [11] and recently [10, 14]) to suggest that learning halfspaces in strongly polynomial time is *equivalent* to strongly polynomial LP. The catch, of course, is that this equivalence only holds if we restrict ourselves to *proper* learners.

Several weakly polynomial time algorithms for PAC learning halfspaces have been developed over the past thirty years, starting with the pioneering works [7, 11, 20] and recently in [10, 14, 17]. (These works do not proceed by a black-box reduction to solving LPs.) These learners succeed not only in the realizable setting, but also in the presence of (semi)-random label noise. Importantly, all prior learners are weakly polynomial — even restricted to the realizable setting. This discussion serves as a motivation for the following question:

Is there a strongly *polynomial time algorithm for PAC learning halfspaces?*

The main learning-theoretic result of this paper (Theorem 1.6) answers the above question in the affirmative. This algorithmic result generalizes to yield strongly polynomial time algorithms for learning halfspaces in "benign" noise models, including Random Classification Noise (RCN) [2] and, more generally, Massart noise [38] (Theorem 1.8).

1.2 Our Results

The main algorithmic result of this work is the first randomized strongly polynomial time algorithm for computing an approximate Forster transform of a given dataset, assuming that one exists.

Theorem 1.5 (Approximate Forster Transforms in Strongly Polynomial Time). *There exists a randomized algorithm that given a set $X \subset \mathbb{R}^d_*$ of size n and a parameter $\epsilon \in (0, 1)$, runs in time strongly polynomial in nd/ϵ, and has the following high probability guarantee: either the algorithm computes an ϵ-approximate Forster transform of X, or it correctly detects that no Forster transform of X exists by finding a proper subspace $W \subset \mathbb{R}^d$ such that $|X \cap W| > (n/d)\ \dim(W)$.*

In more detail, the algorithm of Theorem 1.5 performs $\mathrm{poly}(n, d, 1/\epsilon)$ arithmetic operations on $\mathrm{poly}(n, d, 1/\epsilon, b)$-bit numbers, where b is the bit complexity of the points in X. As discussed in the introduction, previous algorithms for this problem rely on the ellipsoid method and therefore are weakly polynomial even for constant values of ϵ. The running time of our algorithm has a polynomial dependence in $1/\epsilon$; hence, our algorithm does not run in polynomial time when ϵ is inverse super-polynomially small in n, d. Importantly, for our application in halfspace learning (and several other applications of Forster transforms) constant values of the parameter ϵ suffice.

By using the algorithm of Theorem 1.5 as a black-box (for $\epsilon = 1/2$), we establish our main learning result (see Theorem 6.5 for a more detailed statement).

Theorem 1.6 (PAC Learning Halfspaces in Strongly Polynomial Time). *Let $\mathcal{D}$ be a distribution over labeled examples $(x, y) \in \mathbb{R}^d \times \{\pm 1\}$ such that the distribution over examples is arbitrary and the label y of example x satisfies $y = f(x)$, for an unknown halfspace $f : \mathbb{R}^d \mapsto \{\pm 1\}$. There is an algorithm that, given $\gamma > 0$, draws $n = \mathrm{poly}(d/\gamma)$ i.i.d. samples from $\mathcal{D}$, runs in strongly polynomial time, and returns a strongly polynomial time computable hypothesis $h : \mathbb{R}^d \mapsto \{\pm 1\}$ such that with high probability we have that $\mathbf{Pr}_{(x,y)\sim\mathcal{D}}[h(x) \neq y] \leq \gamma$.*

Given the *equivalence* of *proper* halfspace learning and LP, we view this algorithmic result as fairly surprising. Theorem 1.6 gives the first strongly polynomial time PAC learning algorithm for halfspaces. In more detail, if b is the bit complexity of the examples (i.e., the maximum number of bits required to represent each coordinate of each example vector), our algorithm uses $\mathrm{poly}(n)$ arithmetic operations on $\mathrm{poly}(n, b)$-bit numbers. Finally, we note that the hypothesis h computed by our algorithm is a decision-list of $\mathrm{poly}(d/\gamma)$ many halfspaces. Importantly, for each point x, the value $h(x)$ is computable in strongly polynomial time (in n).

Remark 1.7. The list of concept classes for which efficient learners have been developed in Valiant's distribution-free PAC model is fairly short. The class of halfspaces is of central importance in this list. Specifically, a strongly polynomial algorithm for PAC learning halfspaces immediately implies (via the kernel trick) strongly polynomial learners for broader concept classes, including degree-k polynomial threshold functions for any $k = O(1)$ (see, e.g., [8]).

It is worth pointing out that the idea of using Forster transforms for halfspace learning was recently used in [17] for the problem of PAC learning with Massart noise. In the Massart model [38], an adversary independently flips the label of each point x with unknown probability $\eta(x) \leq \eta < 1/2$. The learner of [17] used a weakly polynomial Forster transform routine. By instead using our algorithm of Theorem 1.5, we obtain the following generalization of Theorem 1.6.

Theorem 1.8 (PAC Learning Massart Halfspaces in Strongly Polynomial Time). *Let $\mathcal{D}$ be a distribution over labeled examples $(x, y) \in \mathbb{R}^d \times \{\pm 1\}$ such that the distribution over examples is arbitrary and the label y of example x satisfies (i) $y = f(x)$ with probability $1 - \eta(x)$, and (ii) $y = -f(x)$ with probability $\eta(x)$, for an unknown halfspace $f : \mathbb{R}^d \mapsto \{\pm 1\}$. Here $\eta(x)$ is an unknown function that satisfies $\eta(x) \leq \eta < 1/2$ for all x. There is an algorithm that, given $\gamma > 0$, draws $n = \mathrm{poly}(d/\gamma)$ i.i.d. samples from $\mathcal{D}$, runs in strongly polynomial time, and returns a strongly polynomial time computable hypothesis $h : \mathbb{R}^d \mapsto \{\pm 1\}$ such that with high probability we have that $\mathbf{Pr}_{(x,y)\sim\mathcal{D}}[h(x) \neq y] \leq \eta + \gamma$.*

Theorem 1.8 generalizes Theorem 1.6 (which corresponds to the case of $\eta = 0$). For the special case of uniform noise (i.e., when $\eta(x) = \eta < 1/2$ for all x) — this is known as Random Classification Noise [2] — Theorem 1.8 achieves the information-theoretically optimal error and runs in *strongly* polynomial time. It thus qualitatively improves on the classical work of [7] who gave a weakly polynomial time algorithm with the same error guarantee.

Theorem 1.8 similarly improves prior work on learning halfspaces with Massart noise. Prior algorithms for learning Massart halfspaces have weakly polynomial runtimes and achieve the same error as Theorem 1.8, which is believed to be the computational limit for the problem. In more detail, the first (weakly) polynomial learner for Massart halfspaces was given in [14] and achieves error $\eta + \gamma$, as our Theorem 1.8. While this error guarantee is not information-theoretically optimal in the Massart model (the optimal error is $\mathrm{OPT} = \mathbf{E}_x[\eta(x)]$), there exists strong evidence [15, 16, 41] that the bound of η cannot be improved by any polynomial time algorithm. Finally, we note that subsequent work to [14] gave a proper learner for Massart halfspaces [10], which is inherently weakly polynomial.

1.3 Our Techniques

1.3.1 Strongly Polynomial Approximate Forster Transform.

Overview of Approach. Letting $f_A(x) \stackrel{\mathrm{def}}{=} Ax/\|Ax\|_2$, given a dataset X of n points in $\mathbb{R}^d_*$, our goal is to efficiently compute an invertible linear transformation $A \in \mathbb{R}^{d\times d}$ such that the matrix $M_A(X) \stackrel{\mathrm{def}}{=} (1/n)\sum_{x\in X} f_A(x) f_A(x)^\top$ is approximately equal to $(1/d)\, I$; in particular, we would like it to have eigenvalues in $[\frac{1-\epsilon}{d}, \frac{1+\epsilon}{d}]$. Since the trace of $M_A(X)$, $\mathrm{tr}(M_A(X))$, is always equal to 1, this goal is equivalent to finding a matrix A such that the squared Frobenius norm of $M_A(X)$, $\|M_A(X)\|_F^2$, is close to $1/d$ (Lemma 3.1). This observation gives rise to the natural idea of using an iterative algorithm to compute such an A. In particular, given a linear transformation A such that $\|M_A(X)\|_F^2$ is somewhat small, our goal is then to find another linear transformation $C \in \mathbb{R}^{d\times d}$ such that the corresponding second moment matrix $M_{CA}(X) = (1/n)\sum_{x\in X} f_{CA}(x) f_{CA}(x)^\top$ has squared Frobenius norm, $\|M_{CA}(X)\|_F^2$, somewhat smaller than $\|M_A(X)\|_F^2$. Equivalently, since for any point $x \in \mathbb{R}^d_*$ it holds that $f_{CA}(x) = f_C(f_A(x))$, we consider the set of transformed points $X_A = f_A(X) \stackrel{\mathrm{def}}{=} \{f_A(x) : x \in X\}$ and aim to make the second moment matrix of $f_C(X_A)$ smaller than the second moment matrix of X_A. If for any invertible A we can find such a C, then by iteratively replacing A by CA we can achieve smaller and smaller values of $\|M_A(X)\|_F^2$, until in the limit it approaches $1/d$.

Since $\mathrm{tr}(M_A(X)) = 1$, if $\|M_A(X)\|_F^2$ is bounded away from $1/d$, some of the eigenvalues of $M_A(X)$ (which average to $1/d$) must differ substantially from $1/d$. This in turn implies that $M_A(X)$ must have a reasonably-sized *eigenvalue gap*. In particular, this means that there exist subspaces V and $V^\perp$, that are each spanned by eigenvectors of $M_A(X)$, such that the eigenvalues of $V^\perp$ exceed the eigenvalues on V by at least some reasonably large $\delta > 0$. Roughly speaking, if we can find a matrix C that decreases the squared Frobenius norm of $M_A(X)$ on $V^\perp \times V^\perp$ and increases the squared Frobenius norm on $V \times V$, this will improve the desired squared Frobenius norm.

A natural approach to achieve this goal is to let C be equal to $I_{V^\perp} + (1+\alpha) I_V$, the identity on $V^\perp$, and $(1+\alpha)$ times the identity on V, for some suitable $\alpha > 0$. It is not hard to see that this choice of C strictly decreases the second moment matrix on $V^\perp$, and strictly increases it on V. Unfortunately,

it might also create cross-terms that will increase the Frobenius norm. To understand the effect of the cross-terms, it is important to consider how close vectors in X_A are to being in V or in $V^\perp$. In particular, let β be the maximum distance that any vector in X_A is from being in either V or $V^\perp$. If $\alpha = O(1)$, this moves approximately $\alpha\beta^2$ of the trace of $M_A(X)$ from $V^\perp$ to V, which improves (i.e., decreases) the squared Frobenius norm by roughly $\alpha\beta^2$ (times some inverse $\mathrm{poly}(dn/\epsilon)$ factors). On the other hand, this also creates cross-terms in the order of $\alpha\beta$, which increases the squared Frobenius norm by a quantity on the order of $\alpha^2\beta^2$. Thus, as long as α is less than β times a sufficiently small polynomial in dn/ϵ, we obtain an improvement in the squared Frobenius norm on the order of $\alpha\beta^2/\mathrm{poly}(dn/\epsilon)$.

This improvement suffices for our purposes, unless β happens to be very small. The latter occurs if all of the points in X_A are either very close to V or very close to $V^\perp$. In such a case, the simple choice of matrix C described in the previous paragraph may not be sufficient, as it will produce too many cross-terms. In order to make progress here, we require a different approach, which we describe next. To describe our approach for this case, we introduce additional terminology. We let X_A^{B} be the set of points in X_A that are close to $V^\perp$. Moreover, let U be the span of the $|V|$ smallest eigenvectors of the matrix $\sum_{x\in X_A^{\mathrm{B}}} xx^\top$, and let $U^\perp$ be the orthogonal subspace. We now define the new matrix C to be $I_{U^\perp} + (1+\alpha)I_U$, the identity on $U^\perp$ and some very large multiple $(1+\alpha)$ of the identity on U. We claim that this choice actually does not create much in the way of cross-terms. In particular, the matrix $\sum_{x\in X_A^{\mathrm{B}}} (Cx)(Cx)^\top = C^\top \sum_{x\in X_A^{\mathrm{B}}} xx^\top C$ will have no $U\times U^\perp$ term, since $\sum_{x\in X_A^{\mathrm{B}}} xx^\top$ does not — as U is an eigenspace of $\sum_{x\in X_A^{\mathrm{B}}} xx^\top$. The second moment matrix $\sum_{x\in X_A^{\mathrm{B}}} f_C(x) f_C(x)^\top$ will have some contribution to $U\times U^\perp$ cross-terms coming from the renormalization; but these will only be on the order of $(\alpha\,\beta)^4$. On the other hand, the matrix $\sum_{x\in X_A\setminus X_A^{\mathrm{B}}} f_C(x) f_C(x)^\top$ will have small $U\times U^\perp$ terms, because each $f_C(x)$ will nearly lie in $U^\perp$. If β is sufficiently small, this leads to roughly $(\alpha\,\beta)^2$ mass being moved from $U^\perp\times U^\perp$ to $U\times U$, while only creating off-diagonal terms on the order of $(\alpha\,\beta)^4$. Thus, this alternate choice of C can be used to decrease the squared Frobenius norm by $\mathrm{poly}(\alpha/(dn))$.

The preceding outline provides a procedure that produces a sequence of matrices $A_1, A_2, \ldots$ such that if $e_i = \|M_{A_i}(X)\|_F^2 - 1/d$, then $e_{i+1} < e_i - \mathrm{poly}(e_i/(dn))$. Therefore, after polynomially many iterations, we have that $\|M_{A_m}(X)\|_F^2 < 1/d + (\epsilon/d)^2$, which implies we have obtained an ϵ-approximate Forster transform. This gives us an efficient algorithm for computing an approximate Forster transform in the real RAM model, assuming the availability of an algorithm for exact eigendecomposition computation.

Additional Technical Obstacles. The above iterative procedure forms the basis of our final strongly polynomial time algorithm. Unfortunately, as is, this procedure does not directly imply a strongly polynomial time algorithm for two reasons: First, we need to control the bit complexities of the matrices A_i (which might become exponentially large). Second, we need to show that our algorithm works with approximate eigendecompositions (which can further be implemented in strongly polynomial time). We elaborate on these issues in the following discussion.

Controlling the Bit Complexity via Rounding. Recall that, in a strongly polynomial time algorithm, all intermediate numbers computed throughout the algorithm must fit in polynomial space. To handle the bit complexity in our setting, we establish the following statement. If the points in the initial dataset $X \subset \mathbb{R}_*^d$ of size n have bit complexity at most b, then the following holds: given a matrix $A \in \mathbb{R}^{d\times d}$ and any $\delta > 0$, we can approximate A by another matrix A' of bit complexity $\mathrm{poly}(b, d, n, \log(1/\delta))$ such that $\|M_{A'}(X)\|_F^2 < \|M_A(X)\|_F^2 + \delta$ (see Theorem 5.1). This structural result suffices for our purposes for the following reason: Replacing each intermediate matrix A_i (in our iterative procedure) by the corresponding A_i' obtained by rounding (for an appropriately small δ) at each step of our algorithm suffices to keep the bit-complexity under control.

To prove the desired structural result, we proceed as follows: First, if A has condition number at most $\exp(\mathrm{poly}(n, b, d))$, it suffices to merely approximate each entry of A to some $\mathrm{poly}(bdn/\log(1/\delta))$ bits of precision. The difficulty arises if the condition number of A is quite large — in fact, exponentially large in our other parameters. If the condition number of A is large, it is because there are large multiplicative gaps in the singular values of A. In such a case, there will be subspaces V and $V^\perp$ such that the $V^\perp$-component of any vector is multiplied by a huge amount relative to the V-component. In particular, any vector that was not exponentially close to V to begin with, after multiplying by A ends up essentially in $V^\perp$. Our basic strategy here is to decrease the size of this singular value gap of A to be at most (merely) exponential, without much affecting any of the normalized transformed vectors. Our goal is to scale down the subspace $V^\perp$ to decrease the multiplicative eigenvalue gap. However, we must ensure that the vectors of X that are sufficiently close to $V^\perp$ after applying A do not end up being essentially in V. To achieve this, we consider a subspace W spanned by such problematic vectors and build an improved matrix AT such that T does not affect vectors in W, but rescales significantly vectors lying in a subspace R that is very close to $V^\perp$. Via this step, we can reduce the condition number of A to be appropriately bounded without affecting the mapping f_A significantly; after that, we can make do with a suitably precise rounding to obtain the output matrix A'.

Approximate Eigendecomposition in Strongly Polynomial Time. So far, we have assumed the availability of a routine for exact eigendecomposition. In fact, there are several places in the above intuitive overview of our algorithmic approach where we need to compute an eigenvalue decomposition of a matrix. This is required first when we need to find the initial eigenvalue gap in $M_A(X) \propto \sum_{x\in X_A} xx^\top$, and again later when we need to find the span of the large eigenvalues of $\sum_{x\in X_A^{\mathrm{B}}} xx^\top$. Unfortunately, computing exact eigenvalues is impossible in our model of computation (as doing so might require finding roots of high-degree polynomials). Fortunately, it is sufficient for us to find merely an *approximate* eigenvalue

decomposition of these matrices. A subtle and important point is that our required notion of approximation is *significantly stronger than the typical guarantees explicitly available in the literature*. Interestingly, we show that the desired strongly polynomial guarantees can be achieved in our model using some variation of the power iteration method. This requires a novel proof of correctness, that we provide here.

We are now ready to describe our strongly polynomial approximate eigendecomposition routine in tandem with a sketch of its analysis (see Proposition 4.1). The standard power iteration method says that in order to approximate the principal eigenvector of a symmetric, PSD matrix M, it suffices to multiply a random vector v by a large power t of M. If we express v as a linear combination of eigenvectors of M, then multiplying by a large power of M scales each of these components by an amount depending on the eigenvalue. It is not hard to see that if there is a reasonable gap between the largest and second largest eigenvalues, then the vector $M^t v$ will likely end up close to a multiple of the largest eigenvector. Once an approximate principal eigenvector is computed, one can attempt to repeat the same procedure, i.e., projecting onto the orthogonal subspace to find the second largest eigenvalue; and so on. This iterative procedure is known to succeed in finding approximations to the eigenvectors and eigenvalues in question, so long as the eigenvalues are not too close to each other. On the other hand, if M has (nearly) degenerate eigenspaces, then this method may fail to separate eigenvectors with very similar eigenvalues. However, in this (near-)degenerate case, such an approximation is usually not needed, as the eigenvalues are close to begin with. One can hope that the matrix $\hat{M}$ corresponding to the computed eigendecomposition is close to M in an appropriate sense. In particular, standard results (see, e.g., [47]) show how to compute such an $\hat{M}$ satisfying $\|M - \hat{M}\|_2 \leq \epsilon \|M\|_2$.

Unfortunately, this notion of approximation is not sufficient for our purposes. For example, in the case where the parameter β is small in our Forster algorithm, it is important for us to compute the spaces V' and W' to *very good accuracy*. This is because the linear transformation that we apply will multiply elements of V' by a large factor of roughly $1/\beta$. This means that we need to compute V' to error on the order of β in order to ensure the accuracy of our result. More generally, we will need a qualitatively stronger guarantee for our approximate eigenvalue decomposition. In particular, we need that for some small $\epsilon > 0$, for any vector v, it holds that $|v^\top (M - \hat{M})v| \leq \epsilon (v^\top M v)$. This means that if v lies in a space spanned by eigenvectors of M with very small eigenvalues (as V' is above), then we need that $\hat{M}v$ to be correspondingly small. Fortunately, we can obtain this much stronger "multiplicative" guarantee via power iteration. The intuitive reason this works is essentially because if we have a space V' spanned by eigenvectors of M with eigenvalues at most β, then multiplying a random vector v by powers of M reduces the size of the projection of v onto V' by a power of β. This means that power iteration produces vectors that are very nearly orthogonal to V' with the error in this approximation scaling with β.

1.3.2 Learning Halfspaces in Strongly Polynomial Time. As already mentioned in the introduction, we leverage our algorithm for approximate Forster transforms to obtain the first strongly polynomial algorithm for PAC learning halfspaces. It turns out that this approach goes through both in the realizable case (Definition 1.3) and in the presence of (semi-random) Massart noise on the labels. In fact, it is not difficult to verify that by plugging in our new Forster algorithm into the learning algorithm of [17], one directly obtains a strongly polynomial halfspace learner in the presence of Massart noise. For the sake of the completeness, here we focus on the realizable case and provide a simpler, self-contained algorithm and proof.

Note that it is without loss of generality to assume that the threshold of the target halfspace is zero (one can reduce the general case to the homogeneous case). The main challenge in PAC learning halfspaces is that the target halfspace may have very bad anti-concentration (aka "margin"). If the margin is not too small (i.e., at least inverse polynomial), simple iterative algorithms (e.g., perceptron) efficiently learn halfspaces (in strongly polynomial time). A natural idea is then to reduce the general case to the large margin case by appropriately transforming the data. A number of such reductions have been developed in the literature [7, 17, 19, 20]. The methods developed in [7, 19, 20] are inherently not strongly polynomial. Recently, [17] pointed out that one can use Forster transforms for this purpose.

For our purposes, we require a stronger guarantee than what is provided by the vanilla perceptron algorithm. Specifically, we want a learning algorithm for halfspaces that correctly classifies at least some reasonable fraction of points, if the points are guaranteed to be well-conditioned (for example, in the sense of being unit vectors with $\mathbf{E}[xx^\top] \approx I$). By using an approximate Forster algorithm, we can transform the input points in order to make them well-conditioned, while preserving the notion of halfspaces. We can then apply our learner to this set in order to learn a classifier that works on some reasonable fraction of the points. Repeating this procedure iteratively on the unclassified points eventually gives a halfspace learning algorithm.

More precisely, the modified perceptron algorithm of [20] is a strongly polynomial time algorithm with the following performance guarantee: given labeled examples consistent with an unknown linear classifier, the algorithm learns a classifier that correctly labels all points whose margin is not too small. It is not hard to see that, for points in approximate radial isotropic position, at least a $1/d$-fraction of points have not-too-small margin. Therefore, if we have a set of points in approximate radial isotropic position, the modified perceptron algorithm finds (in strongly polynomial time) an explicit halfspace that separates out a roughly $1/d$-fraction of the points all of the same sign. By standard generalization bounds, this gives us an algorithm that in strongly polynomial time learns a *partial classifier*, i.e., outputs a partial function that correctly classifies an $\Omega(1/d)$-fraction of the points while misclassifying an $O(\gamma/d)$-fraction. In other words, this procedure produces a partial classifier that labels at least a $1/d$-fraction of points and misclassifies at most a γ-fraction of these points.

To learn an arbitrary halfspace, we use our approximate Forster transform to put the points in approximate radial

isotropic position without changing the notion of a halfspace on them. We then apply the above partial learner to these new points in order to obtain a non-trivial partial classifier that makes mistakes on only a γ-fraction of its classified set. We repeat this process on the unclassified points, using a new approximate Forster transform, to learn a non-trivial fraction of the unclassified points. Repeating this procedure iteratively as necessary, we eventually obtain a partial classifier that produces an answer on essentially all points of the domain and only makes mistakes on a γ-fraction of them.

1.4 Related Work

Comparison to Strongly Polynomial Algorithm for Matrix Completion. It is worthwhile to compare our techniques for the Forster transform to [36], who developed the first strongly polynomial time algorithm for the matrix scaling problem. To put this problem in terms more analogous to ours, one is given a set of d vectors $x_1, x_2, \ldots, x_d$ in $\mathbb{R}^d$. The goal is to find a *diagonal* matrix A such that if $y_i := Ax_i/\|Ax_i\|_1$ is the ℓ_1 normalization of Ax_i, then the absolute deviation of the j^{th} coordinates of the y's around 0 are (approximately) the same for all j. In particular, it should hold that $\sum_{i=1}^{d} |(y_i)_j| \approx 1$ for all $1 \leq j \leq d$. Note that for our problem, we have n (possibly greater than d) vectors, A can be *any* matrix, we take y_i to be the ℓ_2 normalization and we want the mean square deviation of the y's in *any* direction (not just along coordinate axes) to be approximately the same.

The algorithm in [36] works roughly as follows. We construct A through an iterative sequence of improvements. Given a specific A, we compute the appropriate values of y_i and then compute the absolute deviations of each coordinate. If these are all close to each other, we are done. Otherwise, by sorting the deviations and finding the largest gap, we can split our coordinates into two sets, B and S, so that the deviation of any coordinate in B is substantially larger than the deviation of any coordinate in S. One then defines the diagonal matrix C to be $(1+\delta)$ on the coordinates in S and 1 on the coordinates in B, and replaces A by $A' := CA$. It is not hard to see that by doing this, one increases the deviations along all coordinates in S while decreasing it along all coordinates in B (and keeping the total sum of deviations the same). By picking δ carefully, [36] show that the variance of these coordinate-wise deviations can be decreased by some polynomial amount in each step. Thus, by iterating this method a polynomial number of times, one obtains a scaling where the coordinate-wise deviations are sufficiently close.

The starting point for our algorithm is somewhat similar. Given a matrix A, we try to find a matrix C such that the matrix $A' := CA$ is closer to satisfying our condition (in the sense that $\|M_{A'}(X)\|_F$ should be smaller than $\|M_A(X)\|_F$ by an additive inverse polynomial term). To do this, we compute subspaces V_S and V_B (by finding an eigenvalue gap in $M_A(X)$) such that the variance of the $y_i := Ax_i/\|Ax_i\|_2$ in any direction along V_B is substantially larger than along any direction in V_S. Ideally, we would like to take $C = I + \alpha I_{V_S}$ for some carefully selected α. While this does only increase the variance in directions along V_S and decrease it along V_B, in our setting this *also* creates off-diagonal terms that increase our potential. While it is always possible to ensure that this error does not overwhelm the progress we make by taking α small enough, in some cases (particularly where all of the y's are either very close to lying in V_B or very close to lying in V_S), this is not compatible with making polynomial progress in each step. In this other case, we need to use a subtly different method for finding C in order to minimize the contribution of these off-diagonal terms. Furthermore, unlike in [36], the matrices C used might have large numerical complexity (perhaps on the order of the complexity of A). If we naively apply the iterative algorithm as is, it might lead to computations involving matrices with exponentially large bit complexity. In order to fix this, we also need to add a rounding step, whereby in each stage we reduce the numerical complexity of A down to some manageable level but without substantially affecting our potential.

Comparison to Other Data Transformations. The Forster transform is one of several data transformations that have been studied in the literature to make a dataset "well-conditioned". Here we explain two similar in spirit such transformations, namely the "outlier removal" technique [7, 19] and the rescaling method of [20]. Both of these techniques have been used to obtain weakly polynomial learners for halfspaces with random noise.

The "outlier-removal" technique was introduced in [7] and was significantly refined by Dunagan and Vempala [19]. Given a dataset X and a parameter $\beta > 0$, a point in X is called a β-outlier if there exists a direction v such that the squared length of x along v is more than β times the average squared length of X along v. The goal of the method is to efficiently find a large subset of $X' \subseteq X$ such that X' has no β-outliers, for as small β as possible. This would give a reasonable sized sub-distribution on which the desired anti-concentration holds. As shown in [19], the parameter β (which affects the quality of the resulting anti-concentration) needs to scale polynomially with the bit complexity b of the dataset X. Consequently, the resulting runtimes in applications of this method will be *inherently weakly polynomial*. Interestingly, this is the reason that the (random noise tolerant) halfspace learner of [7] is only weakly polynomial.

A different algorithm for learning halfspaces with random classification noise is implicit in the *rescaled perceptron* algorithm of Dunagan and Vempala [20] for efficiently solving linear programs (see also [5]). The key ingredient of their approach is a rescaling step that linearly transforms the data so that, roughly speaking, the margin increases in each iteration by a factor of $1 + 1/d$. Since the initial margin scales with the bit complexity, so does the total number of iterations. (Since this leads to a proper learning algorithm, a dependence on the bit complexity is expected; otherwise, one would obtain a strongly polynomial algorithm for LP!)

Strongly Polynomial Special Cases of LP. A line of work, starting in the 80s, has developed strongly polynomial time algorithms for interesting special cases of LP, including minimum cost circulations [27, 46, 51], min cost flow and multi-commodity flow problems [52, 55], and generalized flow maximization [44, 45, 56] (see also [12, 13]). Strongly polynomial

time algorithms have also been developed for certain structured convex programs, see, e.g. [26, 57] in the context of equilibrium computation, and [36] for matrix scaling.

1.5 Organization

In Section 2, we record basic notation and facts that will be used throughout this paper. Section 3 presents our Forster decomposition algorithm, assuming exact eigendecomposition and ignoring bit complexity issues. Section 4 establishes our strongly polynomial guarantees for approximate eigendecomposition. Section 5 shows that we can efficiently round the entries of the underlying matrix without losing much in the desired guarantees. Section 6 presents our strongly polynomial halfspace learning algorithm. Finally, in Section 7 we summarize our results and provide directions for future work. Due to space limitations, most proofs have been deferred to the full version of this work.

2 PRELIMINARIES

Basic Notation. We use $\mathbb{Z}_+$ to denote the non-negative integers, $\mathbb{R}^d$ for the d-dimensional real coordinate space, $\mathbb{R}^d_*$ for $\mathbb{R}^d \setminus \{\mathbf{0}\}$, and $\mathbb{S}_d$ for the unit ℓ_2-sphere. For a set $S \subset \mathbb{R}$, we will denote $\max(S) \stackrel{\text{def}}{=} \max_{x\in S} x$ and $\min(S) \stackrel{\text{def}}{=} \min_{x\in S} x$.

For $x \in \mathbb{R}^d$, we use $\|x\|_2$ to denotes the ℓ_2-norm of x. We use $\mathrm{tr}(\cdot)$, $\|\cdot\|_F$, and $\|\cdot\|_2$ for the trace, Frobenius norm, and spectral norm of a square matrix. For matrices $A, B \in \mathbb{R}^{d\times d}$, we write $A \succeq B$ (or $B \preceq A$) to denote that $A - B$ is positive semidefinite (PSD). We use I for the $d \times d$ identity matrix, where the dimension will be clear from the context. If $M \in \mathbb{R}^{d\times d}$ is a PSD matrix, we denote by $\lambda_i(M)$ and $q_i(M)$ the i-th largest eigenvalue and corresponding eigenvector of M. That is, $\lambda_1(M) \geq \lambda_2(M) \geq \ldots \geq \lambda_d(M) \geq 0$ and $Mq_i(M) = \lambda_i(M)q_i(M)$ for all $i \in [d]$. We denote by $\Lambda(M)$ the set of eigenvalues of M and $Q(M)$ the set of eigenvectors. That is, $\Lambda(M) = \{\lambda_i(M), i \in [d]\}$ and $Q(M) = \{q_i(M), i \in [d]\}$. For $S \subseteq [d]$, we denote $\Lambda_S(M) = \{\lambda_i(M), i \in S\}$ and $Q_S(M) = \{q_i(M), i \in S\}$.

For a finite set of vectors $S \subset \mathbb{R}^d$, we use $\mathrm{span}(S)$ for their span. For a subspace $V \subset \mathbb{R}^d$, we use $\dim(V)$ for its dimension and $V^\perp$ for its orthogonal complement. For $x \in \mathbb{R}^d$ and a subspace V, we will denote by $\mathrm{proj}_V x$ the projection of x onto V. If $V = \mathrm{span}(S)$, we will sometimes use $\mathrm{proj}_S x$ to denote $\mathrm{proj}_V x$. For conciseness, we sometimes use $x^{(V)}$ for $\mathrm{proj}_V x$. We denote by I_V the $d \times d$ matrix with eigenvalues 1 in V and 0 in $V^\perp$ (the projection of I onto V).

Additional Notation. For a dataset $X \subset \mathbb{R}^d_*$ of size $|X| = n$ and a linear transformation $A \in \mathbb{R}^{d\times d}$, let $f_A : \mathbb{R}^d_* \to \mathbb{S}_d$ be defined by $f_A(x) \stackrel{\text{def}}{=} \frac{Ax}{\|Ax\|_2}$. We aim to find an invertible $A \in \mathbb{R}^{d\times d}$ such that f_A brings a given dataset X in (approximate) radial isotropic position. We denote $f_A(X) \stackrel{\text{def}}{=} \left\{\frac{Ax}{\|Ax\|_2} \mid x \in X\right\}$ We will use various "covariance-like" matrices for the initial dataset X and its subsets. For $X' \subseteq X$, we denote $M_A(X') \stackrel{\text{def}}{=} (1/n)\sum_{x\in X'} f_A(x)f_A(x)^\top$. For subspaces $V_1, V_2 \subset \mathbb{R}^d$ and $X' \subseteq X$, we denote by $M_A^{V_1,V_2}(X') \stackrel{\text{def}}{=} (1/n)\sum_{x\in X'} f_A^{(V_1)}(x)f_A^{(V_2)}(x)^\top$, where we used the shorthand notation $y^{(V)} = \mathrm{proj}_V y$. Note that the normalization factor is fixed in both cases.

3 APPROXIMATE FORSTER TRANSFORM IN STRONGLY POLYNOMIAL TIME

In this section, we describe and analyze our algorithm that either computes an approximate Forster transform of a given dataset or certifies that no Forster transform exists. There are two technical caveats in the algorithm presented in this section: First, we assume the existence of exact routines for matrix eigendecomposition. Second, we do not bound the bit complexity of the associated numbers. Both of these technical issues are handled in subsequent sections.

3.1 Algorithm Pseudocode

The algorithm aims to find a matrix $A \in \mathbb{R}^{d\times d}$ such that the transformation $f_A : \mathbb{R}^d \to \mathbb{R}^d$ brings the set X in (approximate) radial isotropic position. Starting from the initial guess $A = I$, the algorithm iteratively improves the current matrix A until the desired approximation is obtained or a proper subspace W of $\mathbb{R}^d$ is found such that $|X \cap W|/n \geq \dim(W)/d$.

Algorithm 1 Algorithm for computing Forster Transform

function ForsterTransform (set $X \subset \mathbb{R}^d_*$ of n points, accuracy parameter ϵ)
  Let $A \leftarrow I$ ▷ Initialization of transformation matrix A
  $M_A \leftarrow M_A(X) = (1/n)\sum_{x\in X} f_A(x)f_A(x)^\top$
  while $\|M_A\|_F^2 > \frac{1}{d} + \frac{\epsilon^2}{d^2}$ **do**
    Set $A \leftarrow$ ImproveTransform(A, X)
    Set $M_A \leftarrow M_A(X) = (1/n)\sum_{x\in X} f_A(x)f_A(x)^\top$
  return A

3.2 Analysis of Algorithm 1

Our Potential Function. Our algorithm measures the improvements between consecutive iterations using the potential function

$$\Phi_X(A) \stackrel{\text{def}}{=} \|M_A\|_F^2 \tag{1}$$

corresponding to the squared Frobenius norm of the matrix

$$M_A \stackrel{\text{def}}{=} M_A(X) \stackrel{\text{def}}{=} (1/n)\sum_{x\in X} f_A(x)f_A(x)^\top .$$

Recall that approximate radial isotropy condition amounts to the condition $\frac{1-\epsilon}{d} I \preceq M_A \preceq \frac{1+\epsilon}{d} I$. Equivalently, we want that $\|M_A - \frac{1}{d}I\|_2 \leq \frac{\epsilon}{d}$ or that the eigenvalues of M_A lie in $[\frac{1-\epsilon}{d}, \frac{1+\epsilon}{d}]$. This is guaranteed to hold when the potential function becomes less than $1/d + \epsilon^2/d^2$, as shown in the following lemma.

Lemma 3.1. *Consider any dataset $X \subseteq \mathbb{R}^d_*$ and any full-rank matrix $A \in \mathbb{R}^{d\times d}$. The following properties hold for the potential $\Phi_X(A) = \|M_A\|_F^2$.*

(1) $1/d \leq \Phi_X(A) \leq 1$.
(2) If $\Phi_X(A) \leq 1/d + \epsilon^2/d^2$ for some $\epsilon \in (0, 1)$, then for every eigenvalue λ of M_A it holds that $|\lambda - 1/d| \leq \epsilon/d$.

Algorithm 2 Find Improved Transform Matrix

function ImproveTransform (current matrix $A \in \mathbb{R}^{d\times d}$, $X \subset \mathbb{R}^d_*$, accuracy parameter ϵ)
  Set $M_A \leftarrow M_A(X) = (1/n)\sum_{x\in X} f_A(x) f_A(x)^\top$
  Compute the set of eigenvalues, $\Lambda = \Lambda(M_A)$, and eigenvectors, $Q = Q(M_A)$, of M_A.
  Set $\gamma \leftarrow O(\frac{\epsilon^2}{d^4 n^2})$, where $n := |X|$
  Partition (Λ, Q) into two sets of eigenvalues and corresponding eigenvectors, (Λ_B, Q_B) and (Λ_S, Q_S), maximizing $\min(\Lambda_B) - \max(\Lambda_S)$.
  ▷ Consider the Following Two Cases
  if $\exists x \in X$ s.t. $\left\|\text{proj}_{Q_B} f_A(x)\right\|_2, \left\|\text{proj}_{Q_S} f_A(x)\right\|_2 \geq \gamma$ **then**
    Set $U \leftarrow \text{span}(Q_S)$.
    Set $\alpha \leftarrow \frac{\epsilon}{8nd^3}$.
  else
    $X^B \leftarrow \{x \in X : \|\text{proj}_{Q_B} f_A(x)\|_2 \geq \gamma\}$.
    $M_A^B \leftarrow M_A(X^B) \stackrel{\text{def}}{=} (1/n)\sum_{x\in X^B} f_A(x) f_A(x)^\top$.
    Let Q_b and Q_s be the sets of top $|Q_B|$ and bottom $|Q_S|$ eigenvectors of M_A^B respectively.
    $U \leftarrow \text{span}(Q_s)$.
    $\beta \leftarrow \max_{x\in X^B} \|f_A^{(U)}(x)\|_2$.
    if $\beta = 0$ **then**
      ▷ No Forster Transform Exists
      Output the subspace $\text{span}(Q_b)$.
    else ▷ Case where $\beta > 0$
      Set $\alpha \leftarrow \frac{1}{\beta}\epsilon/(3d^2 n) - 1$
  return $A' := (I + \alpha I_U)\, A$

(3) If $\Phi_X(A) > 1/d + \epsilon^2/d^2$ for some $\epsilon \in (0,1)$, then (a) there exists an eigenvalue λ of M_A such that $|\lambda - 1/d| > \epsilon/d^2$ and (b) there exists a pair of consecutive eigenvalues λ_i and λ_{i+1} of M_A such that $\lambda_i - \lambda_{i+1} > \epsilon/d^3$.

Bounding the Decrease in Potential. We now proceed with the analysis. We show that the algorithm ImproveTransform either correctly determines that no Forster transform exists or computes a transformation matrix with significantly reduced potential value. This statement implies correctness and simultaneously allows us to bound the running time of our algorithm.

The main result of this section is the following proposition.

Proposition 3.2. *Let $A \in \mathbb{R}^{d\times d}$ be a full-rank matrix and X be a set of n points in $\mathbb{R}^d_*$ such that $\Phi_X(A) > \frac{1}{d} + \frac{\epsilon^2}{d^2}$ for some $\epsilon \in (0,1)$. The algorithm* ImproveTransform *returns a matrix A' such that*

$$\Phi_X(A) - \Phi_X(A') \geq \Omega(\epsilon^5/(n^5 d^{11})) \tag{2}$$

or correctly determines that no Forster Transform of X exists, in which case it returns a subspace W such that $|X \cap W| > (n/d)\dim(W)$.

In the rest of this section, we provide a proof of Proposition 3.2.

Assuming that a Forster transform of X exists, the algorithm ImproveTransform returns the matrix $A' = (I + \alpha I_V)\, A$, where $V \subset \mathbb{R}^d$ is an appropriate proper subspace of $\mathbb{R}^d$ and $\alpha \in \mathbb{R}_{>0}$ is a carefully selected parameter (that depends on the structure of the dataset X). The algorithm distinguishes two cases: In the first case, α is a small positive quantity, equal to $\epsilon/(8nd^3)$, see Line 8 in Algorithm 2. In the second case, α is set to $\frac{1}{\beta}\epsilon/(3d^2 n) - 1$, and can be significantly larger than 1 as it depends on a small parameter β which is a function of the dataset X. See Line 18 in Algorithm 2.

3.2.1 A Useful Structural Result. We will use the notation $M_A = M_A(X)$ and $M_{A'} = M_{A'}(X)$. To bound the desired quantity, $\Phi_X(A) - \Phi_X(A') = \|M_A\|_F^2 - \|M_{A'}\|_F^2$, we will make essential use of the following key lemma:

Lemma 3.3. *For any $X \subset \mathbb{R}^d_*$ and any full-rank matrix $A \in \mathbb{R}^{d\times d}$ the following holds. For any subspace $V \subset \mathbb{R}^d$ and any scalar $\alpha > 0$, for $A' \stackrel{def}{=} (I + \alpha I_V)A$, we have that*

$$\Phi_X(A) - \Phi_X(A') \geq 2(\lambda_k(M_A^{V^\perp, V^\perp}) - \lambda_1(M_A^{V,V}) - 2D_f)D_f - 2\|M_{A'}^{V,V^\perp}\|_F^2\,, \tag{3}$$

where $k = \dim(V^\perp)$ and

$$D_f \stackrel{def}{=} \frac{1}{n}\sum_{x\in X}\left(\|f_{A'}^{(V)}(x)\|_2^2 - \|f_A^{(V)}(x)\|_2^2\right)\,.$$

Lemma 3.3 bounds the improvement in potential in terms of two opposing contributions. On the one hand, there is a decrease in the potential proportional to the amount of mass D_f transferred from the subspace $V^\perp$ to the subspace V times the eigenvalue gap between the subspaces V and $V^\perp$. On the other hand, there is an increase in potential due to the cross terms $V \times V^\perp$ that get created after the transformation by A'.

In the following two subsections, we analyze the two cases of ImproveTransform separately. Note that ImproveTransform requires that we be able to do exact singular value decompositions in order to compute the subspace U. Our final algorithm will not be able to do this exactly and will need to make do with an approximate singular value decomposition (see Section 4). In order to make our extension easier, we will show that the potential decrease holds even when U is replaced by some V which satisfies some approximation of the properties that U does.

3.2.2 Case I: $\exists x \in X$ s.t. $\|\text{proj}_{Q_B} f_A(x)\|_2, \|\text{proj}_{Q_S} f_A(x)\|_2 \geq \gamma$. To analyze this case, we prove the following proposition:

Proposition 3.4. *Suppose that X is a set of n points in $\mathbb{R}^d_*$ and A an invertible $d \times d$ matrix. Suppose that $V \subset \mathbb{R}^d$ is a subspace so that for $\alpha, \rho > 0$ with $\alpha \leq \epsilon/(64nd^3)$:*

(1) The maximum over $x \in X$ of $\min(\|f_A^{(V)}(x)\|_2, \|f_A^{(V^\perp)}(x)\|_2)$ equals ρ.
(2) $\lambda_{\min}(M_A^{V^\perp V^\perp}(X)) - \lambda_{\max}(M_A^{VV}(X)) \geq \frac{\epsilon}{2d^3}$.
(3) $\|M_A^{VV^\perp}(X)\|_F \leq \alpha\rho$.

Then for $C = (I + \alpha I_V)A$ we have that $\Phi_X(C) \leq \Phi_X(A) - \rho^2\epsilon/(8nd^3)$.

We note that if $\Phi_X(A) > \frac{1}{d} + \frac{\epsilon^2}{d^2}$, then by Lemma 3.1 part 3, the difference between the largest and smallest eigenvalues of $M_A(X)$ will be at least ϵ/d^2, and therefore the largest eigenvalue gap will be at least ϵ/d^3. Thus, for V taken to be the U given in Algorithm 2, Property 2 will hold. Furthermore, for as U is an eigenspace of $M_A(X)$, $M_A^{U,U^\perp}(X) = \mathbf{0}$ and Property 3 will hold.

The rest of this section will be devoted to proving Proposition 3.4.

To bound below the improvement in potential, we will make essential use of Lemma 3.3. We bound the relevant quantities in the following lemmas.

Lemma 3.5. *Let* $D_f = \frac{1}{n}\sum_{x\in X}\left(\|f_C^{(V)}(x)\|_2^2 - \|f_A^{(V)}(x)\|_2^2\right)$. *We have that* $\frac{\alpha\rho^2}{2n} \le D_f \le 2\alpha$.

Finally, we bound $\|M_C^{V,V^\perp}\|_F^2$ from above in the following lemma:

Lemma 3.6. *We have that* $\|M_C^{V,V^\perp}\|_F^2 \le 4\alpha^2\rho^2$.

Combining the above lemmas, we obtain that

$$\Phi_X(A) - \Phi_X(C) \ge \left(\frac{\epsilon}{2d^3} - 4\alpha\right)\frac{\alpha\rho^2}{n} - 8\alpha^2\rho^2 \ .$$

We note that so long as $\alpha \le \epsilon/(64nd^3)$ the above is at least

$$\left(\alpha\rho^2\right)\left(\left(\frac{\epsilon}{4d^3}\right)\frac{1}{n} - 8\alpha\right) \ge \alpha\rho^2\epsilon/(8nd^3) \ .$$

This completes our proof of Proposition 3.4.

3.2.3 Case II: For all $x \in X$, either $\|\mathrm{proj}_{Q_B} f_A(x)\|_2 \le \gamma$ or $\|\mathrm{proj}_{Q_S} f_A(x)\|_2 \le \gamma$. In this case, all points $x \in X$ lie within a γ margin from the subspaces spanned by the vectors Q_B and Q_S. The algorithm updates the matrix A by considering only the set of "big" points X^B, i.e., the points in X whose images under f_A have sufficiently large projections on the subspace spanned by the large eigenvectors of M_A. In more detail, instead of using the eigenvectors Q_B, Q_S of the matrix $M_A = M_A(X)$, the algorithm uses the eigenvectors Q_b, Q_s of the matrix $M_A(X^B) = (1/n)\sum_{x\in X^B} f_A(x)f_A(x)^\top$, setting $U = \mathrm{span}(Q_s)$. This is done to ensure that the cross-terms $M_A^{U,U^\perp}(X^B)$ start out at 0 initially and remain small despite significant rescaling of the subspace U. Moreover, despite the change in the definition, we show (in Claim 3.8 and Claim 3.9) that the corresponding subspaces U and $U^\perp$ satisfy a similar margin condition to the subspaces spanned by Q_B and Q_S and that there is still a significant eigenvalue gap between U and $U^\perp$. The margin condition is shown in Claim 3.8 and Claim 3.9, and the eigenvalue bounds are proven in Lemmas 3.11 and 3.10.

These properties will allow us to bound the decrease in potential in this case. We will show the following result.

Proposition 3.7. *Suppose that X is a set of n points in $\mathbb{R}^d_*$ and A an invertible $d\times d$ matrix. Suppose that for some $k < n$ that $\lambda_k(M_A(X)) - \lambda_{k+1}(M_A(X)) \ge \epsilon/(2d^3)$. Let W be the span of the $d-k$ smallest eigenvalues of $M_A(X)$.*

Suppose furthermore that for some γ at most a sufficiently small multiple of $\epsilon^2/(d^4n^2)$ that every $x \in X$ satisfies

$$\min(\|f_A^{(W)}(x)\|_2, \|f_A^{(W^\perp)}(x)\|_2) \le \gamma \ .$$

Let X^B denote the set of $x \in X$ so that $\|f_A^{(W)}(x)\|_2 \le \gamma$ and $X^S = X\backslash X^B$. Let $0 < \delta < \gamma$. Suppose that $V \subset \mathbb{R}^d$ is a $(d-k)$-dimensional subspace so that:

(1) $\mathrm{tr}(M_A^{V,V}(X^B)) \le \mathrm{tr}(M_A^{W,W}(X^B)) + \delta^2$,
(2) $\mathrm{tr}(M_A^{V^\perp,V^\perp}(X^B)) \ge \mathrm{tr}(M_A^{W^\perp,W^\perp}(X^B)) - \delta^2$,
(3) $\lambda_k(M_A^{V^\perp,V^\perp}(X^B)) \ge \lambda_k(M_A(X^B)) - \delta$,
(4) $\|M_A^{V,V^\perp}(X^B)\|_F \le \beta\delta$,

where $\beta = \max_{x\in X^B}\|f_A^{(V)}(x)\|_2$. Then if $\beta = 0$, $V^\perp$ contains more than kn/d elements of X. Otherwise, setting $\alpha = \epsilon/(3\beta d^2 n) - 1$ and $C = (I + \alpha I_V)A$, we have that

$$\Phi_X(C) \le \Phi_X(A) - \Omega(\epsilon^3/(d^7n^3)) \ .$$

We note that if V is taken to be the space of the bottom $d-k$ eigenvalues of $M_A(X^B)$ that the above properties trivially hold with $\delta = 0$. Properties 1 and 2 follow from the variational characterization of eigenspaces. Properties 3 and 4 hold trivially.

We know that elements of X^B are close to $W^\perp$ and elements of X^S are close to W. We will need to claim that elements of X^B are also close to $V^\perp$ and elements of X^S are close to V. We establish this in the next two claims.

Claim 3.8. *We have that $\frac{1}{n}\sum_{x\in X^B}\|f_A^{(V)}(x)\|_2^2 \le \gamma^2 + \delta^2$. In particular, for any $x \in X^B$, it holds that $\|f_A^{(V)}(x)\|_2 \le \sqrt{2n}\gamma$.*

Claim 3.9. *We have that $\frac{1}{n}\sum_{x\in X\backslash X^B}\|f_A^{(V^\perp)}(x)\|_2^2 \le \gamma^2 + \delta^2$. In particular, for any $x \in X\backslash X^B$, it holds that $\|f_A^{(V^\perp)}(x)\|_2 \le \sqrt{2n}\gamma$.*

The case that $\beta > 0$: Here we assume that $\beta > 0$ and show that we can obtain an improvement in our potential function. To bound below the improvement in potential, we will make essential use of Lemma 3.3, and we bound the relevant quantities in a sequence of lemmas.

We begin by bounding below $\lambda_k(M_A^{V^\perp,V^\perp}(X))$. In particular, we show that it is nearly as big as $\lambda_k(M_A(X))$. Morally, this holds because

$$\lambda_k(M_A(X)) = \lambda_k(M_A^{W^\perp,W^\perp}(X)) \approx \lambda_k(M_A^{V^\perp,V^\perp}(X)) \ .$$

Formally, we have the following.

Lemma 3.10. *We have that $\lambda_k(M_A^{V^\perp,V^\perp}(X)) \ge \lambda_k(M_A(X)) - 4\gamma$, for $k = \dim(V^\perp)$.*

Next we bound from above $\lambda_1(M_A^{V,V}(X))$. In particular, we show that it is not much larger than $\lambda_{k+1}(M_A(X))$. Morally, this holds because

$$\lambda_{k+1}(M_A(X)) = \lambda_1(M_A^{W,W}(X)) \approx \lambda_1(M_A^{V,V}(X)) \ .$$

Formally, we have the following.

Lemma 3.11. *We have that $\lambda_1(M_A^{V,V}(X)) \le \lambda_{k+1}(M_A(X)) + 8\gamma$.*

Together Lemmas 3.10 and 3.11 show that $\lambda_k(M_A^{V^\perp,V^\perp}(X)) - \lambda_1(M_A^{V,V}(X))$ is nearly as large as $\lambda_k(M_A(X)) - \lambda_{k+1}(M_A(X)) \ge \epsilon/(2d^3)$.

Next we bound the off-diagonal terms of the transformed vectors.

Lemma 3.12. *We have that* $\|M_C^{V,V^\perp}\|_F^2 \le ((1+\alpha)^3\beta^3 + (1+\alpha)\beta\delta + 2\gamma)^2$.

Finally, we need to bound D_f, showing that it is neither too big nor too small. This follows by noting that the greatest amount that any vector was modified is on the order of $\alpha\beta$. In particular, we have that:

Lemma 3.13. *We have that* $\frac{1}{n}\left(\frac{(1+\alpha)^2\beta^2}{1+(1+\alpha)^2\beta^2} - \gamma^2\right) \le D_f \le (1+\alpha)^2\beta^2 + 2\gamma^2$, *where*

$$D_f = \frac{1}{n}\sum_{y\in f_C(X)} \|y^{(V)}\|_2^2 - \frac{1}{n}\sum_{y\in f_A(X)} \|y^{(V)}\|_2^2 \ .$$

Combining the Lemmas 3.10, 3.11, 3.12, 3.13 with Lemma 3.3 and setting $\eta = (1+\alpha)\beta = \epsilon/(3d^2n)$, we get that

$$\begin{aligned}
&\Phi_X(A) - \Phi_X(C) \ge \\
&\ge \left(\frac{\epsilon}{2d^3} - 16\gamma - 2\eta^2\right)\left(\frac{\eta^2 - 2\gamma^2}{n}\right) - 2(\eta^3 + \eta\delta + 2\gamma)^2 \\
&\ge \left(\frac{\epsilon}{2d^3} - 16\gamma - 2\eta^2\right)\left(\frac{\eta^2 - 2\gamma^2}{n}\right) - 6\eta^6 - 6(\eta\delta)^2 - 24\gamma^2 .
\end{aligned}$$

Given that both γ and η^2 are less than a sufficiently small multiple of $\frac{\epsilon}{d^3}$, the above is at least

$$\frac{\epsilon\eta^2}{3d^3n} - 6\eta^6 - 6(\eta\delta)^2 - 24\gamma^2 .$$

Given that δ is less than a sufficiently small multiple of $\frac{\epsilon}{dn}$, and γ a small multiple of $\epsilon^2/(d^4n^2)$, this is

$$\Omega(\epsilon\eta^2/(d^3n)) = \Omega(\epsilon^3/(d^7n^3)).$$

The case of $\beta = 0$: We now argue that in the case that $\beta = 0$, no Forster transform exists, as the algorithm correctly identifies a subspace of dimension k containing more than a k/d fraction of the points of X.

Since we have that $\lambda_k(M_A(X)) - \lambda_{k+1}(M_A(X)) \ge \epsilon/(2d^3)$ by assumption, either $\lambda_k(M_A(X)) \ge \frac{1}{d} + \frac{\epsilon}{4d^3}$ or $\lambda_{k+1}(M_A(X)) \le \frac{1}{d} - \frac{\epsilon}{4d^3}$. The algorithm returns the subspace $V^\perp$ of dimension k, which contains all points X^{B} as $\beta = 0$. We claim that $|X^{\mathrm{B}}|/n > k/d$, which would complete our analysis. This is essentially because the large eigenvalues of $M_A(X)$ on $V^\perp$ imply that X^{B} must have many points.

We consider two subcases below.

Case 1: $\lambda_k(M_A(X)) \ge \frac{1}{d} + \frac{\epsilon}{4d^3}$. In this case, we have that

$$\begin{aligned}
\frac{|X^{\mathrm{B}}|}{n} &= \mathrm{tr}(M_A(X^{\mathrm{B}})) \ge k\,\lambda_k(M_A(X^{\mathrm{B}})) \\
&\ge k\lambda_k(M_A^{V^\perp,V^\perp}(X^{\mathrm{B}})) - k\delta \\
&\ge k\lambda_k(M_A^{V^\perp,V^\perp}(X)) - k\delta - 2k\gamma^2 \\
&\ge k\lambda_k(M_A(X)) - 7k\gamma \\
&\ge k/d + k(\epsilon/(4d^3) - 7\gamma) > k/d \ ,
\end{aligned}$$

where the second line above follows from Property 3, the third line from Claim 3.9. The fourth line follows from Lemma 3.10, and the rest from $\epsilon/d^3 \gg \gamma > \delta$.

Case 2: $\lambda_{k+1}(M_A(X)) \le \frac{1}{d} - \frac{\epsilon}{4d^3}$. In this case, we have that

$$\frac{|X^S|}{n} = \mathrm{tr}(M_A(X^S)) = \mathrm{tr}(M_A^{V,V}(X^S)) + \mathrm{tr}(M_A^{V^\perp,V^\perp}(X^S)).$$

By Claim 3.9 we have that $\mathrm{tr}(M_A^{V^\perp,V^\perp}(X^S)) \le 2\gamma^2$. On the other hand since $\beta = 0$, all elements of X^{B} are orthogonal to V and thus

$$\mathrm{tr}(M_A^{V,V}(X^S)) = \mathrm{tr}(M_A^{V,V}(X)).$$

This is at most $(k-d)\lambda_1(M_A^{V,V}(A))$, which by Lemma 3.11 is at most $(k-d)\lambda_{k+1}(M_A(X)) + 8d\gamma$. Combining with the above, we get that

$$\begin{aligned}
\frac{|X^S|}{n} &\le (k-d)\lambda_{k+1}(M_A(X)) + 10d\gamma \\
&\le (k-d)/d - (k-d)\epsilon/(4d^3) + 10d\gamma < (k-d)/d \ .
\end{aligned}$$

Hence, in this case as well $|X^{\mathrm{B}}|/n = 1 - |X^S|/n > k/d$.

This completes the proof of Proposition 3.7.

This completes the proof of Proposition 3.2. □

4 APPROXIMATE EIGENDECOMPOSITION

In this section, we give a simple algorithm that computes an approximate eigendecomposition with multiplicative error guarantees in strongly polynomial time.

Proposition 4.1. *Given a $d \times d$ PSD matrix M, an accuracy parameter $\epsilon > 0$ and a failure probability $\delta > 0$, there is an algorithm that computes orthogonal vectors $q_1, \ldots, q_d$ and scalars a_i such that the matrix $\hat{M} = \sum_{i=1}^d a_i q_i q_i^\top$ satisfies the following: for all $v \in \mathbb{R}^d$, it holds that*

$$|v^\top(M - \hat{M})v| \le \epsilon\,(v^\top M v) \ .$$

The algorithm performs $\mathrm{poly}(d/\epsilon, \log(1/\delta))$ *arithmetic operations on* $\mathrm{poly}(d/\epsilon, \log(1/\delta), b)$*-bit numbers, where b is the bit complexity of the entries of M.*

Our algorithm is presented in pseudocode below.

Algorithm 3 Computing the approximate eigendecomposition of a matrix M

1: **function** EigenDecomposition(Matrix $M_{d\times d}$, accuracy parameter ϵ, error probability δ)
2: Let A be a random $d \times d$ matrix where the entries are i.i.d. uniform samples from $\{1, 2, \ldots, N\}$, for N at least a sufficiently large constant multiple of d/δ.
3: Let $w_1, w_2, \ldots, w_d$ be the column vectors of M^tA, for t a sufficiently large constant multiple of $d^6/\epsilon^2 \log(d/\delta)$.
4: **for** $i = 1$ to d **do**
5: Let q_i be the projection of w_i onto the orthogonal complement of $w_1, w_2, \ldots, w_{i-1}$.
6: Let $a_i = 0$ if $q_i = 0$ and $a_i = q_i^\top M q_i/(q_i \cdot q_i)$ otherwise.
7: **return** $\{a_i, q_i\}$

It is easy to see that this algorithm runs in the appropriate time and bit-complexity bounds. The difficulty is in showing that the resulting $\hat{M}$ satisfies the desired error bounds. The proof is given in the full version.

5 MATRIX ROUNDING

In this section, we establish our efficient rounding procedure, establishing the following:

Theorem 5.1 (Matrix Rounding). *There is an algorithm that given (i) a set of n points $X \subseteq \{-2^b, \ldots, 2^b\}^d \setminus \{\mathbf{0}\}$ with $b \in \mathbb{Z}_+$, so that X spans $\mathbb{R}^d$, (ii) a full-rank $d \times d$ matrix $A \in \{-2^{rb}, \ldots, 2^{rb}\}^{d\times d}$, with $r \in \mathbb{Z}_+$, and (iii) an accuracy parameter $\epsilon \in (0,1)$, outputs a matrix A' with integer entries of magnitude at most $(\frac{d}{\epsilon})^{O(d^3 b)}$ such that for all points $x \in X$ it holds $\|f_A(x) - f_{A'}(x)\|_2 \le \epsilon$. The algorithm performs $\mathrm{poly}(d, n, r)$ arithmetic operations on $\mathrm{poly}(d, n, r, b, \log(1/\epsilon))$-bit numbers.*

This theorem will allow us to avoid having the matrices A in our main algorithm blow up in bit complexity since every round we can replace A by A' to reduce the bit complexity with at most a small loss of potential. See the full version for the details.

6 PAC LEARNING HALFSPACES IN STRONGLY POLYNOMIAL TIME

In this section, we give our strongly polynomial improper PAC learner for halfspaces, thereby establishing Theorem 1.6.

6.1 Approximate Forster Decomposition

Theorem 1.5 is often difficult to use directly as it does not always guarantee a Forster ransform. This is necessary because if many points are concentrated on a subspace, it may be the case that no such transform exists. However, in this case we can at least find a dense subspace and hopefully can find a Forster transform on that subspace. In general, we have the following result:

Proposition 6.1 (Forster Decomposition). *There is an algorithm that given a multiset X of n points in $\mathbb{R}^d_*$ and $\epsilon > 0$, runs in time strongly-polynomial in $d\, n/\epsilon$, and with high probability returns a subspace $V \subseteq \mathbb{R}^d$ with $V \neq \mathbf{0}$ and a linear transformation $A : V \to \mathbb{R}^{\dim(V)}$, such that*

(1) $|X \cap V| \ge (n/d)\ \dim(V)$.
(2) The eigenvalues of $\frac{1}{|X\cap V|}\sum_{x\in X\cap V} f_A(x)(f_A(x))^\top$ are in $[(1-\epsilon)/\dim(V), (1+\epsilon)/\dim(V)]$.

6.2 PAC Learning Halfspaces

Since we work in the distribution-independent setting, will assume without loss of generality that the target halfspace is homogeneous, i.e., has zero threshold. We can straightforwardly reduce the general case to the homogeneous case by increasing the dimension by 1. In particular, if we associate point $x \in \mathbb{R}^d$ with $x' = (x, -1) \in \mathbb{R}^{d+1}_*$, then we note that $w \cdot x - t = (w, t) \cdot (x, -1)$, and thus a general halfspace over the x vectors is equivalent to a homogeneous halfspace over the x'.

The basic idea of our PAC learning algorithm is that if we are given a set of points in approximate radial isotropic position, we can use a variant of the perceptron algorithm to efficiently compute a hypothesis that correctly classifies a reasonable fraction of these points. In particular, we will be using the following lemma, a version of which appears in [6, 20]:

Lemma 6.2. *Let S be a set of n labeled examples $(x, y) \in \mathbb{R}^d \times \{\pm 1\}$ such that there exists an unknown vector $w \in \mathbb{R}^d_*$ with $y = \mathrm{sign}(w \cdot x)$ for each $(x, y) \in S$, and let $\gamma > 0$ be a parameter. There exists an algorithm that given S and γ has running time strongly polynomial in $n\, d/\gamma$, and returns a vector $v \in \mathbb{R}^d_*$ that for all $(x, y) \in S$ with $|v \cdot x| \ge \gamma \|v\|_2 \|x\|_2$ satisfies $y = \mathrm{sign}(v \cdot x)$.*

Combining the modified perceptron algorithm of Lemma 6.2 with an approximate Forster transform, gives us a way to learn a reasonable fraction of the points for any linearly separable dataset.

Lemma 6.3. *Let S be a multiset of labeled examples $(x, y) \in \mathbb{R}^d_* \times \{\pm 1\}$ such that there exists an unknown vector $w \in \mathbb{R}^d_*$ with $y = \mathrm{sign}(w \cdot x)$ for each $(x, y) \in S$. There exists a strongly polynomial time algorithm that with high probability returns a subspace V of $\mathbb{R}^d$, a linear transformation $A : V \to \mathbb{R}^{\dim(V)}$, and a vector $v \in V$ such that for every $(x, y) \in S$ with $x \in V$ and $|v \cdot (Ax)| \ge \|v\|_2 \|Ax\|_2/(2\sqrt{d})$ we have that $y = \mathrm{sign}(v \cdot x)$. Furthermore, this holds for at least a $1/(4d)$-fraction of points $(x, y) \in S$.*

Ideally, we would like a version of Lemma 6.3 that works over a distribution rather than a finite set. This can be achieved by running the algorithm of Lemma 6.3 on a suitably large set of samples. To establish generalization guarantees, we leverage the fact that the collection of possible classifiers comes from a set of bounded VC-dimension.

Proposition 6.4. *Let $\mathcal{D}$ be a distribution over $\mathbb{R}^d \times \{\pm 1\}$ such that for some unknown vector $w \in \mathbb{R}^d_*$ we have that for $(x, y) \sim \mathcal{D}$ that $y = \mathrm{sign}(w \cdot x)$ almost surely. Given $\epsilon, \delta > 0$ with $\epsilon < 1/(20d)$, there exists an algorithm that draws $n = O(d^2 \log(1/\delta)/\epsilon^2)$ i.i.d. samples from $\mathcal{D}$, runs in time strongly polynomial in n, d, and with probability at least $1 - \delta$ returns a vector subspace V in $\mathbb{R}^d$, a linear transformation $A : V \to \mathbb{R}^{\dim(V)}$ and a vector $v \in V$, such that:*

(1) The probability over $(x, y) \sim \mathcal{D}$ that $x \in V$, $|v \cdot (Ax)| \ge \|v\|_2 \|Ax\|_2/(2\sqrt{d})$, and $y \neq \mathrm{sign}(v \cdot x)$ is at most ϵ.
(2) The probability over $(x, y) \sim \mathcal{D}$ that $x \in V$ and $|v \cdot (Ax)| \ge \|v\|_2 \|Ax\|_2/(2\sqrt{d})$ is at least $1/(5d)$.

We are now ready to prove the main result of this section.

Theorem 6.5. *Let $\mathcal{D}$ be a distribution over $\mathbb{R}^d \times \{\pm 1\}$ such that for some unknown vector $w \in \mathbb{R}^d_*$ we have that for $(x, y) \sim \mathcal{D}$ that $y = \mathrm{sign}(w \cdot x)$ almost surely. Given $\epsilon, \delta > 0$ with $\epsilon < 1/(20d)$ there is an algorithm that draws $n = O(d^{9/2} \log(1/\epsilon) \log(d/\epsilon\delta)/\epsilon^2)$ i.i.d. samples from $\mathcal{D}$, runs in strongly polynomial time, and returns a strongly polynomial time computable function $f : \mathbb{R}^d \to \{\pm 1\}$ such that with probability $1 - \delta$ over the samples it holds that $\mathbf{Pr}_{(x,y)\sim\mathcal{D}}[f(x) \neq y] \le \epsilon$.*

7 CONCLUSIONS AND OPEN PROBLEMS

In this work, we designed the first strongly polynomial time algorithm for computing ϵ-approximate Forster transforms of

Algorithm 4 Halfspace Learning Algorithm

function LearnLTF (sample access to distribution $\mathcal{D}$ over $\mathbb{R}^d_* \times \{\pm 1\}$, accuracy parameter ϵ)
  Let $f_0 \equiv 0$.
  Let $C > 0$ be a sufficiently large universal constant.
  Let $r = C\sqrt{d}\log(1/\epsilon) \in \mathbb{Z}_+$.
  for $i = 1$ to r **do**
    Take $M := Cd^4 \log(d/\epsilon\delta)/\epsilon^2$ samples from $\mathcal{D}$ and call the resulting multiset T.
    Let S be the set of $(x, y) \in T$ such that $f_{i-1}(x) = 0$.
    if $|S| < \epsilon M/4$ **then**
      return f_{i-1}
    else
      Run the algorithm from Proposition 6.4 with parameters $\epsilon \leftarrow \epsilon/(10d)$ and $\delta \leftarrow \delta/(2r)$ to obtain V, A, v, using S as the set of samples.
      Let

$$f_i(x) := \begin{cases} f_{i-1}(x) & \text{, if } f_{i-1}(x) \neq 0 \\ \operatorname{sign}(v \cdot (Ax)) & \text{, if } f_{i-1} = 0, x \in V, \text{and } |v \cdot (Ax)|/(2\sqrt{d}) \\ 0 & \text{, otherwise.} \end{cases}$$

a given dataset[2]. By using this algorithm is an essential ingredient, we gave the first strongly polynomial time algorithm for distribution-free PAC learning of halfspaces, both in the realizable setting and in the presence of semi-random label noise. This algorithmic result is surprising (even in the realizable case), as obtaining a strongly polynomial *proper* PAC learner is *equivalent* to strongly polynomial LP — a major unsolved problem in TCS.

A number of open problems suggest themselves:

- Our ϵ-approximate Forster transform algorithm has runtime scaling polynomially with $1/\epsilon$. That is, our algorithm runs in strongly polynomial time when ϵ is at least inverse polynomial in n, d. An obvious open question is to develop a strongly polynomial algorithm with a polylog$(1/\epsilon)$ runtime dependence. To achieve such a guarantee with our approach, one needs to circumvent two obstacles: First, one would need to reduce the number of iterations of our algorithm (that is controlled by the progress in our potential function). Second, one would require a strongly polynomial approximate eigendecomposition subroutine with a polylog$(1/\epsilon)$ runtime dependence.
- We believe that the following question is of independent interest: Is there a strongly polynomial time algorithm for approximate eigendecomposition with a polylog$(1/\epsilon)$ runtime dependence? Moreover, is there a deterministic algorithm?
- The running time of our algorithm is strongly polynomial in n, d, but the polynomial dependence is quite large (of the order of $(nd)^{10}$). While we did not make any effort to optimize the degree of the polynomials, it would be interesting to understand the quantitative limitations of our approach. Can our approach lead to algorithms with good practical performance?
- As mentioned in the introduction of this paper, Forster's rescaling can be viewed as a very special cases of operator scaling and tensor scaling [24]. These tasks have attracted significant attention in recent years from various communities, and efficient (weakly polynomial) algorithms (in some cases with a poly$(1/\epsilon)$ dependence) have been developed, see, e.g., [1, 9] and references therein. It would be interesting to explore whether our approach can be extended to yield strongly polynomial algorithms (when ϵ is not too small) for such generalizations.

[2]While our Forster algorithm is randomized, we remark that the only source of randomness is due to the method we use to compute an approximate eigendecomposition. It is plausible that deterministic algorithms exist for this purpose, in which case our Forster algorithm becomes deterministic as well.

ACKNOWLEDGEMENTS

I.D. was supported by by NSF Medium Award CCF-2107079, NSF Award CCF-1652862 (CAREER), a Sloan Research Fellowship, and a DARPA Learning with Less Labels (LwLL) grant. C.T. was supported by NSF Award CCF-2008006 and NSF Award CCF-2144298 (CAREER). D.K. was supported by NSF Medium Award CCF-2107547, NSF Award CCF-1553288 (CAREER), and a grant from CasperLabs.

We thank Ravi Kannan, Santosh Vempala, and Mihalis Yannakakis for encouragement and insightful conversations about this work. We are grateful to Daniel Dadush for sharing his expertise on optimization, and for detailed feedback that improved the presentation of this paper. We are indebted to Nikhil Srivastava for answering our questions about the complexity of eigenvalue decomposition, and for technical correspondence regarding our strongly polynomial eigendecomposition routine.

REFERENCES

[1] Z. Allen-Zhu, A. Garg, Y. Li, R. M. de Oliveira, and A. Wigderson. 2018. Operator scaling via geodesically convex optimization, invariant theory and polynomial identity testing. In *Proceedings of the 50th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2018*. ACM, 172–181.
[2] D. Angluin and P. Laird. 1988. Learning From Noisy Examples. *Mach. Learn.* 2, 4 (1988), 343–370.
[3] S. Artstein-Avidan, H. Kaplan, and M. Sharir. 2020. On Radial Isotropic Position: Theory and Algorithms. *CoRR* abs/2005.04918 (2020). arXiv:2005.04918 https://arxiv.org/abs/2005.04918
[4] F. Barthe. 1998. On a reverse form of the Brascamp-Lieb inequality. *Inventiones mathematicae* 134 (1998), 335–361.
[5] U. Betke. 2004. New Combinatorial and Polynomial Algorithms for the Linear Feasibility Problem. *Discrete & Computational Geometry* 32 (2004), 317–338.
[6] A. Blum, A. Frieze, R. Kannan, and S. Vempala. 1997. A Polynomial Time Algorithm for Learning Noisy Linear Threshold Functions. *Algorithmica* 22, 1/2 (1997), 35–52.
[7] A. Blum, A. M. Frieze, R. Kannan, and S. Vempala. 1996. A Polynomial-Time Algorithm for Learning Noisy Linear Threshold Functions. In *37th Annual Symposium on Foundations of Computer Science, FOCS '96*. 330–338.
[8] A. Blumer, A. Ehrenfeucht, D. Haussler, and M. K. Warmuth. 1989. Learnability and the Vapnik-Chervonenkis dimension. *J. ACM* 36, 84 (Oct. 1989), 929–965.
[9] P. Bürgisser, C. Franks, A. Garg, R. M. de Oliveira, Michael Walter, and A. Wigderson. 2018. Efficient Algorithms for Tensor Scaling, Quantum Marginals, and Moment Polytopes. In *59th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2018*. IEEE Computer Society, 883–897.
[10] S. Chen, F. Koehler, A. Moitra, and M. Yau. 2020. Classification Under Misspecification: Halfspaces, Generalized Linear Models, and Evolvability. In *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020*.
[11] E. Cohen. 1997. Learning noisy perceptrons by a perceptron in polynomial time. In *Proceedings of the Thirty-Eighth Symposium on Foundations of Computer Science*. 514–521.

[12] D. Dadush, S. Huiberts, B. Natura, and L. A. Végh. 2020. A scaling-invariant algorithm for linear programming whose running time depends only on the constraint matrix. In *Proccedings of the 52nd Annual ACM SIGACT Symposium on Theory of Computing, STOC 2020*. ACM, 761–774.

[13] D. Dadush, B. Natura, and L. A. Végh. 2020. Revisiting Tardos's Framework for Linear Programming: Faster Exact Solutions using Approximate Solvers. In *61st IEEE Annual Symposium on Foundations of Computer Science, FOCS 2020*. IEEE, 931–942.

[14] I. Diakonikolas, T. Gouleakis, and C. Tzamos. 2019. Distribution-Independent PAC Learning of Halfspaces with Massart Noise. In *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019*, Hanna M. Wallach, Hugo Larochelle, Alina Beygelzimer, Florence d'Alché-Buc, Emily B. Fox, and Roman Garnett (Eds.). 4751–4762.

[15] I. Diakonikolas and D. M. Kane. 2020. Near-Optimal Statistical Query Hardness of Learning Halfspaces with Massart Noise. *CoRR* abs/2012.09720 (2020). arXiv:2012.09720 https://arxiv.org/abs/2012.09720 Conference version in COLT'22..

[16] I. Diakonikolas, D. M. Kane, P. Manurangsi, and L. Ren. 2022. Cryptographic Hardness of Learning Halfspaces with Massart Noise. *CoRR* abs/2207.14266 (2022). https://doi.org/10.48550/arXiv.2207.14266 arXiv:2207.14266 Conference version in NeurIPS'22..

[17] I. Diakonikolas, D. M. Kane, and C. Tzamos. 2021. Forster Decomposition and Learning Halfspaces with Noise. *CoRR* abs/2107.05582 (2021). arXiv:2107.05582 https://arxiv.org/abs/2107.05582 Conference version appeared in NeurIPS'21..

[18] I. Diakonikolas, J. Park, and C. Tzamos. 2021. ReLU Regression with Massart Noise. *CoRR* abs/2109.04623 (2021). arXiv:2109.04623 https://arxiv.org/abs/2109.04623 Conference version appeared in NeurIPS'21..

[19] J. Dunagan and S. Vempala. 2004. Optimal outlier removal in high-dimensional spaces. *J. Computer & System Sciences* 68, 2 (2004), 335–373.

[20] J. Dunagan and S. Vempala. 2004. A simple polynomial-time rescaling algorithm for solving linear programs. In *Proceedings of the 36th Annual ACM Symposium on Theory of Computing*. 315–320.

[21] Z. Dvir, S. Saraf, and A. Wigderson. 2017. Superquadratic Lower Bound for 3-Query Locally Correctable Codes over the Reals. *Theory Comput.* 13, 1 (2017), 1–36.

[22] J. Forster. 2002. A linear lower bound on the unbounded error probabilistic communication complexity. *J. Comput. Syst. Sci.* 65, 4 (2002), 612–625.

[23] Y. Freund and R. Schapire. 1997. A decision-theoretic generalization of on-line learning and an application to boosting. *J. Comput. System Sci.* 55, 1 (1997), 119–139.

[24] A. Garg and R. M. de Oliveira. 2018. Recent progress on scaling algorithms and applications. *Bull. EATCS* 125 (2018). http://eatcs.org/beatcs/index.php/beatcs/article/view/533

[25] A. Garg, L. Gurvits, R. M. de Oliveira, and A. Wigderson. 2017. Algorithmic and optimization aspects of Brascamp-Lieb inequalities, via operator scaling. In *Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2017*. ACM, 397–409.

[26] J. Garg and L. A. Végh. 2019. A strongly polynomial algorithm for linear exchange markets. In *Proceedings of the 51st Annual ACM SIGACT Symposium on Theory of Computing, STOC 2019*. ACM, 54–65.

[27] A. V. Goldberg and R. E. Tarjan. 1989. Finding minimum-cost circulations by canceling negative cycles. *J. ACM* 36, 4 (1989), 873–886.

[28] M. Goldmann, J. Håstad, and A. Razborov. 1992. Majority gates vs. general weighted threshold gates. *Computational Complexity* 2 (1992), 277–300.

[29] M. Grötschel, L. Lovász, and A. Schrijver. 1988. *Geometric Algorithms and Combinatorial Optimization*. Vol. 2. Springer.

[30] L. Gurvits and A. Samorodnitsky. 2002. A Deterministic Algorithm for Approximating the Mixed Discriminant and Mixed Volume, and a Combinatorial Corollary. *Discrete & Computational Geometry* 27 (2002), 531–550.

[31] L. Hamilton and A. Moitra. 2019. The Paulsen Problem Made Simple. In *10th Innovations in Theoretical Computer Science Conference, ITCS 2019, January 10-12, 2019 (LIPIcs, Vol. 124)*. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 41:1–41:6.

[32] M. Hardt and A. Moitra. 2013. Algorithms and Hardness for Robust Subspace Recovery. In *COLT 2013*. 354–375.

[33] M. Hopkins, D. Kane, S. Lovett, and G. Mahajan. 2020. Point Location and Active Learning: Learning Halfspaces Almost Optimally. In *61st IEEE Annual Symposium on Foundations of Computer Science, FOCS 2020*. IEEE, 1034–1044.

[34] M. Kearns and U. Vazirani. 1994. *An Introduction to Computational Learning Theory*. MIT Press, Cambridge, MA.

[35] T. C. Kwok, L. C. Lau, Y. T. Lee, and A. Ramachandran. 2018. The Paulsen problem, continuous operator scaling, and smoothed analysis. In *Proceedings of the 50th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2018*. ACM, 182–189.

[36] N. Linial, A. Samorodnitsky, and A. Wigderson. 2000. A Deterministic Strongly Polynomial Algorithm for Matrix Scaling and Approximate Permanents. *Comb.* 20, 4 (2000), 545–568. Conference version in STOC'98..

[37] W. Maass and G. Turan. 1994. How fast can a threshold gate learn?. In *Computational Learning Theory and Natural Learning Systems*, S. Hanson, G. Drastal, and R. Rivest (Eds.). MIT Press, 381–414.

[38] P. Massart and E. Nedelec. 2006. Risk bounds for statistical learning. *Ann. Statist.* 34, 5 (10 2006), 2326–2366.

[39] N. Megiddo. 1983. Towards a Genuinely Polynomial Algorithm for Linear Programming. *SIAM J. Comput.* 12, 2 (1983), 347–353.

[40] M. Minsky and S. Papert. 1968. *Perceptrons: an introduction to computational geometry*. MIT Press, Cambridge, MA.

[41] R. Nasser and S. Tiegel. 2022. Optimal SQ Lower Bounds for Learning Halfspaces with Massart Noise. *CoRR* abs/2201.09818 (2022). arXiv:2201.09818 https://arxiv.org/abs/2201.09818 Conference version in COLT'22..

[42] A. Novikoff. 1962. On convergence proofs on perceptrons. In *Proceedings of the Symposium on Mathematical Theory of Automata*, Vol. XII. 615–622.

[43] R. O'Donnell. 2014. *Analysis of Boolean Functions*. Cambridge University Press. http://www.cambridge.org/de/academic/subjects/computer-science/algorithmics-complexity-computer-algebra-and-computational-g/analysis-boolean-functions

[44] N. Olver and L. A. Végh. 2017. A simpler and faster strongly polynomial algorithm for generalized flow maximization. In *Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2017*. ACM, 100–111.

[45] N. Olver and L. A. Végh. 2020. A Simpler and Faster Strongly Polynomial Algorithm for Generalized Flow Maximization. *J. ACM* 67, 2 (2020), 10:1–10:26.

[46] J. B. Orlin. 1993. A Faster Strongly Polynomial Minimum Cost Flow Algorithm. *Operations Research* 41, 2 (1993), 338–350.

[47] B. N. Parlett. 1998. *The Symmetric Eigenvalue Problem*. Society for Industrial and Applied Mathematics, Philadelphia.

[48] F. Rosenblatt. 1958. The Perceptron: a probabilistic model for information storage and organization in the brain. *Psychological Review* 65 (1958), 386–407.

[49] J. Shawe-Taylor and N. Cristianini. 2000. *An introduction to support vector machines*. Cambridge University Press.

[50] S. Smale. 1998. Mathematical problems for the next century. *The Mathematical Intelligencer* 20 (1998), 7–15.

[51] E. Tardos. 1985. A strongly polynomial minimum cost circulation algorithm. *Comb.* 5, 3 (1985), 247–256.

[52] E. Tardos. 1986. A Strongly Polynomial Algorithm to Solve Combinatorial Linear Programs. *Operations Research* 34, 2 (1986), 250–256.

[53] L. G. Valiant. 1984. A theory of the learnable. In *Proc. 16th Annual ACM Symposium on Theory of Computing (STOC)*. ACM Press, 436–445.

[54] V. Vapnik. 1998. *Statistical Learning Theory*. Wiley-Interscience, New York.

[55] S. A. Vavasis and Y. Ye. 1996. A primal-dual interior point method whose running time depends only on the constraint matrix. *Math. Program.* 74 (1996), 79–120.

[56] L. A. Végh. 2014. A strongly polynomial algorithm for generalized flow maximization. In *Symposium on Theory of Computing, STOC 2014*. ACM, 644–653.

[57] L. A. Végh. 2016. A Strongly Polynomial Algorithm for a Class of Minimum-Cost Flow Problems with Separable Convex Objectives. *SIAM J. Comput.* 45, 5 (2016), 1729–1761.

[58] A. Yao. 1990. On ACC and threshold circuits. In *Proceedings of the Thirty-First Annual Symposium on Foundations of Computer Science*. 619–627.

Received 2022-11-07; accepted 2023-02-06