Finding Correlations in Subquadratic Time, with Applications to Learning Parities and the Closest Pair Problem *

Gregory Valiant
Stanford University
gregory.valiant@gmail.com

August 29, 2013

Abstract

Given a set of n d-dimensional Boolean vectors with the promise that the vectors are chosen uniformly at random with the exception of two vectors that have Pearson–correlation ρ (Hamming distance $d \cdot \frac{1-\rho}{2}$), how quickly can one find the two correlated vectors? We present an algorithm which, for any constant $\epsilon > 0$, and constant $\rho > 0$, runs in expected time $O\left(n^{\frac{5-\omega}{4-\omega}+\epsilon} + nd\right) < O(n^{1.62} + nd)$, where $\omega < 2.4$ is the exponent of matrix multiplication. This is the first subquadratic–time algorithm for this problem for which ρ does not appear in the exponent of n, and improves upon $O(n^{2-O(\rho)})$, given by Paturi et al. [29], the Locality Sensitive Hashing approach of [18] and the Bucketing Codes approach of [13].

Applications and extensions of this basic algorithm yield significantly improved algorithms for several other problems:

Approximate Closest Pair: For any sufficiently small constant $\epsilon > 0$, given n d-dimensional vectors, our algorithm returns a pair of vectors whose Euclidean (or Hamming) distance differs from that of the closest pair by a factor of at most $1+\epsilon$, and runs in time $O(n^{2-\Theta(\sqrt{\epsilon})})$. The best previous algorithms (including Locality Sensitive Hashing) have runtime $O(n^{2-O(\epsilon)})$.

Learning Sparse Parities with Noise: Given samples from an instance of the learning parities with noise problem where each example has length n, the true parity set has size at most $k << n$, and the noise rate is η, our algorithm identifies the set of k indices in time $n^{\frac{\omega+\epsilon}{3}k} poly(\frac{1}{1-2\eta}) < n^{0.8k} poly(\frac{1}{1-2\eta})$. This is the first algorithm with no dependence on η in the exponent of n, aside from the trivial $O\left(\binom{n}{k}\right) \approx O(n^k)$ brute-force algorithm, and for large noise rates ($\eta > 0.4$), improves upon the results of Grigorescu et al. [15] that give a runtime of $n^{(1+(2\eta)^2+o(1))\frac{k}{2}} poly(\frac{1}{1-2\eta})$.

Learning k-Juntas with Noise: Given uniformly random length n Boolean vectors, together with a label, which is some function of just $k << n$ of the bits, perturbed by noise rate η, return the set of relevant indices. Leveraging the reduction of Feldman et al. [14] our result for learning k-parities implies an algorithm for this problem with runtime $n^{\frac{\omega+\epsilon}{3}k} poly(2^k, \frac{1}{1-2\eta}) < n^{0.8k} poly(2^k, \frac{1}{1-2\eta})$, which is the first runtime for this problem of the form n^{ck} with an absolute constant $c < 1$.

Learning k-Juntas without Noise: Given uniformly random length n Boolean vectors, together with a label, which is some function of $k << n$ of the bits, return the set of relevant indices. Using a modification of the algorithm of Mossel et al. [24], and employing our algorithm for learning sparse parities with noise via the reduction of Feldman et al. [14], we obtain an algorithm for this problem with runtime $n^{\frac{\omega+\epsilon}{4}k} poly(2^k, n) < n^{0.6k} poly(2^k, n)$, which improves on the previous best of $n^{\frac{\omega+1}{\omega}k} \approx n^{0.7k} poly(2^k, n)$ of Mossel et al. [24].

*A preliminary version of a portion of this work appeared at the IEEE Symposium on Foundations of Computer Science (FOCS) 2012, as "Finding Correlations in Subquadratic Time, with Applications to Learning Parities and Juntas" [37].

1 Introduction

One of the most basic statistical tasks is the problem of finding correlations in data. In some settings, the data is gathered with the specific goal of ascertaining the set of correlated or anti-correlated variables; in many other settings, the identification of correlated features is used repeatedly as a key data analysis primitive within the context of more complex algorithms. This prompts the basic question: *how quickly can one find correlated features in data?*

With the recent rise of big data, this question has immediate practical relevance. To give one concrete example, in the case of genome-wide association studies (GWAS), it is the computational challenge of efficiently inferring the structural relationships between the millions of genetic markers that is the bottleneck in unlocking information into the genetic factors that predict disease and health risks (see e.g. [22, 40]).

Perhaps the most simple formalization of this problem of finding correlations is the following: Given a set of n d-dimensional Boolean vectors with the promise that the vectors are chosen uniformly at random from the Boolean hypercube, with the exception of two vectors that have Pearson–correlation ρ (Hamming distance $d \cdot \frac{1-\rho}{2}$), how quickly can one find the correlated pair? This problem was, apparently, first posed by Leslie Valiant in 1988 as the light bulb problem [38]. The first positive solution was provided by Paturi et al. [29], who gave an $n^{2-\Theta(\rho)}$ algorithm. Curiously, this basic setting arises as a special case of various problems that are central to theoretical computer science.

Perhaps most obviously, the light bulb problem is a special case of the *approximate closest pair* problem: given a set of vectors, how can one quickly find two vectors with near-minimal Hamming, or Euclidean distance? Surprisingly, previous algorithms obtained comparable runtimes for the light bulb problem and approximate closest pair problem; phrased differently, in contrast to our algorithms, these algorithms do not significantly leverage the randomness in the light bulb problem, and the fact that nearly all the pairwise distances are extremely concentrated near $d/2$.

The light bulb problem is also easily recognized as a special case of learning parity with noise[1]: suppose one is given access to a sequence of examples (x, y), where $x \in \{-1, +1\}^n$ is chosen uniformly at random, and $y \in \{-1, +1\}$ is set so that $y = z \prod_{j \in S} x_i$, for some fixed, though unknown set $S \subset [n]$, where $z \in \{-1, +1\}$ is chosen to be -1 independently for each example with probability $\eta \in [0, 1/2)$. In the case where the *noise rate* $\eta = 0$, the problem of recovering the set S is easy: given n such examples, with high probability one can recover the set S by Gaussian elimination—translating this problem into a problem over $\mathbf{F}_2$, S is given simply as the solution to a set of linear equations. From an information theory standpoint, the addition of some nonzero amount of noise ($\eta > 0$) does not change the problem significantly; for constant η, given $O(n)$ examples, with high probability there will only be one set $S \subset [n]$ where the parities of the corresponding set of indices of the examples are significantly correlated with the labels. From a computational standpoint, the addition of the noise seems to change the problem entirely. In contrast to the simple polynomial-time algorithm for the noise-free case, when given a small constant amount of noise, the best known algorithm, due to Blum et al. [8] takes time $2^{O(\frac{n}{\log n})}$.

This problem of learning parity with noise is, increasingly, understood to be a central problem in learning theory. Additionally, this problem reoccurs in various forms in several areas of theoretical computer science, including coding theory as the problem of decoding random binary linear codes, and in cryptography in the form of the "learning with errors" problem that underlies lattice-based cryptosystems, see e.g. [31, 9].

[1] In particular, the light bulb problem is the problem of learning parity with noise where the true parity has size $k = 2$. To see one direction of the reduction, given a matrix whose rows consist of samples from an instance of parity with noise, if one simply throws out the data points that have an odd label, the columns in the remaining matrix will be uniformly random, except with the two columns corresponding to the indices of the true parity being correlated.

Our results for learning parities apply to the setting in which the true parity set S is much smaller than n, say $k := |S| = O(\log n)$. This problem of learning k-parities is especially relevant to Learning Theory, as was revealed by a series of reductions given in work of Feldman et al. [14], showing that the problem of learning k-parities (under the uniform distribution, with random classification noise) is at least as hard as the problems of learning k-juntas (where the labels are an arbitrary function of k bits), learning 2^k-term DNFs from uniformly random examples, and the variants of these problems in which the noise is adversarial (rather than random). The existence of such a reduction should not be surprising: the problem of learning a parity with noise is the problem of finding a heavy low-degree fourier coefficient, given the promise that one exists; in the case of learning a $k = \log(n)$ sized junta, for example, one knows that there will be at most 2^k significant fourier coefficients. Intuitively, the presence of more heavy low-degree fourier coefficients should, if anything, facilitate the task of finding such a coefficient. For reference, the specific statements of the reductions are given in Appendix A.

All of our results rely on fast matrix multiplication: our results for the light bulb problem and learning parity with noise use the fact that $n \times n$ matrices may be multiplied in time $O(n^\omega)$, for $\omega < 3$, with the best known bound of $\omega < 2.372$ [42]. Our results for the Closest Pair problem rely on fast rectangular matrix multiplication—namely that for any $\epsilon > 0$, for $\alpha < 0.29$, the product of an $n \times n^\alpha$ and $n^\alpha \times n$ matrix may be computed in time $O(n^{2+\epsilon})$ [11]. While the matrix multiplication does some of the algorithmic heavy lifting, our approach introduces some new metric embedding machinery, in particular the "Chebyshev embedding" for the approximate closest pair problem, which may be of independent interest and utility.

1.1 Related Work

Historically, the first line of work on finding close pairs of vectors focussed on "nearest neighbor search": given a set of vectors how can one preprocess them such that given a new vector, one can efficiently find the vector in the set that is closest of the new vector (with respect to some metric–typically Euclidean distance, or Hamming distance in the Boolean setting)? For such problems, there are typically two parameters of interest: the amount of space required to store the preprocessed set of vectors, and the amount of time required to perform a single query.

The earliest work on this question considered the case in which the n points lie in very low dimensional space, $d = 1, 2, 3, \ldots$. In the case that $d = 1$, each point is a real number, and one can simply sort the list of numbers, and store the sorted list. Given a new number, one can perform a binary search over the sorted list to find the closest number. Thus the storage space is linear in the size of the input, and each query requires $O(\log n)$ comparisons. For $d \geq 2$, the analogous scheme corresponds to partitioning the space into n regions, indexed by the points in the set, where the region corresponding to the ith point x_i consists of those points that are closer to x_i than to any of the other $n - 1$ points in the set.

In the case of $d = 2$, such a partition of the plane is known as the *Voronoi diagram* of the set of n points, and yields space bounds and query time analogous to the $d = 1$ setting. Unfortunately, such schemes suffer a curse of dimensionality and do not generalize well to higher dimensions; while the query time remains polynomial in $d \log n$, the space required to store such partitions scales as $O(n^{\lceil d/2 \rceil})$ [10], and quickly cease to be preferable to performing the brute-force comparisons (see, for example, [23, 41]). On the practical side, there has been considerable work in developing reasonable data structures to partition slightly higher dimensional Euclidean spaces ($d < 20$), starting with the notion of k-dimensional trees (*kd–trees*), introduced by Bentley [6] (see [34] for a summary including more recent developments).

In an effort to overcome some of the difficulties of returning the exact closest point, starting in

the late 1990's, significant effort was spent considering the c–approximate nearest neighbor problem in which the goal of returning the closest point is relaxed to the more modest goal of returning a point whose distance is at most a multiplicative factor of $c = 1 + \epsilon$ larger than that of the closest point. Additionally, a small probability of failure is allowed. In many practical settings, such a relaxation is essentially equivalent to the exact nearest neighbor problem. Starting with the results of Kushilevitz et al. [20] and Indyk and Motwani [18], algorithms requiring space that is polynomial in n and d, with query time polynomial in $d \log n$ were given (for constant $\epsilon > 0$).

Introduced in work of Indyk and Motwani [18], the concept of *locality sensitive hashing* offered both sublinear query time, as well as subquadratic space, thereby yielding nontrivial algorithms for the approximate closest pair problem. Specifically, they gave an algorithm with query time $O(n^{\frac{1}{1+\epsilon}})$ and space $O(n^{1+\frac{1}{1+\epsilon}})$. (Throughout, we ignore $\log n$ factors, and the additive dn term in the space.) The basic idea was to use a series of hashing functions that all have the property that close points have a higher probability of hashing to the same bucket. To perform a given query, one simply hashes the query point, and then checks the subset of the n data points that have also been hashed to those buckets. Since the original paper, there have been a number of improvements in the parameters, and generalizations from Hamming and Euclidean distance to other ℓ_p metrics [12, 28, 3]. The current state of the art for the $1+\epsilon$ nearest neighbor problem under the Euclidean metric is given in Andoni and Indyk [3], achieving query time and space $O(dn^{\alpha}), O(n^{1+\alpha})$, respectively, for $\alpha = \frac{1}{(1+\epsilon)^2} + o(1)$. These results were recently shown to be essentially tight in the sense that for any scheme based on locality sensitive hashing, the exponent $\alpha \geq \frac{1}{(1+\epsilon)^2} - o(1)$ [25, 26]. (See [4] for a survey on locality sensitive hashing.)

For the problem of finding a pair of points whose distance is at most a factor of $1+\epsilon$ further than that of the closest pair of points, by simply running the nearest–neighbor search n times—once for each vector—one obtains algorithms with runtimes $O(n^{1+\frac{1}{1+\epsilon}})$, and $O(n^{1+\frac{1}{(1+\epsilon)^2}})$, respectively in the Hamming and Euclidean settings which are the best previously known algorithms for these problems. For small ϵ, these runtimes are roughly $O(n^{2-\epsilon})$, and $O(n^{2-2\epsilon})$, respectively.

1.1.1 The Light Bulb Problem

The *light bulb problem* is the problem of finding a single planted pair of correlated vectors from among a set of random Boolean vectors:

Definition 1. *Given a set of n vectors in $\{-1, +1\}^d$, with the promise that the vectors are chosen uniformly at random with the exception of two vectors that have Pearson–correlation ρ (Hamming distance $d \cdot \frac{1-\rho}{2}$), the* light bulb problem *with parameters n, d, ρ is the problem of recovering the true correlated pair of vectors.*

The light bulb problem is easily seen to be a special case of the approximate closest pair problem. To see the correspondence in parameters, for sufficiently large dimension, d, the Hamming distance between the correlated vectors will be at most $\frac{1-\rho}{2}d$ whereas, with high probability, the Hamming distance between any other pair of vectors will be close to $\frac{d}{2}$. Thus solving the $1 + \epsilon$ approximate closest pair problem for $1 + \epsilon \approx \frac{1}{1-\rho}$ will, with high probability, return the correlated pair of vectors.

This special setting of the approximate closest pair problem captures much of the challenge of the more general problem in the rigorous sense that the lower bounds for locality sensitive are based on this planted correlation setting [25, 26]. The fact that the light bulb problem is the hardest setting for locality sensitive hashing, together with the fact that this setting has structure that can be leveraged—namely that, with high probability, the inner products between most of the "uninteresting" pairs of vectors will be extremely tightly concentrated around zero—make the light bulb problem a natural starting point for efforts to improve upon locality sensitive hashing.

Despite this, the light bulb problem has received much less attention than the nearest-neighbor problem; the early work on locality sensitive hashing seemed unaware that somewhat similar ideas had appeared nearly a decade earlier in the work of Paturi et al. [29], which gave an algorithm for the light bulb problem with runtime $O(n^{1+\frac{\log \frac{\rho+1}{2}}{\log 1/2}})$, which is slightly better than that given by the application of locality sensitive hashing for the Hamming cube given in [18]. More recently, Dubiner introduced the "Bucketing Codes" approach [13], which is similar in spirit to the approach of Andoni and Indyk [3], and yields an algorithm for the light bulb problem with a runtime of $O(n^{\frac{2}{\rho+1}})$.

For small values of ρ, all these approaches yield runtimes of $n^{2-O(\rho)}$, with [13] achieving the best asymptotic runtime of $O(n^{2-2\rho})$, in the limit as $\rho \to 0$.

For our results on the light bulb problem, and closest pair problem, we will perform some metric embeddings: the hope is to construct some embedding $f : \mathbb{R}^d \to \mathbb{R}^m$ with the property that if $\langle u, v\rangle$ is large, then the inner product of the images of u and v, $\langle f(u), f(v)\rangle$ will also be large, and if the inner product is small, the inner product of the images is tiny. For the light bulb problem, we will be able to choose the embedding f to be a simple "XOR"/"tensor" embedding, which sets each coordinate of $f(u)$ to be the product of entries of u. Such an embedding has appeared previously in several contexts, and was used by Lyubashevsky [21] to show that given few examples from an instance of learning parity with noise, one can generate new "simulated" examples, that can be used in place of actual examples.

Our results for the approximate closest pair problem will require a more sophisticated embedding. In fact, it seems unlikely that we would be able to construct a single embedding f with the desired properties (see Fact 10). Instead of using a single embedding, we will construct a pair of embeddings, $f, g : \mathbb{R}^d \to \mathbb{R}^m$ with the property that $\langle f(u), g(v)\rangle$ is large [small] if $\langle u, v\rangle$ is large [small]. This observation that a pair of embeddings can be more versatile than a single embedding was also fruitfully leveraged by Alon and Naor in their work on approximating the cut norm of a matrix [2].

1.1.2 Parities, Juntas, and DNFs

The problem of learning parity with noise is defined as follows:

Definition 2. *An* example (x, y) *from an* (n, k, η)-instance *of parity with noise, consists of* $x \in \{-1, +1\}^n$*, chosen uniformly at random, together with a* label $y \in \{-1, +1\}$ *defined by* $y = z \cdot \prod_{i \in S} x_i$*, where* $z \in \{-1, +1\}$ *is chosen independently of* x *to be* -1 *with probability* η*, for some fixed set* $S \subset [n]$ *with* $|S| = k$*.*

The problem of learning parity with noise is the task of recovering the set S*, given access to a set of example pairs* (x, y)*.*

In the case that the *noise rate* $\eta = 0$, by translating the entries of the examples from being in $\{-1, 1\}$ to being elements of $\mathbb{F}_2$, this problem of recovering the set S is simply the task of solving a linear system of equations over $\mathbb{F}_2$, since the dot product (over $\mathbb{F}_2$) of each example with the indicator of S will yield the label. Such a linear system can trivially be solved in time $O(n^3)$ via Gaussian elimination, irrespective of $k = |S|$.

In contrast to the setting of solving systems of noisy linear equations over the real numbers, there is no easy least squares regression algorithm over finite fields. For even a small positive noise rate, $\eta > 0$, the complexity of this problem seems to change drastically; algorithms such as Gaussian elimination will no longer work, as they proceed by adding and subtracting examples from other examples, and the noise in the labels of the corrupted examples will thereby be spread throughout the set of examples until there is essentially no signal left in the final output of the algorithm.

It is worth stressing that the difficulty of this problem is strictly computational. From an information theoretic standpoint, the addition of a small amount of noise does not change the problem

significantly—given $O(n)$ examples, Chernoff bounds yield that with overwhelming probability, the true parity set S will be the only set for which the product of the corresponding indices correlates significantly with the labels.

Interest in this problem of learning parity with noise was sparked by the results of Blum et al. [8], who first showed that there exists a class of functions that can be learned in polynomial-time with a constant amount of random classification noise, but which, provably, cannot be efficiently learned in the *statistical query* (SQ) learning model. The SQ learning framework, introduced by Kearns in 1993 [19], sought to abstract and formalize the restricted manner in which many types of learning algorithms interact with data. Specifically, given a distribution over labelled examples, an SQ algorithm interacts with the data via the following protocol: it describes a function, f_1 from an example/label pair to $\{0, 1\}$, and then receives the average value of that function over the examples, with the addition of a small amount of (potentially adversarial) noise. The algorithm then produces a second query, f_2, and receives a perturbed expectation of that function, and so on. This framework captures many learning algorithms: stochastic gradient descent, the perceptron algorithm, etc. The salient feature of all SQ algorithms, is that because they only interact with the data via receiving noisy expectations, they are robust to modest amounts of random classification noise. Intuitively, the main limitation of SQ algorithms is that they can not interact directly with the data, precluding algorithms such as Gaussian elimination which seem to require access to the actual data points.

Blum et al. [8] showed that parity functions on $O(\log n \log \log n)$ bit strings with constant noise rate can be learned in polynomial time, whereas the earlier results of Blum et al. [7] imply that any SQ algorithm provably requires a super-polynomial number of queries (provided the noise rate of each query is at least inverse polynomial). Phrased differently, they presented an algorithm for learning parity with noise over n bit strings, with runtime $2^{O\left(\frac{n}{\log n}\right)}$, whereas any SQ algorithm provably required runtime $2^{\Omega(n)}$.

Their algorithm proceeds by obtaining a huge number of examples, $2^{O\left(\frac{n}{\log n}\right)}$, and then performs a sort of "block" Gaussian elimination in which the vast number of examples is leveraged to ensure that sets of no more than $O(\sqrt{n})$ examples are added together, as opposed to $O(n)$ that would occur in typical Gaussian elimination. This reduction in the number of examples that are added together implies that the level of noise in the output (which increases geometrically with every additional addition of an example), is significantly reduced, allowing for a slightly sub-exponential algorithm.

This algorithm prompted several other works, including work by Lyubashevsky [21], who showed that a similar approach could be applied to a much smaller set of examples $O(n^{1+\epsilon})$ and still obtain a sub-exponential, though slightly larger, runtime of $2^{O\left(\frac{n}{\log \log n}\right)}$. The algorithm of Blum et al. was also shown to have applications to various lattice problems, including the shortest lattice vector [1].

The assumption that the noise in each example's label is determined independently seems crucial for the hardness of learning parity with noise. In the case in which noise is added in a structured manner—for example, if examples arrive in sets of three, with the promise that *exactly* one out of each set of three examples has an incorrect label, the recovery problem can be solved in polynomial time, as was shown by Arora and Ge [5].

More recently, with the surge of attention on lattice problems prompted by the development of lattice-based cryptosystems, there has been much attention on the related problem of *learning with errors* (LWE). The LWE problem, introduced by Regev in 2005 [31], corresponds to the problem of learning parity with noise with two modifications: instead of working over $\mathbb{F}_2$ the LWE is over a larger finite field, and every example is perturbed by adding a small amount of (discrete) Gaussian noise. One of the attractions of basing cryptosystems on the LWE problem is that it has been shown to be as hard as the *worst–case* hardness of lattice problems such as GapSVP (the decision variant of the shortest lattice vector problem), and SIVP (the shortest independent vectors problem) [31, 30].

See [32], for a relatively recent survey on LWE. There are no known such hardness reductions for learning parity with noise.

1.2 Sparse Parities and Juntas

Our results will concern the problem of learning *sparse* parities with noise. Specifically, this is the problem of learning parities with noise in the special case when the size of the parity set $k = |S|$ is known to be very small. Such a restriction clearly makes the problem easier, as one could simply perform a brute-force search over all $\binom{n}{k} \approx n^k$ sets of k indices. In light of the subexponential algorithm of Blum et al. [8] for learning large parities, it is tempting to hope that analogous savings over the brute-force approach can be achieved in the sparse setting, perhaps yielding an $n^{o(k)}$ algorithm, though no such algorithm is known, and adapting the approach of Blum et al. to the sparse setting seems problematic.

This problem of learning sparse parities is especially relevant to learning theory, as several other basic problems in learning theory have been reduced to it. In 2006, Feldman et al. [14], showed that algorithms for learning k-sparse parities with noise can be used to learn k-juntas—functions from $\{0,1\}^n \to \{0,1\}$ which only depend on the values of $k << n$ of the indices (see Definition 5)—and learning 2^k-term DNF, from uniformly random examples.

The reductions of Feldman et al. transform instances of k-juntas or 2^k-term DNF into instances of parity with noise, with a parity of size $\leq k$, by adding some specially designed extra noise, which zeros out nearly all the heavy Fourier coefficients of the juntas or DNF. With some reasonable probability, however, exactly one heavy Fourier coefficient will remain, in which case this process has created an instance of parity with noise. It is worth stressing that such a transformation adds a large amount of noise—corresponding to noise rate $\eta = \frac{1}{2} - \frac{1}{2^k}$, thus motivating the development of algorithms for sparse parity with noise that are very noise robust; for example, algorithms whose runtimes depend only as $poly(\frac{1}{1/2-\eta})$, as opposed to having the noise rate in the exponent of n.

For completeness, in Appendix A we include formal statements of the reductions of Feldman et al. [14], which we use to obtain improved algorithms for learning k-juntas and DNF from our algorithm for learning parities. We now briefly summarize the previous algorithmic work on learning sparse parities and k-juntas.

For the problem of learning k-sparse parities with noise, in a recent paper, Grigorescu et al. [15] adapt the approach of Hopper and Blum [16] to the noisy setting to give an algorithm that runs in time $poly(\frac{1}{1-2\eta})n^{((1+2\eta)^2+o(1))k/2}$. In particular, as the noise rate goes to 0, the performance of this algorithm tends to $O(n^{k/2})$, and as the noise rate tends towards $\frac{1}{2}$, the dependency on n tends towards $O(n^k)$.

For the problem of learning juntas over the uniform distribution, Mossel et al. [24] show that size k juntas can be learned *in the absence of noise*, in time $n^{\frac{\omega k}{\omega+1}}poly(2^k) \approx n^{0.70k}poly(2^k)$. This result leverages a powerful characterization of k-juntas: in particular, they show that any k-junta either has a nonzero Fourier coefficient of degree at most d, or, when regarded as a polynomial over $\mathbb{F}_2$, the k-junta has degree at most $k - d$. Their result follows from balancing a brute-force search for low-degree Fourier coefficients, with solving a large system of linear equation (using fast matrix multiplication) to find the low-degree representation over $\mathbb{F}_2$ in the case that the brute-force search did not find any heavy Fourier coefficients. As this approach involves solving a large system of linear equations, the assumption that there is no noise is necessary. In particular, for constant noise η, prior to our results, no algorithm for learning k-juntas with noise $\eta > 0$ running in time $O(n^{ck})$ for any constant $c < 1$ was previously known.

For the problem of (ϵ, δ) PAC-learning s-term DNF under the uniform distribution, the results

of Grigorescu et al. [15] imply a runtime of

$$poly(\log \frac{1}{\delta}, \frac{1}{\epsilon}, s) n^{(1-\tilde{O}(\epsilon/s)+o(1)) \log \frac{s}{\epsilon}},$$

which improves upon the $O(n^{\log \frac{s}{\epsilon}})$ of Verbeurgt [39] from 1990.

2 Summary of Results and Techniques

We provide a summary of our main theorems, and outline the intuitions and techniques leveraged to obtain each result.

2.1 Main Theorems

We begin by stating our main results for the light bulb problem, and finding approximate closest pairs of points. We then give our results for learning parity with noise, and state its corollaries for learning juntas, both with and without noise, and DNFs.

Theorem 1. *Consider a set of n vectors in $\{-1, 1\}^d$ and constants $\rho, \tau \in [0, 1]$ with $\rho > \tau$ such that for all but at most s pairs u, v of distinct vectors, $|\langle u, v \rangle| \leq \tau d$. There is an algorithm that, with probability $1 - o(1)$, will output* all *pairs of vectors whose inner product is least ρd. Additionally, the runtime of the algorithm is*

$$\tilde{O}\left(sdn^{1/(4-\omega)} + n^{\frac{5-\omega}{4-\omega} + \omega \frac{\log \rho}{\log \tau}}\right) \leq O\left(sdn^{0.7} + n^{1.62 + 3\frac{\log \rho}{\log \tau}}\right),$$

where $\omega < 2.4$ is the exponent of matrix multiplication, and the $\tilde{O}$ notation hides polylogarithmic factors of n.

The above theorem together with basic Chernoff bounds immediately yields the following corollary bounding the complexity of the light bulb problem in the setting in which the dimension d is near the information theoretic limit of $O(\frac{\log n}{\rho^2})$.

Corollary 3. *For any constants $\rho, \epsilon > 0$, there exists a constant c_ϵ dependent on ϵ such that for $d \geq c_\epsilon \frac{\log n}{\rho^2}$, there is an algorithm that, with probability $1 - o(1)$, will find a planted set of ρ-correlated vectors from among n random vectors in $\{\pm 1\}^d$, and will run in time*

$$\tilde{O}(n^{\frac{5-\omega}{4-\omega} + \epsilon}) \leq O(n^{1.62}),$$

were $\omega > 2.4$ is the exponent of matrix multiplication.

We stress that the runtimes given in the above results differs in nature from those given by previous approaches in that the dependence on the correlation, ρ, has been removed from the exponent of n, yielding significant improvements in the asymptotic performance for small values of ρ.

Theorem 2. *Given n vectors in $\mathbb{R}^d$ and sufficiently small approximation parameter $\epsilon > 0$, with probability $1 - o(1)$ our algorithm will return a pair of vectors u, v such $||u - v|| \leq (1 + \epsilon)d^*$, where d^* is the (Euclidean) distance between the closest pair of vectors. Additionally, the algorithm runs in time*

$$O\left(n^{2-\Omega(\sqrt{\epsilon})} + nd\right).$$

2.1.1 Parities, Juntas, and DNFs

Definition 4. *An* example (x, y) *from an* (n, k, η)-instance *of parity with noise, consists of* $x \in \{-1, +1\}^n$*, chosen uniformly at random, together with a* label $y \in \{-1, +1\}$ *defined by* $y = z \cdot \prod_{i \in S} x_i$*, where* $z \in \{-1, +1\}$ *is chosen independently of* x *to be* -1 *with probability* η*, for some fixed set* $S \subset [n]$ *with* $|S| = k$*.*

Corollary 3 can be leveraged to yield an algorithm for an (n, k, η) instance of parity with noise with runtime $n^{k\frac{5-\omega+\epsilon}{8-2\omega}} poly(\frac{1}{1-2\eta}) \approx n^{0.81} poly(\frac{1}{1-2\eta})$. We are able to very slightly improve this exponent via a related, though alternate approach to obtain the following:

Theorem 3. *For any fixed* $\epsilon > 0$*, for sufficiently large* n *and* k*, given examples from an* (n, k, η) *instance of parity with noise, with probability* $1 - o(1)$*, our algorithm will correctly return the true set of* k *parity bits. Additionally, the algorithm will run in time*

$$n^{\frac{\omega+\epsilon}{3}k} poly(\frac{1}{1-2\eta}) < n^{0.80k} poly(\frac{1}{1-2\eta}).$$

The above theorem has immediate implications for the problems of learning juntas and DNFs, via the reductions of Feldman et al. [14], which are summarized in Theorems A.1 and A.2 and Corollary A.3 in the Appendix:

Definition 5. *An* example (x, y) *from a* (n, η)-instance *of a noisy* k*-junta consists of* $x \in \{-1, +1\}^n$*, chosen uniformly at random, together with a* label $y \in \{-1, +1\}$ *defined by* $y = z \cdot f(x_S)$*, where* $z \in \{-1, +1\}$ *is chosen independently of* x *to be* -1 *with probability* η*,* f *is a fixed though unknown function* $f : \{-1, +1\}^k \to \{-1, +1\}$*, and* x_S *denotes the indices of* x *occurring in a fixed (though unknown) set* $S \subset [n]$ *with* $|S| = k$*.*

The above theorem together with Theorem A.1 immediately implies the following corollary:

Corollary 6. *For sufficiently large* n *and* k *given access to examples from an* (n, η) *instance of a noisy* k*-junta, with constant probability our algorithm will correctly return the true set of* $k' \leq k$ *relevant indices, and truth table for the function. Additionally, the algorithm has runtime, and sample complexity bounded by*

$$n^{\frac{\omega+\epsilon}{3}k} poly(\frac{1}{1-2\eta}) < n^{0.80k} poly(2^k, \frac{1}{1-2\eta}).$$

For the problem of learning juntas *without* noise, one can further improve the exponent by combining our improved algorithm for parity with noise, with the approach of Mossel et al. [24]. Theorem 3 together with Corollary A.3 yields the following corollary for learning juntas without noise, where the exponent is obtained by setting $\alpha = \frac{3}{4}$ in the statement of Corollary A.3 so as to equate the two arguments of the max operation:[2]

Corollary 7. *For sufficiently large* n *and* k *given access to examples from an* (n, η) *instance of a noisy* k*-junta with* $\eta = 0$*, with constant probability our algorithm will correctly return the true set of* $k' \leq k$ *relevant indices, and truth table for the function. Additionally, the algorithm has runtime, and sample complexity bounded by*

$$n^{\frac{\omega+\epsilon}{4}k} poly(2^k, n) < n^{0.60k} poly(2^k, n).$$

[2] I am grateful to Vitaly Feldman for drawing my attention to this corollary—that an improved algorithm for learning small parities that is very noise-robust yields an improved algorithm for learning k-juntas *without* noise.

Definition 8. *An* example (x, y) *from a r-term DNF over n bits under the uniform distribution consists of $x \in \{-1, +1\}^n$, chosen uniformly at random, together with a* label $y \in \{-1, +1\}$ *given by a fixed (though unknown) r-term DNF applied to x.*

The following corollary follows from first arguing that an analog of Theorem 3 holds (Theorem 6) in which the sample complexity has been reduced, and then applying Theorem A.2.

Corollary 9. *For sufficiently large n and k, there exists an algorithm that (ϵ, δ)–PAC learns r-term DNF formulae over n bits from uniformly random examples that runs in time*

$$poly\left(\frac{1}{\delta}, \frac{r}{\epsilon}\right) n^{0.80 \log \frac{r}{\epsilon}},$$

where the logarithm is to the base 2*.*

2.2 Technique Overview

In this section we provide a high-level overview of the main techniques. Throughout, we favor clarity over optimization of the constants. For the light-bulb and closest pair problems, because we are concerned with improving the dependency on the error parameter ϵ *beyond constant factors*, there is little distinction between the Hamming setting and Euclidean setting (and, for example, one can easily convert Euclidean unit vectors into ± 1 vectors by simply choosing a random rotation of the unit sphere, then truncating coordinates according to their signs). Thus the following techniques will apply both to the Hamming and Euclidean settings, and we do not belabor the distinction in this overview.

Light Bulb Problem: Section 3 begins by describing an improved algorithm for the light bulb problem that applies to the setting where the dimension of the vectors is rather large, $d \approx n^{2/3}$, and, in particular, is much larger than the dimension required to information theoretically recover the true planted correlated pair. This improved algorithm leverages the following two basic observations about this high-dimensional light bulb setting, neither of which can be exploited within the locality sensitive hashing framework:

- With the exception of the correlated pair of vectors, given a large amount of data (high dimensional vectors) the inner product between pairs of vectors is extremely tightly concentrated around a single value.
- One need not iteratively solve n nearest neighbor search problems, one for each vector; instead, one can perform many such searches *simultaneously.*

The XOR/Tensor Embedding: Given the ability to efficiently solve the light bulb problem in the setting in which the amount of data is rather large, the natural question is whether one can reduce the low-dimensional setting, in which the inner products between typical vectors are less tightly concentrated, to the high-dimensional setting in which the "uninteresting" inner products have significant concentration. Specifically, given a set of vectors just over the information theoretic dimension, we are hoping to project them into a significantly higher dimensional space in such a way that the inner product between the planted pair remains relatively large, but the pairwise inner products between all the other vectors become tightly concentrated (about zero). It is worth stressing that this embedding into a higher dimensional space is, in some sense, the antithetical approach to locality sensitive hashing in which one projects into a lower-dimensional space then buckets the resulting projections.

Formally, we are searching for some metric embedding $f : \mathbb{R}^d \to \mathbb{R}^m$, for some $m > d$, such that for all unit vectors $u, v \in \mathbb{R}^d$, if $\langle u, v\rangle$ is large, then $\langle f(u), f(v)\rangle$ is relatively large, yet if $\langle u, v\rangle$ is small, then $\langle f(u), f(v)\rangle$ is as small as possible. The "XOR/tensor" embedding (scaled so as to map unit vectors to unit vectors), in which each vector is replaced by its degree q tensor power, is one such embedding. Letting f be such an embedding, we have $\langle f(u), f(v)\rangle = (\langle u, v\rangle)^q$, and hence the multiplicative gap between "big" inner products and "small" inner products is amplified by an exponent of q. While the dimensionality of the image $m = d^q$, one can implicitly subsample indices of the projections so as to ensure that the dimensionality of the vectors remains $<< n$. Our main result for the light bulb problems is obtained by selecting $q \approx \log n$ to be as large as possible, up until the magnitude of the noise in the inner products introduced by the subsampling begins to swamp the discrepancy between the image of the "big" inner product and the "small" inner products. We note that such an embedding has been used in a variety of other settings, including by Lyubashevsky [21] to show that given few examples from an instance of learning parity with noise, one can generate new "simulated" examples, in much the same way as we are creating new dimensions of data in the light bulb setting. The use of such embeddings in the nearest-neighbor setting, or closest-pair setting is novel.

Two Embeddings are Better than One—The Chebyshev Embedding: If one tries to apply the above approach to obtain an algorithm for finding a pair of unit vectors whose inner product is within an additive ϵ from that of the pair with maximal inner product, one would obtain an algorithm with runtime $O(n^{2-\Theta(\epsilon)})$; namely, a comparable dependence on ϵ as would be obtained via locality sensitive hashing, but with worse constants. How could one improve upon this runtime? The natural hope is that one could employ a better embedding than the XOR/tensor embedding to achieve better concentration in the "uninteresting" inner products. Unfortunately, a classical structural result of Schoenberg from the 1940s suggests that this embedding is, in some sense, optimal:

Fact 10 (Schoenberg [35])**.** *Consider a function $g : \mathbb{R} \to \mathbb{R}$ which has the property that for any d, there exists $f : S^{d-1} \to \mathbb{R}^m$ such that $\langle f(u), f(v)\rangle = g(\langle u, v\rangle)$ for all $u, v \in S^{d-1}$, where S^{d-1} denotes the d-dimensional spherical shell. The Taylor expansion of g about 0 has exclusively nonnegative coefficients (and converges uniformly).*

Note that the degree q XOR/tensor embedding corresponds to the function $g(x) = x^q$, which, among degree q polynomials with non-negative coefficients, is, in some sense optimal for our purposes of amplifying the gap between the small and large inner products.

The crucial idea, however, is that we are not restricted to a single embedding. In particular, we can use two embeddings, f, h and create two copies of our vectors, and project each according to a different embedding, and then consider inner products across these two sets of vectors. Given such a pair of embeddings, we are no longer limited to Schoenberg's characterization; it is not hard to see that given *any* polynomial, p, one can design a pair of embeddings f, h such that for all vectors u, v $\langle f(u), h(v)\rangle = p(\langle u, v\rangle)$. Indeed, the XOR/tensor embedding shows how to obtain monomials $p(x) = x^q$, and by negating h, the corresponding embedding yields $p(x) = -x^q$, and an embedding corresponding to $p_1 + p_2$ can be obtained simply by appending the embeddings corresponding to p_1 and p_2.

Given the ability to fashion embeddings that implement any polynomial, the question is which polynomial to use? If we are in the setting in which all pairwise inner products lie in the interval $[\alpha, \beta]$, and we hope to find a pair of vectors with inner product at least $\beta - \epsilon$, then we wish to employ a polynomial that is as small as possible on the interval $[\alpha, \beta - \epsilon]$, and as large as possible outside this interval. Chebyshev polynomials, scaled appropriately are such extremal polynomials:

Fact. *(e.g. Thm. 2.37 of [33]) For any $x \notin [-1,1]$,*

$$T_q(x) = \max\left\{|p(x)| : p \in \mathcal{P}_q \text{ and } sup_{y\in[-1,1]}|p(y)| \leq 1\right\},$$

where T_q is the degree q Chebyshev polynomial (of the first kind), and $\mathcal{P}$ denotes the set of all degree q polynomials with real coefficients.

By employing a pair of embeddings that realize a "Chebyshev embedding", we are able to obtain an algorithm for finding pairs of unit vectors with inner product within an additive ϵ from that of the maximal pair, in time $O(n^{2-\Theta(\sqrt{\epsilon})})$. The $\sqrt{\epsilon}$ arises because $T_q(1+\epsilon)/T_q(1) \approx e^{q\sqrt{\epsilon}}$, in contrast to the polynomial $p(x) = x^q$ for which $p(1+\epsilon) \approx e^{q\epsilon}$, which would lead to a runtime of $O(n^{2-\Theta(\epsilon)})$. There are a number of details that we have not discussed in this overview, including why we are restricted in the degree of the polynomial.

This use of a pair of embeddings to eschew Schoenberg's characterization of what is implementable via a single embedding was also fruitfully leveraged by Alon and Naor in their work on approximating the cut norm of a matrix [2]. Our Chebyshev embedding appears to be novel, and because of its extremal nature, we believe it might be useful in other setting that are amenable to the use of a pair of embeddings.

From Maximal Inner Product to Closest Pair: In Section 4.2 we describe a reduction from the general $1+\epsilon$ approximate closest pair problem, to the problem of finding a pair of unit vectors whose inner product is within an additive ϵ of that of the maximal pair. While this section is rather involved, the techniques are, perhaps, less widely applicable than those of the other sections. The first step of this reduction shows how to convert the general problem into one in which all vectors have unit norm. Given this, provided that the minimum distance is bounded away from 0 (e.g. the maximum inner product is bounded away from 1), up to constant factors, finding the ϵ-closest pair is equivalent to finding the $c\epsilon$ maximal inner product. The main challenge of the reduction is handling the case when one has a set of unit vectors whose closest pair is extremely close. If this minimal distance is $> 1/n^{0.9}$, we can handle this case via another application of the "XOR/tensor" embedding to effectively increase all distances. If the minimal distance is $< 1/n^{0.9}$, we argue that we can efficiently cluster the points into sets of rather close points, with the property that after subtracting off the mean of each set and converting the set back into a set of unit vectors, the minimum distance is now $> 1/n^{0.9}$, and this process does not significantly alter the relative distances between different pairs of points.

Learning Parity with Noise: Given the ability to learn a parity with noise over n-bit strings, where the size of the parity set is c in time n^α, one can generally obtain an algorithm for learning parities of size k that runs in time roughly $n^{k\alpha/c}$. To illustrate, consider taking an instance of parity with noise where the size of the parity set is $2c$. One can form the instance of parity with noise over n^2-bit strings by creating an index corresponding to each of the pairs of original indices, and populating them with the XOR of the two corresponding values. If the original instance had a set of $2c$ indices whose XOR was correlated with the label, then the resulting instance (over n^2-bit strings) would have several sets of c indices whose XOR would correlate with the label, and hence one could apply an algorithm for learning parities of size c, with runtime $(n^2)^\alpha$ to find the set of c indices in the larger instance, corresponding to $2c$ indices in the original instance. Of course, the larger instance lacks the strong independence assumptions of the actual distribution of examples that the original instance had, though most algorithms should be robust to these mild dependencies.

This reduction, together with our most basic result for finding correlations (the light bulb problem), immediately yields an algorithm for learning parities of size k with noise η in time roughly $O(n^{k\frac{5-\omega}{2(4-\omega)}} poly(\frac{1}{1/2-\eta})) < O(n^{0.81k} poly(\frac{1}{1/2-\eta})$. The extreme noise robustness is due to the fact that the correlation did not appear in the exponent of n in our algorithm for finding correlations.

We obtain the very slightly improved exponent of $\omega k/3 < 0.8k$ by directly designing an algorithm for the $k = 3$ setting, as opposed to the $k = 2$ setting corresponding to the problem of finding correlations. Given an example x, y, for $x = (x_1, \ldots, x_n)$ and y is the corresponding label from an instance of parity with noise with $k = 3$, for any i, $\Pr[x_i = 1|y = 1] = 1/2$. Similarly, $\Pr[x_i x_j = 1|y = 1] = 1/2$ for distinct $i, j \in [n]$. The improved algorithm for finding parities rests on the following observation about parity sets of size $k = 3$: if the bits of x are not chosen uniformly at random, but instead are chosen independently to be 1 with probability $\frac{1}{2} + \alpha$, for some small bias α, then the above situation no longer holds. In such a setting, it is still the case that $\Pr[x_i = 1|y = 1] \approx \frac{1}{2} + \alpha$, however

$$\Pr[x_i x_j = 1|y = 1] = \begin{cases} \frac{1}{2} + \Theta(\alpha) & \text{if } i \text{ and } j \text{ are both in the true parity set,} \\ \frac{1}{2} + \Theta(\alpha^2) & \text{if } i \text{ or } j \text{ is not in the set of parity bits.} \end{cases}$$

The punchline of the above discrepancy is that very small biases—even a bias of $\alpha = 1/\sqrt{n}$ can be quite helpful. Given such a bias, for any pair $i, j \in [n]$, for sufficiently large n, even $n \cdot \text{polylog}\,(n)$ examples will be sufficient to determine whether i and j are both in the parity set by simply measuring the correlation between the ith and jth indices for examples with even label, namely estimating $\Pr[x_i x_j = 1|y = 1]$ based on the examples. How does one compute these $\binom{n}{2}$ correlations between vectors of length $n\text{polylog}\,(n)$ in time $o(n^3)$? By (fast) matrix multiplication. It is worth stressing that, provided this argument is sound, the resulting algorithm will be extremely noise-robust, since the discrepancy between $\Pr[x_i x_j = 1|y = 1]$ in the cases that i, j are both parity bits and the case that they are not will degrade linearly as $\eta \to 1/2$.

It should now be intuitively clear how to extend this approach from the small-biased setting to the setting in which the examples are generated uniformly at random, since a bias of $1/\sqrt{n}$ is quite modest. In particular, with constant probability, a random length–n example will have at least $\frac{n}{2} + \sqrt{n}$ positive indices, thus simply filtering the examples by removing those with fewer than $n/2$ positive indices should be sufficient to instill the necessary bias (at the minor expense of independence).

3 A New Algorithm for the Light Bulb Problem

We begin by presenting our new approach to the light bulb problem; our improved algorithm for finding the closest pair of vectors in the general setting will build upon this approach. There are two essential features of the light bulb problem which we exploit:

- With the exception of the correlated pair of vectors, the Hamming distance between pairs of vectors is tightly concentrated around a single value, $d/2$.
- One need not iteratively solve n nearest neighbor search problems, one for each vector; instead, one can effectively perform many such searches *simultaneously*.

We now provide an intuitive overview of how we exploit these features. We begin by assuming that we can choose the dimension of the vectors, d.

Given a $d \times n$ matrix X with entries in $\{-1, +1\}$ whose columns are uniformly random, with the exception of a pair of ρ-correlated columns, one naive approach to finding the correlated columns is to simply compute $W = X^t X$, the matrix whose i, jth entry is the inner product between the ith and jth columns of matrix X. With overwhelming probability, the largest off-diagonal entry of W will correspond to the correlated columns, as that entry will have value roughly $d\rho$, whereas all the other off-diagonal entries have expected value 0, and will be tightly concentrated around 0, with standard deviation $O(\sqrt{d})$. The obvious issue with this approach is that W has n^2 entries, precluding

a sub–quadratic runtime. This remains an issue even if the number of rows, d is taken to be near the information theoretic limit of $O(\frac{\log n}{\rho^2})$.

Our approach is motivated by the simple observation that if two columns of X are highly correlated, then we can compress X, by simply aggregating sets of columns. If one randomly partitions the n columns into, say, $n^{2/3}$ sets, each of size $n^{1/3}$, and then replaces each set of columns by a single vector, each of whose entries is given by the sum (over the real numbers) of the corresponding entries of the columns in the set, then we have shrunk the size of the matrix from $d \times n$, to a $d \times n^{2/3}$ matrix, Z (that now has integer entries in the range $[-n^{1/3}, n^{1/3}]$). It is still the case that most pairs of columns of Z will be uncorrelated. If, in the likely event that the two original correlated columns are assigned to distinct sets, the two columns of Z to which the two correlated columns contribute, will be *slightly* correlated. Trivially, the expected inner product of these two columns of Z is ρd, whereas the inner product between any two other columns of Z has expected value 0, and variance $O(n^{2/3}d)$. Thus provided $\rho d >> \sqrt{n^{2/3}d}$, and hence $d >> n^{2/3}/\rho^2$, there should be enough data to pick out the correlated columns of matrix Z, by computing $W' = Z^t Z$, and then finding the largest off-diagonal element. This computation of the product of an $n^{2/3} \times n^{2/3}/\rho^2$ matrix with its transpose, via fast matrix multiplication, is relatively cheap, taking time $n^{2\omega/3} poly(1/\rho) \leq n^{1.6} \cdot poly(1/\rho)$, where ω is the exponent of matrix multiplication.

Once one knows which two columns of Z contain the original correlated columns of W, one can simply brute-force check the inner products between all pairs of columns of W that contribute to the two correlated columns of Z, which takes time $dn^{2/3}$. (One could also recursively apply the algorithm on the two relevant sets of $n^{1/3}$ columns, though this would not improve the asymptotic running time.) The computation, now, is dominated by the size of the initial $n^{2/3}/\rho^2 \times n > n^{1.66}$ matrix! It is also clear that the runtime of this algorithm will depend only inverse polynomially on the correlation–in particular, ρ will not appear in the exponent of n. Optimizing the tradeoff between the size of the initial matrix, and the time spent computing the product, yields an exponent of $(5 - \omega)/(4 - \omega) < 1.62$.

Because our more general results will build off these ideas, we formally describe a slight extension of the above algorithm that applies to sets of vectors whose only guarantee is that most of the pairwise inner products are small, with the exception of a small number of large inner products. Since we will no longer make any assumptions that the entries of the vectors are chosen independently, our algorithm will employ a fairly standard trick of randomly flipping the signs of each vector to achieve pairwise independence between the inner products of different pairs of vectors. The author is grateful to Rasmus Pagh for pointing out this approach for dealing with a lack of independence.

Algorithm 11. VECTOR AGGREGATION

Input: *Two $m \times n$ matrices X, X' with entries $x_{i,j}, x'_{i,j} \in \mathbb{R}$, a constant $\alpha \in (0,1]$, and an integer $s \leq n^{2-2\alpha}$.*

Output: *s pairs of indices, $(c, c') \in [n]$.*

- *Repeat the following $10 \log n$ times:*
  - *Randomly partition $[n]$ into $n^{1-\alpha}$ disjoint subsets, each of size n^α, denoting the sets $S_1, \ldots, S_{n^{1-\alpha}}$.*
  - *For each $i = 1, 2, \ldots, 10 \log n$ do the following:*
    - *Select n coefficients $q_1, \ldots, q_n \in \{-1, +1\}$ at random.*
    - *Form the $m \times n^{1-\alpha}$ matrix Z^i with entries $z^i_{j,k} = \sum_{\ell \in S_k} q_\ell \cdot x_{j,\ell}$. and the $m \times n^{1-\alpha}$ matrix Z'^i with entries $q^i_{j,k} = \sum_{\ell \in S_k} q_\ell \cdot x'_{j,\ell}$.*
    - *Let $W^i = (Z^i)^\dagger Z'^i$.*
  - *Define the $n^{1-\alpha} \times n^{1-\alpha}$ matrix W with $w_{i,j} = quartile\{|w^1_{i,j}|, |w^2_{i,j}|, \ldots, |w^{10 \log n}_{i,j}|.\}$ where* quartile *is the smallest value greater than the lowest 75% of the entries.*
  - *Find the largest s entries of W, and for each such entry $w_{i,j}$ using a brute-force search, taking time $O(mn^{2\alpha})$ compute all the pairwise inner products between columns of X indexed by S_i and columns of X' indexed by S_j, and record those indices and inner product values of the largest s inner products computed. If this algorithm is being run in the setting with $X = X'$ then do the above brute-force search on the largest s off-diagonal entries of W.*
- *Traverse the list of $10s \log n$ saved indices and inner products, and output the s pairs with the largest inner products.*

We remark that the final brute-force search step of the above algorithm can be replaced by a recursive application of the entire algorithm to each subset of $2n^\alpha$ vectors, though for most ranges of parameters of interest, including the light bulb problem, the runtime of this brute-force search component is dominated by the computation of the matrix product $W^i = (Z^i)^\dagger Z'^i$.

The following proposition describes the performance of the above algorithm, and the proof is a straightforward application of tail bounds, arguing that if the pair of partitions S_j, S_k have the property that for all $\ell \in S_j$ and $h \in S_k$ the inner product of the ℓth column of X and hth column of X' is bounded by τ, then with high probability the entry of W corresponding to S_j, S_k satisfies $w_{j,k} \leq 3n^\alpha \tau$, in which case with high probability any entry $w_{j',k'}$ corresponding to sets that contains a pair of columns with large inner product will be detected and searched in the second-to-last step of the algorithm.

Proposition 12. *The algorithm* VECTOR-AGGREGATION*, when given as input s, α, and two $m \times n$ matrices with the property that for at most s pairs of columns u, v with u a column of X and v a column of X', $|\langle u, v \rangle| \leq \tau$, with probability $1 - o(1)$ will output all pairs of columns whose inner product is greater than $12n^\alpha \tau$. Additionally, the runtime of the above algorithm is*

$$\tilde{O}\left(mn + smn^{2\alpha} + timeMult(n^{1-\alpha}, m)\right),$$

where $timeMult(k, \ell)$ is the time it takes to multiply a $k \times \ell$ matrix by its a $\ell \times k$ matrix, and the $\tilde{O}$ hides a polylogarithmic factor of n.

Note that trivially, $timeMult(k, \ell) = O\left(\ell^\omega \max\left(1, \frac{k}{\ell}\right)^2\right)$ where $\omega < 2.4$ is the exponent of matrix multiplication, since this trivially holds if $k \leq \ell$, and if $k > \ell$, this this product can be computed via a series of $\left(\frac{k}{\ell}\right)^2$ computations of a $\ell \times \ell$ square matrix product. We use this bound in the following corollary for the light bulb problem, which follows from the above proposition by basic Chernoff bounds.

Corollary 13. *For any constant $\epsilon > 0$, the algorithm* Vector-Aggregation*, when given as input $\alpha = \frac{1}{2(4-\omega)}$, and $s = 1$, and two copies of the matrix X whose columns consist of the vectors given in an n, d, ρ instance of the light bulb problem with $d = \frac{n^{2\alpha+\epsilon}}{\rho^2}$, will output the true pair of correlated columns with probability $1 - o(1)$, and will run in time*

$$O\left(\frac{n^{\frac{5-\omega}{4-\omega}+\epsilon}}{\rho^{2\omega}}\right) < n^{1.62} \cdot \text{poly}(1/\rho).$$

The proof of Proposition 12 will rely on the following extremely crude anti-concentration lemma to argue that if an entry $w^i_{j,k}$ of W^i contains a contribution from a pair of columns with large inner product, then with a reasonable probability over the random choice of $q_1, \ldots, q_n$, the entry $w^i_{j,k}$ will not be too small.

Lemma 14. *Given a $t \times t$ matrix A with entries $a_{i,j}$, for $q_1, \ldots, q_t, r_1, \ldots, r_t \in \{\pm 1\}$ chosen uniformly at random,*

$$\Pr\left[|\sum_{i,j} q_i r_j a_{i,j}| \geq \frac{1}{4} \max_{i,j} a_{i,j}\right] \geq \frac{1}{4}.$$

Proof. Assume without loss of generality that $a_{1,1} = \max_{i,j} a_{i,j}$, and for ease of notation, define $a = q_1 r_1 a_{1,1}$. Let $b = \sum_{i \geq 2} q_i r_1 a_{i,1}$, $c = \sum_{j \geq 2} q_1 r_j a_{1,j}$, and $d = \sum_{i,j \geq 2} q_i r_j a_{i,j}$. For a given assignment to the q_i, r_j, if the sum in question has magnitude less than $a/4$, if $|a + b| \geq a/2$ then by flipping q_1 the sum would have magnitude at least $a/4$. Similarly, in the case that $|a + c| \geq a/2$ if r_1 were flipped, the sum would also have magnitude at least $a/4$. If neither of these conditions hold, then $|b + c| \geq a$, in which case by flipping both q_1 and r_1, the magnitude of the sum in question will be at least $\frac{3}{4}a \geq a/4$. Hence for every assignment to $r_2, \ldots, r_n$ and $q_2, \ldots, q_n$ there is at least a $1/4$ probability over the randomness in the assignment to r_1, q_1 that the sum in question exceeds $a/4$, and the lemma holds. □

Proof of Proposition 12. We begin by analyzing the value of $w^i_{j,k}$, viewed as a random variable over the choice of the n coefficients $q_1, \ldots, q_n$. Letting X_ℓ denote the ℓth column of matrix X, by definition

$$w^i_{j,k} = \sum_{\ell \in S_j, h \in S_k} q_\ell q_h \langle X_\ell, X'_h \rangle,$$

which is the sum of $n^{2\alpha}$ pairwise inner products of columns of the original matrices, each with a coefficient of ± 1. Since the coefficients of these inner products are pairwise independent, and each contribution has expectation 0 (since the probability of coefficients ± 1 are equally likely), their contributions to the variance of $w^i_{j,k}$ sums. Hence the contribution to $w^i_{j,k}$ contributed by pairs of columns u, v with $\langle u, v \rangle \leq \tau$ has variance bounded by $n^{2\alpha}\tau^2$. Hence by Chebyshev's inequality, with probability at most $1/9$, this contribution will not exceed $3n^\alpha\tau$ in magnitude. Given a partition $S_1, \ldots, S_{n^{1-\alpha}}$, for S_j, S_k such that all pairs of inner products between columns indexed by S_j of X and columns indexed by S_k of X' are at most τ, tail bounds on the binomial distribution yields that with probability $> 1 - 1/n^2$ over the randomness in the $10 \log n$ choices of coefficients $q_1, \ldots, q_n$, the top quartile of the magnitudes of $w^1_{j,k}, \ldots, w^{10 \log n}_{j,k}$ will be at most $3n^\alpha\tau$, and thus by a union bound over the $n^{2-2\alpha} \leq n^2$ entries of W, with high probability no such element $w_{j,k}$ will be this large. To conclude, by Lemma 14 with probability at least $1/4$, for any pair of sets S_j, S_k for which at least one of the contributing inner products $\langle X_\ell, X'_h \rangle > 12n^\alpha\tau$, the corresponding entry satisfies $|w^i_{j,k}| > 3n^\alpha\tau$, and hence, crudely, with probability at least $1/4$ the top quartile, $w_{j,k}$, of these $10 \log k$ choices of i will be at least this value. Hence for any pair of columns that have inner product at

least $12n^{\alpha}\tau$, the probability that it is discovered in each of the $10 \log n$ rounds of the algorithm is at least $(1 - n^{\alpha-1}) \cdot 1/4$ where the first factor is the probability, in the case that $X = X'$ that the two indices do not end up in the same partition S_j; if $X \neq X'$ then this factor vanishes, as we look in the diagonal entries of W. Thus with probability $1 - o(1)$, all such large inner products will be discovered by the end of the algorithm.

To analyze the runtime of the algorithm, note that, up polylogarithmic factors the runtime is either dominated by the computation of the matrices Z, Z', taking time $O(mn)$, the brute-force search phase, taking time at most $O(smn^{2\alpha})$, or the computation of the matrix product $W^i = (Z^i)^\dagger Z'^i$. □

3.1 Projecting Up

The algorithm Vector Aggregation described above shows how to efficiently find significantly correlated vectors from among a large number of extremely weakly correlated vectors. In this section, we describe a metric embedding which will amplify the gap between the "significantly" correlated vectors and the "weakly" correlated vectors.

As applied to the light bulb problem (Corollary 13), the algorithm of the previous section succeeds *provided that the points have dimension* $d \geq n^{\frac{1}{4-\omega}+\epsilon}/\rho^2 \approx n^{0.62}/\rho^2$. What happens if d is quite small? Information theoretically, one should still be able to recover the correlated pair even for $d = O(\log n/\rho^2)$. How can one adapt the Vector-Aggregation approach to the case when d is near this information theoretic boundary? We will perform a metric embedding that carefully projects the vectors into a *larger* space in such a way so as to guarantee that the projected vectors act like vectors corresponding to an n, d', ρ' instance of the light bulb problem for some $d' > d$, and $\rho' < \rho$, with the property that d' is sufficiently large so as ensure that the approach of the previous section succeeds. We rely crucially on the fact that ρ does not appear in the exponent of n in the runtime, since this transformation will yield $\rho' << \rho$.

While our embedding can be described more generally, we begin by describing it in the setting in which the entries of the vectors in question are ± 1. Consider randomly selecting a small set of the rows of the matrix whose columns consist of the set of vectors. We will produce a new row by component–wise multiplication. We term such a process of generating new dimensions (rows) as "XORing together a set of rows", and we claim that the new row of data thus produced is reasonably faithful. In particular, if two columns are completely correlated, then the result of XORing together a number of rows will produce a row for which the values in the two correlated columns will still be identical. If the correlation is not 1, but instead ρ, after combining q rows, the corresponding columns will only be ρ^q correlated, as XORing degrades the correlation. Recall, however, that the algorithm of the previous section was extremely noise robust, and thus we can afford to degrade the correlation considerably. For constant ρ, we can certainly take $q = o(\log n)$ without increasing the exponent of n in the runtime.

Note that as we are viewing the vectors as having entries in $\{-1, 1\}$, this XORing of sets of rows is simply component-wise multiplication of the rows. Equivalently, it can be seen as replacing each column with a sample of the entries of the qth tensor power of the column.

In the context of learning parity with noise, this expansion approach was used by Lyubashevsky [21] to show that given few examples from an instance of learning parity with noise, one can generate new "simulated" examples, that can be used in place of actual examples. In contrast to the current setting, the challenge in that work was arguing that the new instances are actually information theoretically *indistinguishable* from new examples (with higher noise rate). To prove this strong indistinguishability, Lyubashevsky employed the "Leftover Hash Lemma" of Impagliazzo and Zuckerman [17].

In our setting, we do not need any such strong information theoretic guarantees; Proposition 12

only requires some guarantees on the inner products of pairs of columns, which are given by repeated application of the following trivial lemma, together with Chernoff bounds:

Lemma 15. *Given vectors $u, v, w, z \in \mathbb{R}^d$ with $\langle u, v\rangle = \rho_1 d$ and $\langle w, z\rangle = \rho_2 d$, for i, j chosen uniformly at random from $[d]$,*

$$E[(u_i w_j) \cdot (v_i z_j)] = \rho_1 \rho_2.$$

Phrased differently, letting $x \in \mathbb{R}^{d^2}$ be the vector whose entries are given by the d^2 entries of the outer-product uw^t, and y is given by the entries of vz^t, then $\langle x, y\rangle = \rho_1\rho_2 d^2$. Elementary concentration bounds show that provided one samples sufficiently many indices of this outer product, the inner product between the sampled vectors will be close to this expected value (normalized by the dimension).

Proof. The proof follows from the independence of i, j, the facts that $E[u_i v_i] = \rho_1, E[w_j z_j] = \rho_2$, and the basic fact that the expectation of the product of independent random variables is the product of their expectations. □

We now describe our algorithm, which applies more generally than the light bulb problem setting, and can alternately be viewed as an algorithm for approximating the product of two ± 1 matrices, under the assumption that the product has only a moderate number of large entries.

Algorithm 16. EXPAND AND AGGREGATE
Input: An $m \times n$ matrix X with entries $x_{i,j} \in \{-1, +1\}$, real numbers $\rho, \tau \in (0, 1)$, with $\rho > \tau$, and an integer $s > 0$
Output: A set of s pairs of indices $(c, c') \in [n]$.

- Set $\alpha = \frac{1}{2(4-\omega)}$, for $\omega < 2.4$ is the exponent of matrix multiplication.
- Let $m' = n^{2\alpha + \frac{\log \rho}{\log \tau}} \log^4 n$, and $q = \frac{\log n}{-2 \log \tau}$.
- If $m' \geq n$, calculate all pairwise inner products, taking time $O((m')^\omega)$.
- Otherwise, we create an $m' \times n$ matrix Y with entries in $\{-1, +1\}$:
  - For each of the m' rows of Y, select a list $t_1, \ldots, t_q$ with each t_i selected uniformly at random from $[m]$, and set the jth component of the corresponding row to be $\prod_{i=1}^{q} x_{t_i, j}$.
- Output the result of running algorithm VECTOR-AGGREGATION on input Y with the parameters α and s, where the brute-force searches can be performed on the original columns of X.

The intuition of the above algorithm is that the matrix Y resulting from the XOR/tensor expansion step has the property that the expected inner product between any two "bad" columns is bounded in magnitude by $m'\tau^q = m'\frac{1}{\sqrt{n}}$, and the expected inner product of a "good" pair of vectors will be $m'\rho^q = m' n^{-\frac{\log \rho}{2 \log \tau}} >> m'\frac{1}{\sqrt{n}}$. Chernoff bounds will then guarantee that all inner products are closely concentrated about their expectations, to within $\pm\sqrt{m'}\text{polylog}\, n$, and hence the matrix Y satisfies the assumptions of Proposition 12 in which case the output of algorithm VECTOR-AGGREGATION will be as desired.

The performance of the above algorithm is summarized by the following theorem:

Theorem 1 *Consider a set of n vectors in $\{-1, 1\}^d$ and constants $\rho, \tau \in [0, 1]$ with $\rho > \tau$ such that for all but at most s pairs u, v of distinct vectors, $|\langle u, v\rangle| \leq \tau d$. There is an algorithm that, with*

probability $1 - o(1)$, *will output* all *pairs of vectors whose inner product is least* ρd. *Additionally, the runtime of the algorithm is*

$$\tilde{O}\left(sdn^{1/(4-\omega)} + n^{\frac{5-\omega}{4-\omega}+\omega\frac{\log\rho}{\log\tau}}\right) \leq O\left(sdn^{0.7} + n^{1.62+3\frac{\log\rho}{\log\tau}}\right),$$

where $\omega < 2.4$ *is the exponent of matrix multiplication, and the* $\tilde{O}$ *notation hides polylogarithmic factors of* n.

The above theorem together with basic Chernoff bounds immediately yields Corollary 3, describing an algorithm for the light bulb problem in the setting in which the dimension d is near the information theoretic limit of $O(\frac{\log n}{\rho^2})$.

Corollary 3 *For any constants* $\rho, \epsilon > 0$, *there exists a constant* c_ϵ *dependent on* ϵ *such that for* $d \geq c_\epsilon \frac{\log n}{\rho^2}$, *there is an algorithm that, with probability* $1 - o(1)$, *will find a planted set of* ρ*-correlated vectors from among* n *random vectors in* $\{\pm 1\}^d$, *and will run in time*

$$\tilde{O}(n^{\frac{5-\omega}{4-\omega}+\epsilon}) \leq O(n^{1.62}),$$

were $\omega > 2.4$ *is the exponent of matrix multiplication.*

Proof of Theorem 1. From the independent generation of each row of Y, and the fact that all entries are either ± 1, a union bound over Chernoff bounds guarantees that with probability $1 - o(1)$, all the n^2 pairwise inner products between columns of Y will be within an additive $\sqrt{m'}\log n$ of their expectations. By Lemma 15, for a pair of columns of X with inner product with magnitude at most τm, the expected inner product of the corresponding columns of matrix Y will have magnitude at most $m'\tau^q = m'n^{-1/2} \leq \sqrt{m'}$, and hence by the above tail bounds will be at most $2\sqrt{m'}\log n = 2n^{\alpha+\frac{\log\rho}{2\log\tau}}\log^3 n$ with the claimed probability. Similarly, for a pair of columns with inner product at least ρd, with the claimed probability the inner product between the corresponding columns of Y will be at least $m'\rho^q - \sqrt{m'}\log n = m'n^{-\frac{\log\rho}{2\log\tau}} - \sqrt{m'}\log n \geq \frac{1}{2}n^{2\alpha+\frac{\log\rho}{2\log\tau}}\log^4 n$. Letting $\tau' = 2n^{\alpha+\frac{\log\rho}{2\log\tau}}\log^3 n$, note that for sufficiently large n, the "big" inner products satisfy $\frac{1}{2}n^{2\alpha+\frac{\log\rho}{2\log\tau}}\log^4 n \geq 12n^\alpha \cdot 2n^{\alpha+\frac{\log\rho}{2\log\tau}}\log^3 n = 12n^\alpha\tau'$, and hence the requirements of Proposition 12 are satisfied with probability $1 - o(1)$ over the randomness in the construction of matrix Y, and the theorem follows from Proposition 12. □

4 The Chebyshev Embedding, and Closest-Pair Problem

We now abstract and refine the main intuitions behind the Expand and Aggregate algorithm, to yield our algorithm for the general approximate closest pair problem, which will work in both the Boolean and Euclidean settings. We extend the ideas of the previous section in two steps. First we consider the problem of finding a pair of vectors whose inner product is within an *additive* ϵ from that of the pair with maximal inner product, given that all vectors have unit norm. Accomplishing this step will require the *Chebyshev embedding*—a more powerful embedding than the XOR/tensor embedding of the previous setting. In Section 4.2 we then extend our algorithm for finding the approximately maximal inner product from a set of unit vectors, to the general problem of finding a pair of vectors whose Euclidean distance is within a multiplicative $(1 + \epsilon)$ from that of the closest pair.

4.1 The Chebyshev Embedding

The VECTOR-AGGREGATION algorithm of the previous section relies, crucially, on the tight concentration around 0 of the inner products of the uncorrelated vectors. In the case of Proposition 12, this concentration came "for free", because we assumed that the dimensionality of the data was large $\approx n^{0.6}$. To obtain Theorem 1, we needed to work to obtain sufficiently tight concentration. In particular, we performed a metric embedding $f : \{-1,+1\}^d \to \{-1,+1\}^m$, with the crucial property that for an appropriately chosen integer q, for $u, v \in \{-1,+1\}^d$,

$$\frac{\langle f(u), f(v)\rangle}{m} \approx \left(\frac{\langle u, v\rangle}{d}\right)^q.$$

The key property of this mapping $x \to x^q$ is that if one pair of vectors has an inner product that is a factor of $(1+\epsilon)$ larger than than that of any other pair, after performing this mapping, the inner product of the image of the close pair will now be a factor of $(1+\epsilon)^q >> 1+\epsilon$ larger than that of the images of any other pair of vectors; thus the "gap" has been significantly expanded. Of course, we can not take q to be arbitrarily large, as we would like to maintain a subquadratic amount of data and thus $m << n$, and the variance in the inner products that arises from the subsampling process (choosing which subsets of the rows to XOR) will be $O(m)$. Thus if q is so large that the $O(\sqrt{m})$ standard deviation in the inner product dominates the $m\rho^q$ inner product of the images of the closest pair, all the signal in the data will be lost and the algorithm will fail. (Phrased more generally, if q is too large, the distortion caused by projecting to a lower dimensional space will swamp the signal.)

A not-so-simple calculation shows that if we try to obtain an algorithm for the problem of finding a pair of unit vectors with inner product within ϵ of the maximal inner product (or the $(1+\epsilon)$ approximate closest pair problem) via this EXPAND AND AGGREGATE approach, we would end up with an algorithm with runtime $n^{2-O(\epsilon)}$. Can we do any better? To simplify the exposition, assume that we are told that there is a "good" pair of vectors with inner product at least $1/2+\epsilon$, and that all other pairs of vectors are "bad" and have inner product in the range $[-1/2, 1/2]$. In order to improve upon this runtime of $n^{2-O(\epsilon)}$, we need an improved embedding—one that damps the magnitudes of the "bad" pairs of vectors as much as possible, while preserving the inner product between the closest pair. Specifically, we seek a mapping $f_c : \mathbb{R}^d \to \mathbb{R}^m$ with the following properties:

- For all $u, v \in \mathbb{R}^d$, if $\langle u, v\rangle \geq \frac{1}{2} + \epsilon$, then $\langle f_c(u), f_c(v)\rangle \geq c$.
- For all $u, v \in \mathbb{R}^d$, if $\langle u, v\rangle \in [-\frac{1}{2}, \frac{1}{2}]$, then $\langle f_c(u), f_c(v)\rangle$ is as small as possible.
- For all $u \in \mathbb{R}^d$, $f_c(u)$ can be computed reasonably efficiently.

The dimension of the image, m, is not especially important, as we could always simply choose a random subset of the dimensions to project onto while roughly preserving the inner products (provided this can all be computed efficiently). In general, it is not clear what the optimal such embedding will be, or how extreme a "gap amplification" we can achieve. A classical result of Schoenberg from the 1940s [35], however, characterizes the what can be achieved via a specific type of embedding. Formally, he characterized the set of functions $g : \mathbb{R} \to \mathbb{R}$ which have the property that for any d, there exists $f : S^{d-1} \to \mathbb{R}^m$ such that $\langle f(u), f(v)\rangle = g(\langle u, v\rangle)$ for all $u, v \in S^{d-1}$, where S^{d-1} denotes the d-dimensional spherical shell. In particular, he showed that a necessary and sufficient condition for such functions g is that their Taylor expansion about 0 has exclusively nonnegative coefficients (and converges uniformly). It is not hard to see that for such polynomials g of a given degree, the XOR/tensor embedding $g(x) = x^q$ has the optimal gap amplification.

The crux of our improved Chebyshev embedding is the realization that we are not constrained to a single embedding. For our purposes, if we can construct a pair of embeddings, $f, h : S^{d-1} \to \mathbb{R}^m$ with the property that $\langle f(u), h(v)\rangle = g(\langle u, v\rangle)$, for some function g with superior gap amplification, then we can create two copies of the set of vectors in question, embed one copy according to f

and the other according to h, and then consider inner products across the two sets. The power of two embeddings, is that we can realize *any* polynomial g. To see this, note that the XOR/tensor embedding can realize any monic polynomial $p(x) = x^q$, given a pair of embeddings f, h, by setting $f = -h$ one can obtain negative monomials $p(x) = -x^q$, and by simply concatenating the embeddings that realize polynomials p_1, p_2, one obtains an embedding that realizes $p_1 + p_2$.

Given this power of two embeddings, the question is now *which polynomials should we use?* The following fact suggests an embedding which, at least among a certain class of embeddings, will be optimal.

Fact. *(e.g. Thm. 2.37 of [33]) For any $x \notin [-1, 1]$,*

$$T_q(x) = \max\left\{|p(x)| : p \in \mathcal{P}_q \text{ and } sup_{y\in[-1,1]}|p(y)| \leq 1\right\},$$

where T_q is the degree q Chebyshev polynomial (of the first kind), and $\mathcal{P}$ denotes the set of all degree q polynomials with real coefficients.

Perhaps the most surprising aspect of this simple fact is that a single polynomial, T_q captures this extremal behavior for all x. The following fact quantifies the properties of the Chebyshev polynomials that we will rely on:

Fact 17. *Letting $T_q(x) := \frac{(x-\sqrt{x^2-1})^q+(x+\sqrt{x^2-1})^q}{2}$ denote the qth Chebyshev polynomial (of the first kind), the following hold:*

- *$T_q(x)$ has leading coefficient 2^{q-1}.*
- *$T_q(x)$ has q distinct real roots, all lying within the interval $[-1, 1]$.*
- *For $x \in [-1, 1]$, $|T_q(x)| \leq 1$.*
- *For $\delta \in (0, 1/2]$, $T_q(1 + \delta) \geq \frac{1}{2}e^{q\sqrt{\delta}}$.*

Proof. The first 3 properties are standard facts about Chebyshev polynomials (see, e.g. [36]). To verify the fourth fact, note that for δ in the prescribed range, $\sqrt{(1+\delta)^2 - 1} \geq \sqrt{2\delta}$, and we have the following: $T_q(1 + \delta) \geq \frac{1}{2}(1 + \delta + \sqrt{2\delta})^q \geq \frac{1}{2}(1 + \sqrt{2\delta})^q \geq \frac{1}{2}e^{q\sqrt{\delta}}$. □

To illustrate the high-level idea, we return to our example above in which we hope to isolate a pair of vectors with "large" inner product $1/2 + \epsilon$ from among a set of vectors with "small" inner products in the range $[-1/2, 1/2]$. We will construct an embedding corresponding to the monic polynomial $P(x) = T_q(2x)/2^{2q-1}$, where $T_q(x)$ is the qth Chebyshev polynomial (of the first kind). Note that since $T_q(x)$ has q roots, all in the interval $[-1, 1]$, the polynomial $P(x)$ will also have q real roots in the interval $[-1/2, 1/2]$. The corresponding mappings f, h, constructed as described above, will have the property that $\langle f(u), g(v)\rangle = P\left(\langle u, v\rangle\right)/2^q$. Roughly, we will choose $q = O(\log n)$, and hence the the multiplicative gap between the "large" inner product and the "small" inner products will be $e^{q\sqrt{\epsilon/2}} = n^{O\sqrt{\epsilon}}$, hence we will be able to aggregate sets of $n^{O(\sqrt{\epsilon})}$ vectors in the Vector Aggregation algorithm. We will ensure that the image of the original vectors have dimension $m << n^{0.29}$, hence the most computationally expensive step of our algorithm will be the computation of the product of an $n^{1-\Theta(\sqrt{\epsilon})} \times m$ matrix and an $m \times n^{1-\Theta(\sqrt{\epsilon})}$ matrix, using fast *rectangular* matrix multiplication (see Fact 23), which will have runtime $O(n^{2(1-\Theta(\sqrt{\epsilon})})$. We now formally define the *Chebyshev embedding*. We define this embedding in the setting in which the vectors in question have values ± 1; this uniformity ensures that the entries of the vectors returned by the embedding have the same magnitudes, and hence are amenable to Chernoff bounds to guarantee that the inner products

between the returned vectors are concentrated about their expectations. In the following section in which we use this embedding, we give a general procedure for converting a set of arbitrary unit vectors to a set of unit vectors with entries $\pm 1/\sqrt{d}$, while roughly preserving the inner products.

Algorithm 18. CHEBYSHEV EMEBEDDING
Input: Two $m \times n$ matrices X, X' with entries $x_{i,j} \in \{\pm 1\}$, real numbers $\tau^-, \tau^+ \in [-1, 1]$ with $\tau^- < \tau^+$ and integers q and m'.
Output: Two $m' \times n$ matrices Y, Y' with entries $y_{i,j} \in \{\pm 1\}$.

- Let T_q denote the degree q Chebyshev polynomial (of the first kind), with roots at $r_1, \ldots, r_q \in (0, 1)$.
- Each of the m' rows of the output matrices Y, Z are populated as follows:
  - We will populate two sets of q vectors $s_1, \ldots, s_q$, and $t_1, \ldots, t_q$ each of length n as follows. For $i = 1, \ldots, q$:
    - With probability $1/2$, choose a random index $j \in [m]$ and set both s_i to be the jth row of X and t_i to be the jth row of X'.
    - Let $c_i = \tau^- + \frac{1+r_i}{2}(\tau^+ - \tau^-)$ be the location of the ith root of T_q after the support has been scaled so that the roots lie within $[\tau^-, \tau^+]$ rather than $[-1, 1]$.
    - With probability $\frac{1-c_i}{4}$ set both s_i and t_i to be the all ones vectors.
    - With probability $\frac{1+c_i}{4}$ set s_i to be the all ones vector, and t_i to be the all minus ones vector.
  - Define the ith rows of Y and Y' to be the component-wise products of the s_i's and t_i's respectively:

$$Y_{i,j} = \prod_{\ell=1}^{q} s_\ell(j), \quad Y'_{i,j} = \prod_{\ell=1}^{q} t_\ell(j),$$

  where $s_\ell(j)$ and $t_\ell(j)$ denote the jth entries of the vectors s_ℓ and t_ℓ, respectively.

The following proposition characterizes the Chebyshev embedding.

Proposition 19. *Let Y, Y' be the matrices output by the algorithm* CHEBYSHEV EMBEDDING *on input $X, X', \tau^+, \tau^-, q, m'$. With probability $1 - o(1)$ over the randomness in the construction of Y, Y', for all $i, j \in [n]$, $\langle Y_i, Y'_j \rangle$ is within $\sqrt{m'} \log n$ from the value*

$$T_q\left(\frac{\langle X_i, X'_j \rangle - \tau^-}{\tau^+ - \tau^-} 2 - 1\right) \cdot m' \cdot (\tau^+ - \tau^-)^q \cdot \frac{1}{2^{3q-1}},$$

where T_q is the degree q Chebyshev polynomial of the first kind. Additionally, the algorithm runs in time $O(m'nq)$.

In particular, the above proposition states that the value of the inner products of columns of Y, Y' are concentrated around the Chebyshev polynomial that has been scaled so that its roots lie in the interval $[\tau^-, \tau^+]$, evaluated at the inner product of the corresponding columns of X, X', multiplied by an appropriate scaling factor.

Proof of Proposition 19. The fact that all inner products are concentrated within $\pm\sqrt{m'} \log n$ about their expectations follows from the fact that each row of Y, Y' is generated independently, and all entries of these matrices are ± 1, and hence a union bound over basic Chernoff bounds yields the desired concentration. We now analyze the expectation of the inner products. Let u, u' be columns

of X, X' respectively, and v, v' the corresponding columns of Y, Y'. Letting $x = \frac{\langle u,u' \rangle}{m}$, by Lemma 15,

$$\mathrm{E}[\langle v, v' \rangle] = m' \prod_{i=1}^{q} \frac{x - c_i}{2},$$

where c_i is the location of the ith root of the qth Chebyshev polynomial after the roots have been scaled to lie in the interval $[\tau^-, \tau^+]$. The proposition now follows from noting that the qth Chebyshev polynomial has leading coefficient 2^{q-1}, whereas the above expression when expressed as a polynomial in x has leading coefficient $1/2^q$, disregarding the factor of the dimension m', and then noting that if one has two monic degree q polynomials, P and Q where the roots of Q are given by scaling the roots of P by a factor of α, then the values at corresponding locations differ by a multiplicative factor of $1/\alpha^q$; since the roots of T_q lie between $[-1, 1]$ and the roots of the polynomial constructed in the embedding lie between $[\tau^-, \tau^+]$, this corresponds to taking $\alpha = \frac{2}{\tau^+ - \tau^-}$. □

4.2 Finding Vectors with Maximal Inner Product

We now describe how to employ the Chebyshev embedding of the previous section to obtain an algorithm for finding a pair of vectors with inner product within ϵ from that of the maximal inner product, for arbitrary Euclidean vectors with unit norm. The rough outline is to first convert the set of vectors into vectors with entries ± 1, then apply the Chebyshev embedding, and then simply run the VECTOR AGGREGATION algorithm from Section 3.

Algorithm 20. MAKE UNIFORM
Input: An $m \times n$ matrix X with entries $x_{i,j} \in \mathbb{R}$ whose columns have unit Euclidean norm, and $\delta \in (0, 1)$.
Output: An $m' \times n$ matrix Y with entries $y_{i,j} \in \{\pm 1\}$, where $m' = \frac{10 \log n}{\delta^2}$.

- For each $i = 1, \ldots, m'$, select a random unit vector $v \in \mathbb{R}^m$, and let $w = v^t X$. For all $j = 1, \ldots, n$, set $y_{i,j} = sign(w_j)$.

The following basic lemma characterizes the performance of the above algorithm:

Lemma 21. *Letting Y denote the output of running Algorithm 20 on input X, δ, where X is a matrix whose columns have unit norm, with probability $1 - o(1/n^2)$, for all pairs $i, j \in [n]$,*

$$\left| \frac{\langle Y_i, Y_j \rangle}{m'} - \left(1 - 2\frac{\cos^{-1}(\langle X_i, X_j \rangle)}{\pi}\right) \right| \leq \delta,$$

where X_i, Y_i denote the ith columns of X and Y, respectively. And thus if $\langle Y_i, Y_j \rangle \geq \max_{k \neq \ell} \langle Y_k, Y_\ell \rangle - \delta m'$, then with probability $1 - o(1)$, $\langle X_i, X_j \rangle \geq \max_{k \neq \ell} \langle X_k, X_\ell \rangle - 2\pi\delta$. Additionally, the runtime of the algorithm is $O(\frac{mn \log n}{\delta^2})$.

Proof. Letting α denote the angle between X_i and X_j, hence $\langle X_i, X_j \rangle = \cos(\alpha)$, for any $k \in [m']$,

$$\Pr[y_{k,i} y_{k,j} = -\frac{1}{m'}] = \Pr[r \in [0, \alpha]] = \frac{\alpha}{\pi},$$

where $r \leftarrow Unif[0, \pi]$, is selected uniformly at random from the interval $[0, \pi]$. Hence

$$E[\langle Y_i, Y_j \rangle] = 1 - 2\frac{\alpha}{\pi}.$$

Since all entries $y_{i,j} \in \pm\frac{1}{\sqrt{m'}}$, and the different dimensions are chosen independently, a union bound over n^2 Chernoff bounds yields that with the claimed probability, all pairwise inner products will be within $\pm\delta$ of their expectations.

The second claim follows from noting that the above guarantees that if $\langle Y_i, Y_j\rangle \geq \max_{k\neq\ell}\langle Y_k, Y_\ell\rangle - \delta$, then with the claimed probability the expected inner product of Y_i and Y_j is at most 2δ smaller than that of the maximal expected inner product, and hence the angle between the corresponding columns of X is at most $2\pi\delta$ smaller than that of the optimal pair, and hence the inner products of the corresponding columns of X are at most $2\pi\delta$ smaller than the optimal inner product, since the magnitude of the derivative of the cosine function is at most 1. □

We now describe the main algorithm of this section:

Algorithm 22. Approximate Maximal Inner-Product
Input: An $m \times n$ matrix X with entries $x_{i,j} \in \mathbb{R}$ whose columns have unit Euclidean norm, and $\epsilon \in (0,1)$.
Output: Two distinct indices $c_1, c_2 \in [n]$, s.t. with probability $1 - o(1)$, $\langle X_{c_1}, X_{c_2}\rangle \geq \max_{i\neq j}\langle X_i, X_j\rangle - \epsilon$, where X_c denotes the cth column of matrix X.

- *Let X' denote the $m' \times n$ matrix with $m' = O(\frac{\log n}{\epsilon^2})$ resulting from applying algorithm 20 (Make Uniform) to matrix X with input parameter $\delta = \epsilon/2\pi$.*
- *Choose $n^{1.5}$ random pairs of distinct columns of X', for each pair compute their inner product and let v_{max} be the maximum such inner product divided by m'.*
- *Define $m'' = n^{0.2}$ and $q = \frac{1}{50}\log n$ and let Y, Y' be the $m'' \times n$ matrices returned by running algorithm Chebyshev Embedding on inputs X', X', q, m'' and with $\tau^- := -1$, $\tau^+ := v_{max}$.*
- *Run algorithm Vector Aggregation on the matrices Y, Y' with $s := n$, and $\alpha := \frac{\sqrt{\delta}}{100}$, and return the indices corresponding to the largest of the s returned inner products if it is at least v_{max}, otherwise output the indices corresponding to v_{max}.*

Theorem 4. *The algorithm* Approximate Maximal Inner-Product*, when run on an $m \times n$ matrix X, whose columns have unit Euclidean norm and a sufficiently small constant approximation parameter $\epsilon > 0$ will, with probability at least $1 - o(1)$, output a pair of indices c_1, c_2 such that $\langle X_{c_1}, X_{c_2}\rangle \geq \max_{i\neq j}\langle X_i, X_j\rangle - \epsilon$, where X_i denotes the ith column of matrix X. Additionally, the runtime of the algorithm is $\tilde{O}(mn + n^{2-\frac{1}{200}\sqrt{\epsilon}})$, where the $\tilde{O}$ hides a polylogarithmic factor of n. Additionally, by repeating the algorithm and outputting the pair with maximum inner product over all runs, the probability of failure can be reduced exponentially.*

The proof of the above theorem will follow relatively easily from the guarantees on the Chebyshev Embedding and the Vector Aggregation algorithms given by Proposition 12 and Proposition 19. Additionally, we will rely on fast *rectangular* matrix multiplication—namely that sufficiently skinny matrices can be multiplied in nearly quadratic time. The following result of Coppersmith summarizes this fact.

Fact 23 (Coppersmith [11])**.** *For any positive $\gamma > 0$, provided $\beta < .29$, the product of a $k \times k^\beta$ with a $k^\beta \times k$ matrix can be computed in time $O(k^{2+\gamma})$.*

Proof of Theorem 4. By Lemma 21, with probability $1 - o(1)$ the matrix X' returned by the Make Uniform algorithm will have the property that any pair of columns whose inner product is within an additive $m'\delta = m'\epsilon/2\pi$ of the inner product of the maximal pair, will correspond to a pair of columns of the original matrix X whose inner product is within an additive ϵ of that of the maximal

pair. We now argue that the remainder of the algorithm, with high probability, will find columns of X' whose inner product is within $\delta m'$ from that of the maximal pair.

If $v_{max}m'$ is within $\delta m'$ from the maximal inner product, then the algorithm trivially succeeds. Otherwise, by a Chernoff bound, with probability at least $1 - o(1)$ there are at most n pairs of columns whose inner product is greater than $v_{max}m'$, since if this were not the case, we would have expected to see at least $\sqrt{n}$ such pairs in the set of $n^{1.5}$ random pairs surveyed, but instead we saw no such pairs. Hence in this case, there are at most n pairs with inner product greater than $v_{max}m'$, and there is some pair with inner product at least $(v_{max} + \delta)m'$. We now argue that the properties of the Chebyshev embedding together with the guarantees on the performance of the AGGREGATE VECTORS algorithm will guarantee that with high probability, all pairs of columns with inner product at least $(v_{max} + \delta)m')$ will be returned.

Note that with probability $1 - o(1)$ it will be the case that $v_{max} > -1/2$ simply because for any constant $c > 0$, there can not be a super-constant fraction of unit vectors with inner products less than $-c$, hence with this probability $(1 + v_{max}) > 1/2$. By Proposition 19 with probability $1 - o(1)$, for all pairs of columns X'_i, X'_j of matrix X such that $\langle X'_i, X'_j\rangle \leq v_{max}m'$, the corresponding inner product of their Chebyshev embeddings will satisfy $|\langle Y_i, Y'_j\rangle| \leq m''2^q\frac{1}{2^{3q-1}} + \sqrt{m''}\log n \leq 3n^{0.18}$ for sufficiently large n, by our choice of m'' and q. Letting i, j denote the indices of the columns with the largest inner product which we are assuming is at least $(v_{max}+\delta)m'$, and k, ℓ denoting indices of any pair of columns with $\langle X'_k, X'_\ell\rangle \leq v_{max}m'$, Proposition 19 also yields that $\frac{\langle Y_i, Y'_j\rangle}{\langle Y_i, Y'_j\rangle} \geq \frac{1}{2}T_q(1 + \delta)$, where the factor of two is a crude way of dealing with the additive $\pm\sqrt{m''}\log n$ error in the inner products introduced via the randomness of the Chebyshev embedding, which is insignificant in comparison to $(1 + v_{max})^q\frac{m''}{2^{3q-1}}$, which is a lower bound on the expected inner product of the images of a pair of columns of X' whose inner product is at least $v_{max}m'$.

By Fact 17, $T_q(1 + \delta) > \frac{1}{2}e^{q\sqrt{\delta}} = \frac{1}{2}n^{\sqrt{\delta}/50} >> n^\alpha = n^{\sqrt{\delta}/100}$ for sufficiently large n, hence the conditions of Proposition 12 hold, and with high probability the execution of the AGGREGATE VECTORS algorithm will output all pairs of indices corresponding to columns of X' whose inner products are at least $(v_{max} + \delta)m'$.

To conclude our proof of the theorem, note that runtime of the above algorithm, up to poly-logarithmic factors of n, is either dominated by the $\tilde{O}(mn)$ time to perform MAKE UNIFORM, or is dominated by the computation of the matrix product in the VECTOR AGGREGATION algorithm: $timeMult(n^{1-\alpha}, m'') = timeMult(n^{1-\sqrt{\epsilon/2\pi}/100}, n^{0.2})$. By Fact 23, this runtime is bounded by $O(n^{2-\sqrt{\epsilon}/150+\gamma})$ for any positive constant γ. The theorem now follows from taking γ to be a sufficiently small multiple of $\sqrt{\epsilon}$. □

4.3 The Approximate Closest Pair

Given the algorithm of the previous section for finding an additively approximate maximal inner product in a set of unit vectors, there are two minor hurdles to address in order to obtain an algorithm for finding a multiplicatively $(1 + \epsilon)$ approximation to the closest pair of points from among an arbitrary set of vectors. The first hurdle is translating the problem of finding a close pair of arbitrary vectors, to finding a close pair of unit vectors. This can be relatively easily accomplished by adding a rather large randomly chosen vector v to all other vectors, then normalizing the vectors so as to have unit norm. Provided the vector v has magnitude significantly more than the maximum magnitude of all the vectors of interest, and the dimensionality of the space is sufficiently high so as to guarantee that v is nearly orthogonal to the chords connecting all pairs of the vectors of interest, this operation will have low distortion (and will simply scale all distances by roughly the same factor). Algorithm 25 (STANDARDIZE) given below accomplishes this transformation.

Having reduced the problem to the setting in which all vectors have unit norm, the "law of cosines" $||v-w||^2 = ||v||^2 + ||w||^2 - 2\langle v, w\rangle$, relates the distance between vectors to the inner products. This will allow us to very easily translate between multiplicative $1+\epsilon$ bounds on the distance and additive $c\cdot\epsilon$ bounds on the inner product (for a constant c), provided that the closest distance is not too small. If the closest distance is too small for an additive guarantee on the inner product to correspond to a meaningful multiplicative guarantee on the distance, we have two approaches. If the minimum distance is at least $1/n^{0.9}$, we can amplify this distance, while only increasing the multiplicative gap between distances via an analog of the XOR/tensor embedding. Algorithm 31 (Approximate Closest Pair (large α)) addresses this setting. If the minimum distance is less than $1/n^{0.9}$, then, since all vectors have unit norm, intuitively, we should be able to find it with a divide-and-conquer approach. Since either there are only a few pairs of points that are very close in which case we can easily pick them off, or there is a large set of points that are all extremely close in which case we will be able to subtract off their mean, and rescale that cluster of points; such a transformation has the property that it effectively increases the minimal distance by a factor of $1/z$, where z is the diameter of the cluster of points. Algorithm 29 (Approximate Closest Pair (small α)) addresses this setting.

We note that we may assume that we know the distance between the closest pair, up to a factor of 2, by, for example, running the locality sensitive hashing algorithm of [18], which returns a $(1+\epsilon')$ factor approximate closest pair, for $\epsilon' = 1$.

Fact 24 (From [3])**.** *For any constant $\gamma > 0$, locality sensitive hashing can be used to approximate the minimum distance between a set of n vectors in d-dimensional Euclidean space, up to a multiplicative factor of 2, in time $O(dn^{5/4+\gamma})$.*

The following algorithm reduces the problem of finding a close pair of points among arbitrary vectors, to finding a close pair of points among unit vectors.

Algorithm 25. Standardize
Input: A $d \times n$ matrix X with entries $x_{i,j} \in \mathbb{R}$, constant $\epsilon \in (0,1)$.
Output: A $m' \times n$ matrix Y with all columns having unit norm, and $m' = \log^3 n$.

- Perform a Johnson-Lindenstrauss transformation of the columns of X into dimension m' to yield matrix X'.
- Let c denote the magnitude of the largest column of X'.
- Choose a random m'-dimensional vector v of length $8c/\epsilon$.
- Let matrix Y be the result of adding v to each column of X', and normalizing all columns so as to have unit norm.

Proposition 26. *Letting Y denote the result of algorithm* Standardize *on input X and ϵ, with probability $1 - o(1/poly(n))$ for all sets of four columns Y_1, Y_2, Y_3, Y_4 of matrix Y, with $X_1, \ldots, X_4$ being the corresponding columns of matrix X, it holds that*

$$\frac{||Y_1 - Y_2||}{||Y_3 - Y_4||} \cdot \frac{||X_3 - X_4||}{||X_1 - X_2||} \in [1-\epsilon, 1+\epsilon].$$

Proof. With probability at least $1 - o(1/poly(n))$, the Johnson-Lindenstrauss transformation will preserve all distances up to a multiplicative factor of $1 + \epsilon/8$, hence we proceed with the proof assuming this is the case. For any given vector $X'_i - X'_j$, the length of its projection onto the plane orthogonal to the vector $v + \frac{X'_i + X'_j}{2}$, will decrease by a factor of $o(1/\log n) < \epsilon/8$ with probability

$1-o(1/poly(n))$ over the choice of v. Note that the only difference between this projection of $X'_i - X'_j$ and $8cY_i/\epsilon - 8cY_j/\epsilon$ is due to the scaling factor to make the images of X'_i, X'_j unit vectors, and the discrepancy between this scaling factor and $8c/\epsilon$ is at most a factor of $\frac{2c+8c/\epsilon}{8c/\epsilon} = 1+\epsilon/4$. Thus by the triangle inequality with the claimed probability, the ratios of all pairs of pairwise distances are preserved, up to a multiplicative factor of ϵ. □

The following algorithm finds a $(1+\epsilon)$ approximate closest pair from among a set of unit vectors, given that the distance between the closest pair is at least $\alpha > \frac{1}{n^{0.9}}$.

Algorithm 27. Approximate Closest Pair (large α)
Input: A $d \times n$ matrix X with entries $x_{i,j} \in \mathbb{R}$ with $d = \log^3 n$ whose columns have unit norm, and constant $\epsilon \in (0,1)$, and $\alpha > 1/n^{0.9}$ such that the closest pair of columns have distance in the range $[\alpha, 2\alpha]$.
Output: A pair of indices.

- *If $\alpha > 0.2$ return the best result found in $\log^2 n$ runs of algorithm* Approximate Maximal Inner-Product *on input X, and $\epsilon/20$.*
- *Otherwise, define the $d \times n$ matrix Z as follows: for each $i \in [d]$, select a set of $q = \lfloor \frac{\pi}{2\alpha} \rfloor$ uniformly random unit vectors $v_1, \ldots, v_q$ and for all $j \in [n]$, set*

$$z_{i,j} = sign\left(\prod_{k=1}^{q} X_j^\dagger v_k\right),$$

  where X_j is the jth column of matrix X.
- *Return the best result found in $\log^2 n$ runs of algorithm* Approximate Maximal Inner-Product *with error parameter $\epsilon/20$ and input matrix Z with all entries scaled by $1/\sqrt{d}$ so as to make them have unit norm.*

The following proposition characterizes the performance of the above algorithm.

Proposition 28. *For any constant $\epsilon > 0$, Algorithm 31 (*Approximate Closest Pair (large α)*), when given as input n unit vectors in $\mathbb{R}^d$ with $d = \log^3 n$, whose closest pair have distance lying in the interval $[\alpha, 2\alpha]$, with probability $1 - o(1/poly(n))$ will output a pair with Euclidean distance at most a factor of $(1+\epsilon)$ larger than that of the minimal pair. Additionally, provided $\alpha > 1/n^{0.9}$, the runtime of the algorithm is $\tilde{O}(n^{2-\Theta(\sqrt{\epsilon})})$.*

Proof. In the case that $\alpha > 0.2$, the proposition follows from noting that provided the distance between the closest pair is bounded below by a constant, the task of returning the approximate closest pair to a multiplicative factor of $1+\epsilon$ is the same the task of returning the pair with approximately maximal inner product to additive $c\epsilon$ for some constant c. Specifically, given that the closest pair has distance at least 0.2 the law of cosines yields that the maximal inner product is at most $1-0.08+o(1)$ and hence an additive approximation of the maximal inner product to within $\epsilon/20$ will yield the desired multiplicative guarantee on the distance.

In the case that $\alpha < 0.2$, consider two columns X_i, X_j that form an angle of β (and hence have distance $\sqrt{(1-\cos\beta)^2 + \sin^2\beta}$). For each random vector v, we have that $\mathrm{E}[sign(X_i^\dagger v \cdot X_j^\dagger v)] = 1 - 2\frac{\beta}{\pi}$, and since expectations of independent random variables multiply, we have that for each k,

$$\mathrm{E}[z_{k,i} z_{k,j}] = \left(1 - \frac{2\beta}{\pi}\right)^q.$$

Consider the pair of columns of X with minimal distance $\delta^* \in [\alpha, 2\alpha]$, and hence form an angle $\beta^* \in [\alpha/2, 2\alpha]$, in which case the expected inner product between the corresponding columns of Z

is at most $d\left(1-\frac{\alpha}{\pi}\right)^{\pi/2\alpha} \leq 0.65d$, and is at least $d\left(1-\frac{4\alpha}{\pi}\right)^{\pi/2\alpha} \geq 0.11d$. By the same calculation, the image of any pair of columns of X whose distance was a multiplicative factor of at least $(1+\epsilon)$ larger than δ^* will have expected inner product at least a multiplicative factor of $(1+\epsilon/2)$ smaller than that of the image of the pair with distance δ^*. By a union bound over Chernoff bounds, since $d = log^3 n$, with probability $1-o(1/poly(n))$ the inner products between any two columns of Z differs from their expectations by $o(1)$, and hence finding the pair with inner product within an additive $\epsilon/20$ will correspond to finding a multiplicative $1+\epsilon$ approximation to the closest pair of original vectors.

To bound the runtime, note that provided $\alpha < 1/n^{0.9}$, the runtime of all components of the algorithm aside from the calls to Approximate Maximal Inner-Product, take time bounded by $\tilde{O}(n/\alpha) = \tilde{O}(n^{1.9})$. □

Finally, we address the setting in which the closest pair might be extremely close, having distance $< \frac{1}{n^{0.9}}$. Here, the difficulty is that we cannot compute a sufficiently high powered XOR/tensor embedding without spending super-quadratic time. Instead, we note that if the minimum distance is so small, we can recursively divide the set of points into small clusters, that are all far apart in relation to the minimal distance. If all clusters are small, then we can trivially find the closest pair by brute force search with each cluster. If we have a large cluster (with tiny diameter), then we can can simply subtract off the mean of the cluster; after re-normalizing via the Standardization algorithm, Algorithm 25, the resulting points will have unit norm, and the smallest distance will have increased to at least $1/n^{0.8}$, and we can apply the algorithm Approximate Closest Pair (large α).

Algorithm 29. Approximate Closest Pair (small α)
Input: A $d \times n$ matrix X with entries $x_{i,j} \in \mathbb{R}$ whose columns have unit norm, and constant $\epsilon \in (0,1)$, and $\alpha < 1/n^{0.9}$ such that the closest pair of columns have distance in the range $[\alpha, 2\alpha]$.
Output: A pair of indices $c_1, c_2 \in [n]$.

- *Let $v_1, \ldots, v_n$ denote the n columns of X. We recursively split up these vectors:*
  - *Project all vectors onto a random d dimensional unit vector and sort the resulting projections, $x_1 \leq \ldots \leq x_n$, assume wlog x_i is the proj. of v_i.*
  - *We now traverse the list: we ''pause'' at some x_i, if there are fewer than $n^{0.6}$ points with projected value in the interval $[x_i - 2\alpha, x_i]$. If we have ''paused'' at x_i we do one of two procedures:*
  - *if $|\{j : x_j < x_i\}| \leq n^{0.9}$:*
    - *Brute force search for the closest pair of points from the set $\{v_j : x_j < x_i\}$. Store the closest pair and their distance, and remove all points v_j (and their projections x_j) for which $x_j \leq x_i - 2\alpha$ from all further computations. Continue traversing the list (with those points removed).*
  - *if $|\{j : x_j < x_i\}| > n^{0.9}$:*
    - *Save set $S_i := \{v_j : x_j \leq x_i\}$, and continue traversing the list with all points v_j s.t. $x_j \leq x_i - 2\alpha$ removed.*
  - *Having finished traversing the list, if we have not stored any sets S_i, then we can simply compare the stored closest pair distances, and output the minimum. Otherwise, let $S_1, \ldots, S_k$ denote the sets that are stored. For each set S_i do the following:*
    - *For ease of notation, let S denote S_i. Note that points in S had projections x_i in sets of contiguous intervals of width 2α; each interval had $\geq n^{.6}$ points, hence all x_i are within $2\alpha n^{.4}$.*
    - *Choose $\sqrt{n}$ random pairs of vectors from S, and compute their distances. Let μ be the median of these $\sqrt{n}$ distances. If $\mu > \alpha n^{0.6}$, then output $ERROR$.*
    - *Otherwise, randomly select $v \in S$, sample $n^{0.1}$ distances between v and randomly chosen $v' \in S$; repeat until one has found a v that has distance at most $2\alpha n^{0.6}$ from at least $1/4$ of the points in S.*
    - *Let $0 = d_1 \leq \ldots \leq d_{|S|}$ be the distances beween v and all points in S. Find $c \in [2\alpha n^{0.6}, 4\alpha n^{0.6}]$ s.t. $|\{i : d_i \in [c, c + 2\alpha]\}| < n^{0.4}$.*
    - *For each of the at most $n^{0.4}$ points in S with $d_i \in [c, c + 2\alpha]$, via a brute force search check all $O(n^{1.4})$ distances between such points and all other points in S, recording the smallest such distance.*
    - *Construct the sets $T := \{v_i : d_i < c\}$, and $T' := \{v_i : d_i > c + 2\alpha\}$. We recurse the whole algorithm on the set of points T' noting that $|T'| < \frac{3}{4}|S|$.*
    - *We now address the set T: subtract v from all points in T and run algorithm* Standardize *on the resulting points, and then run algorithm* Approximate Closest Pair (large α) *with parameter $\epsilon/2$ on the results of running* Standardize *on the set T with parameter $\epsilon/2$. (Since all points in T were distance at most $4\alpha n^{0.6}$ from v, after subtracting off v, and running the* Standardize *algorithm, the distance of the closest pair will have increased by a factor of $\Omega(\frac{1}{\alpha n^{0.6}})$, and hence the minimum distance will be $>> 1/n^{0.9}$ and hence will be amenable to running the ''large α'' variant of the closest pair algorithm.)*
- *Compare the closest pairs returned by all branches of the algorithm, and return the closest.*

Proposition 30. *Algorithm 29 (*Approximate Closest Pair (small α)*), on input a set of n unit vectors in $d = \log^3 n$ dimensional space, $\epsilon > 0$, and $\alpha \leq 1/n^{0.9}$ such that the closest pair of points has distance in the interval $[\alpha, 2\alpha]$ will output the $(1+\epsilon)$ approximate closest pair with probability $1 - o(1/poly(n))$, and runs in time $\tilde{O}(n^{2-\Theta(\sqrt{\epsilon})})$.*

Proof. Each time that the algorithm "pauses" at a projected value x_i, if no set is saved during that pause, then a brute-force-search is performed on at most $n^{0.9}$ vectors, which are then removed from all subsequent computation. If the closest pair involves one of those points, then we will find it, since our projection onto a random unit vector only decreases distances.

If a set S is "saved", then the vectors correspond to a set of vectors that ended up unusually close together in the projection. In particular, note that the projection onto a random unit vector, in expectation, shrinks distances by a factor of $\Theta(1/\sqrt{d})$, and, crudely, the probability that a given distance is decreased by a factor of more than $100/\sqrt{d}$ is bounded by $1/10$, and hence the probability that the median of a subset of $\sqrt{n}$ randomly chosen pairwise distances from points in set S is more than $\alpha n^{0.6}$ is $o(1/poly(n))$, since, based on the proximity of the projections of these points, the distances should be $\tilde{(}\alpha n^{0.4})$. Given this, with probability $1 - o(1/poly(n))$ the sampling approach will find a vector v that has the desired distance from at least $1/4$ the points of S, and the sets T, T' we be constructed as desired and have the claimed size. Note that the step that chooses the threshold distance c between the elements of T and T', and then brute-force searches, and discards all points within this critical distance $[c, c + 2\alpha]$ ensures that if the closest pair of points lies in S, then the pair was either found in the brute-force phase, or is contained within either T or T'.

The crux of the algorithm is the fact that we subtract the vector v from all points in set T. Since, by construction, all points in this set have distance at most $4\alpha n^{0.6}$ from v, after v is subtracted from all the points, the ratio of the minimum distance in set $T - v$ to the maximum modulus of these points is now at least $\frac{\alpha}{4\alpha n^{0.6}} = n^{-0.6}/4$. Hence by Proposition 26, the result of running STANDARDIZE on this set will yield a set of unit vectors for which the ratio of the distances have been distorted by at most a factor of $(1 + \epsilon/2)$, and the minimum distance is at least $\Theta(n^{-0.6}/\epsilon) >> 1/n^{0.9}$.

To analyze the runtime of the algorithm, note that all branches of the algorithm terminate, with the exception of the recursive call to the sets T'. Nevertheless, with each call the number of points in each set considered decreases by a factor of at least $3/4$, since $|T'| \leq \frac{3}{4}|S|$, and the total number of points in the union of all such sets is at most $n(1 + o(1))$, from the fact that in the initial creation of sets S_i, at most a $n^{0.6}/n^{0.9}$ fraction of points end up being assigned to more than one of the sets S_i. □

For completeness, we state the general closest-pair algorithm:

Algorithm 31. APPROXIMATE CLOSEST PAIR
Input: *A $d \times n$ matrix X with entries $x_{i,j} \in \mathbb{R}$, constant $\epsilon \in (0, 1)$.*
Output: *A pair of indices.*

- *Let Y be the $m \times n$ matrix output by algorithm* STANDARDIZE *on input X with parameter $\epsilon/4$.*
- *Using locality sensitive hashing, find α such that the minimum distance between pairs of columns of Y is in the interval $[\alpha, 2\alpha]$.*
- *If $\alpha \geq 1/n^{0.9}$ return the results of running* APPROXIMATE CLOSEST PAIR (LARGE α) *on input Y, $\epsilon/2$, and α.*
- *If $\alpha < 1/n^{0.9}$ return the results of running* APPROXIMATE CLOSEST PAIR (SMALL α) *on input Y, $\epsilon/2$, and α.*

Theorem 2, which we restate below for convenience, characterizes the performance of the above algorithm, and follows immediately from Fact 24, and Propositions 28 and 30:

Theorem 2 *Given n vectors in $\mathbb{R}^d$ and sufficiently small approximation parameter $\epsilon > 0$, with probability $1 - o(1)$ our algorithm will return a pair of vectors u, v such $||u - v|| \leq (1 + \epsilon)d^*$, where*

d^* is the (Euclidean) distance between the closest pair of vectors. Additionally, the algorithm runs in time

$$O\left(n^{2-\Omega(\sqrt{\epsilon})} + nd\right).$$

5 Learning Parities and Juntas

The problem of finding a ρ-correlated pair of Boolean vectors from among n random vectors is easily seen to be equivalent to solving the parity with noise problem, in the special case that the size of the true parity set is $k = 2$; the correspondence between the correlation ρ and noise rate η is given by $\eta = 1/2 - \rho/2$. To see one direction of the equivalence, note that given an instance of such a parity with noise problem, if one removes all examples that have label 1, one will be left with a set of examples in which the two true parity indices are correlated. One could thus use the algorithm of Proposition 12 to find the pair of parity indices in time $n^{\frac{5-\omega}{2(4-\omega)}k} poly(\frac{1}{1/2-\eta}) < n^{1.62} poly(\frac{1}{1/2-\eta})$, where $\omega < 2.4$ is the exponent of matrix multiplication.

In general, given an algorithm for solving the parity with noise problem for parities of some fixed size c in time $O(n^\alpha)$, one may attempt to adapt it to obtain an algorithm for the parity with noise problem for parities of any value $k > c$ that runs in time $O(n^{k\frac{\alpha}{c}})$ by performing the following transformation: for each length n example with label ℓ, transform it into a length $N = \binom{n}{k/c} \approx n^{k/c}$ example with label ℓ, where each index represents the XOR (or product in the ± 1 setting) of some set of k/c of the indices of the original example. If the original set of examples contained a set of k indices whose XOR is correlated with the labels, then the transformed examples will contain (several) sets of c indices whose XOR is correlated with the labels. One can now simply apply the original algorithm for finding parities of size c to the transformed set of examples, to yield a runtime of $O\left((n^{k/c})^\alpha\right)$. The minor difficulty, of course, is that the transformed examples are no longer uniformly random bit strings, though most algorithms should be robust to the type of dependencies that are introduced by this transformation.

The above transformation motivates the search for improved algorithms for finding small constant–sized parities ($k = 2, 3, 4, \ldots$). Given the existence of a subquadratic time algorithm for the case $k = 2$, a natural hope is that one can design better and better algorithms for larger k, perhaps with the eventual hope of yielding an $n^{o(k)}$ algorithm.

While the above reduction together with Proposition 12 (corresponding to $k = 2$) yields an algorithm for learning k-sparse parities with runtime

$$n^{\frac{5-\omega}{2(4-\omega)}k} poly(\frac{1}{1/2-\eta}) < n^{0.81k} poly(\frac{1}{1/2-\eta}),$$

we instead consider the $k = 3$ case directly, and obtain an exponent of $< 0.80k$. While the constant in the exponent is only very slightly better than what is yielded from leveraging the $k = 2$ setting, this alternate approach is intuitively appealing and may be of independent interest.

5.1 A Little Bias Goes a Long Way

As for the results of Sections 3 and 4, our $k = 3$ algorithm uses fast matrix multiplication to find a pair of correlated vectors. The crux of the approach is that a parity function has reasonably heavy low-degree Fourier coefficients if one changes from the uniform distribution over the Boolean hypercube to a slightly biased product distribution. The required bias is very small, thereby allowing one to

efficiently subsample a set of uniformly random examples so as to simulate a set of examples with the desired bias. In the remainder of this section we describe the main idea behind the algorithm.

Given an example x, y, for $x = (x_1, \ldots, x_n)$ from an $(n, 3, \eta)$-instance of parity with noise (with three parity bits), for any i, $\Pr[x_i = 1 | y = 1] = 1/2$. Similarly, $\Pr[x_i x_j = 1 | y = 1] = 1/2$ for distinct $i, j \in [n]$. The improved algorithm for finding parities rests on the following observation about parity sets of size $k = 3$: if the bits of x are not chosen uniformly at random, but instead are chosen independently to be 1 with probability $\frac{1}{2} + \alpha$, for some small bias α, then the above situation no longer holds. In such a setting, it is still the case that $\Pr[x_i = 1 | y = 1] \approx \frac{1}{2} + \alpha$, however

$$\Pr[x_i x_j = 1 | y = 1] = \begin{cases} \frac{1}{2} + \Theta(\alpha) & \text{if } i \text{ and } j \text{ are both in the true parity set,} \\ \frac{1}{2} + \Theta(\alpha^2) & \text{if } i \text{ or } j \text{ is not in the set of parity bits.} \end{cases}$$

The punchline of the above discrepancy is that very small biases—even a bias of $\alpha = 1/\sqrt{n}$ can be quite helpful. Given such a bias, for any pair $i, j \in [n]$, for sufficiently large n, even $n \cdot \text{polylog}(n)$ examples will be sufficient to determine whether i and j are both in the parity set by simply measuring the correlation between the ith and jth indices for examples with even label, namely estimating $\Pr[x_i x_j = 1 | y = 1]$ based on the examples. How does one compute these $\binom{n}{2}$ correlations between vectors of length $n\text{polylog}(n)$ in time $o(n^3)$? By (fast) matrix multiplication. It is worth stressing that, provided this argument is sound, the resulting algorithm will be extremely noise-robust, since the discrepancy between $\Pr[x_i x_j = 1 | y = 1]$ in the cases that i, j are both parity bits and the case that they are not will degrade linearly as $\eta \to 1/2$.

It should now be intuitively clear how to extend this approach from the small-biased setting to the setting in which the examples are generated uniformly at random, since a bias of $1/\sqrt{n}$ is quite modest. In particular, with constant probability, a random length–n example will have at least $\frac{n}{2} + \sqrt{n}$ positive indices, thus simply filtering the examples by removing those with fewer than $n/2$ positive indices should be sufficient to instill the necessary bias (at the minor expense of independence). In the following sections we make this approach rigorous.

5.2 Learning Parity by Adding Bias

As in the case of learning a parity of size $k = 3$, outlined in the previous section, for the general case of parities of size k a bias of $1/\sqrt{n}$ in the examples will be sufficient. There are many approaches to achieving this bias; algorithmically, the most simple approach is to take examples, and reject those which have fewer than $\frac{n}{2} + \sqrt{n}$ positive indices. While this approach can be made to work, the conditioning on the total weight being large greatly complicates the analysis. Thus we instead argue that one can filter the examples in such a way that the distribution of the examples that remain is very close in total variational distance (ℓ_1 distance) to the distribution in which the examples are actually generated by independently choosing the value of each index with probability $\frac{1}{2} + \frac{1}{\sqrt{n}}$. Thus the result of applying our algorithm to the filtered examples will, with high probability be identical to the result of applying the algorithm to a set of examples generated according to the idealized process which selects the value of each index of each example independently, to be 1 with probability $1/2 + 1/\sqrt{n}$, and thus it suffices to perform the analysis of the simpler setting in which indices are chosen independently.

We first state the simple filtering process, and then prove that the resulting distribution of examples has the desired property. Throughout, we let $Bin[r, p]$ denote the binomial random variable representing the number of heads that occur after flipping r i.i.d. coins that each land heads with probability p.

Algorithm 32. MAKE BIASED EXAMPLES
Input: An $m \times n$ matrix X with entries $x_{i,j} \in \{-1, +1\}$, a desired bias $\alpha \in (0, \frac{1}{2})$, and $t \in [n]$.
Output: an $m' \times n$ matrix Y, for some $m' \leq m$, consisting of a subset of the rows of X.

- Define $r = \frac{\Pr[Bin[n, \frac{1}{2}] > t]}{\Pr[Bin[n, \frac{1}{2}+\alpha] > t]}$.
- For each row $x_i = x_{i,1}, \ldots, x_{i,n}$ of X:
  - let s_i be the number of 1's in x_i.
  - If $s_i \geq t$ discard row x_i.
  - Otherwise, if $s_i < t$, then include row x_i in matrix Y with probability

$$r \cdot \frac{\Pr[Bin[n, \frac{1}{2} + \alpha] = s_i]}{\Pr[Bin[n, \frac{1}{2}] = s_i]}$$ (Note that this quantity is always bounded by 1.)

Proposition 33. *The algorithm* MAKE BIASED EXAMPLES *when given as input an $m \times n$ matrix X chosen with each entry being 1 independently with probability $1/2$, $\alpha = \frac{1}{\sqrt{n}}$, and $t = \frac{n}{2} + \sqrt{n}\frac{k \log n}{4}$, will output matrix Y satisfying the two following properties:*

- *The total variation distance between the distribution from which each row of Y is chosen and the distribution on rows defined by the process of picking each of the n elements independently to be 1 with probability $\frac{1}{2} + \alpha$, is at most $n^{-\Theta(k^2 \log n)}$*
- *With probability at least $1 - e^{-\frac{mr^2}{32}}$, Y has at least $\frac{mr}{4}$ rows, where $r \geq n^{-(\frac{1}{2} + \frac{k}{4})}$.*

The following lemma will be useful in the proof of the above proposition.

Lemma 34. *For $Bin[n, p]$ denoting a binomially distributed random variable, for $\alpha > 0$ with $\alpha = o(1)$ and $s > \sqrt{n} + \alpha n$,*

$$\frac{\Pr[Bin[n, \frac{1}{2}] > \frac{n}{2} + s]}{\Pr[Bin[n, \frac{1}{2} + \alpha] > \frac{n}{2} + s]} \geq \frac{1}{(1 + 2\alpha)^{2s}\sqrt{n}},$$

for sufficiently large n.

Proof. We first lowerbound the numerator; trivially, $\Pr[Bin[n, \frac{1}{2}] > \frac{n}{2} + s] > \Pr[Bin[n, \frac{1}{2}] = \frac{n}{2} + s] = \binom{n}{\frac{n}{2}+s}\frac{1}{2^n}$. We now upper bound the denominator. To this order, note that for any $s' \geq s$, we have

$$\begin{aligned}
\frac{\Pr[Bin[n, \frac{1}{2} + \alpha] = \frac{n}{2} + s' + 1]}{\Pr[Bin[n, \frac{1}{2} + \alpha] = \frac{n}{2} + s']} &= \frac{\binom{n}{\frac{n}{2}+s'+1}(\frac{1}{2} + \alpha)}{\binom{n}{\frac{n}{2}+s'}(\frac{1}{2} - \alpha)} \\
&= \frac{n - 2s'}{2 + n + 2s'} \cdot \frac{\frac{1}{2} + \alpha}{\frac{1}{2} - \alpha} \\
&\leq \frac{n - 2(\sqrt{n} + \alpha n)}{2 + n + 2(\sqrt{n} + \alpha n)} \cdot \frac{\frac{1}{2} + \alpha}{\frac{1}{2} - \alpha} \\
&= 1 - \frac{4\sqrt{n} - 4\alpha + 2}{(n + 2\sqrt{n} + 2\alpha n + 2)(1 - 2\alpha)} \leq 1 - \frac{1}{\sqrt{n}},
\end{aligned}$$

for sufficiently large n. This shows that we may bound $\sum_{i=\frac{n}{2}+s}^{n} \binom{n}{i}(1/2 - \alpha)^i(1/2 + \alpha)^{n-i}$ by the geometric series

$$\binom{n}{\frac{n}{2} + s}(1/2 - \alpha)^{\frac{n}{2}-s}(1/2 + \alpha)^{\frac{n}{2}+s}\sum_{i=0}^{\infty}(1 - \frac{1}{\sqrt{n}})^i = \binom{n}{\frac{n}{2} + s}(1/2 - \alpha)^{\frac{n}{2}-s}(1/2 + \alpha)^{\frac{n}{2}+s}\sqrt{n}.$$

Thus the desired ratio is at least

$$\frac{\binom{n}{\frac{n}{2}+s}\frac{1}{2^n}}{\binom{n}{\frac{n}{2}+s}(1/2-\alpha)^{\frac{n}{2}-s}(1/2+\alpha)^{\frac{n}{2}+s}\sqrt{n}} \geq \frac{1}{(1+2\alpha)^{2s}\sqrt{n}}.$$

□

Proof of Proposition 33. Each row of Y is distributed as a length-n string with each bit equaling 1 independently with probability α, conditioned on the total number of 1's to be at most t. This distribution has variation distance at most $2\Pr[Bin[n, \frac{1}{2}+\alpha] > t]$ from the corresponding distribution in which no conditioning occurs. By standard Chernoff bounds, $\Pr[Bin[n, \frac{1}{2}+\alpha] > t] \leq e^{-\frac{(n(\frac{1}{2}+\alpha)-t)^2}{(1-2\alpha)n}}$.

The expected number of rows of Y will be at least $m \cdot r \cdot (1-q)$, where $q = \Pr[Bin[n, \frac{1}{2}+\alpha] > t]$, and thus this expectation is trivially at least $\frac{mr}{2}$. Since each row of the input X is inserted into Y independently, with probability at least $\frac{r}{2}$, by a Chernoff bound, $\Pr[|Y| < \frac{mr}{4}] < e^{-\frac{mr^2}{32}}$. Using the lower bound on r of Lemma 34 yields the claim. □

We now state the general algorithm for learning parities of size k. Note that throughout, we assume that we know the size of the true parity set. This is without loss of generality, as we can always simply try $k = 1, 2, 3, \ldots$, and lose at most a factor of k in our runtime. Additionally, we aim to find the parity with some constant probability. Since we can always verify whether the returned parity set is correct (with all but inverse exponential probability), by simply repeating the algorithm many times this constant probability of success can become probability $1-\delta$ at an extra multiplicative expense of $\log \frac{1}{\delta}$. Finally, we assume that k is divisible by 3. This is without loss of generality, as we can always insert up to two extra bits in each example and multiply the label by their values so as to yield examples from an instance of size at most $n+2$ where the size of the parity is actually divisible by 3.

Algorithm 35. Learn Parity with Noise

Input: An $m \times n$ matrix X with entries $x_{i,j} \in \{-1, +1\}$, a length m vector $v \in \{-1, +1\}^m$ of labels, a parameter k that is divisible by 3, and an arbitrarily small constant $\epsilon > 0$.

Output: a set of k indices.

- Let Y be the result of running MAKE BIASED EXAMPLES on matrix X, with $\alpha = \frac{1}{\sqrt{n}}$ and $t = \frac{n}{2} + \frac{k \log n}{4}\sqrt{n}$.
- Remove all rows from Y whose corresponding label (in vector v) is -1, and take the first $m' = \frac{n^{k/3(1+\epsilon)}}{(1-2\eta)^{2+\epsilon}}$ of the remaining rows to form the $m' \times n$ matrix Y'.
- Generate the $m' \times \binom{n}{\frac{k}{3}}$ matrix Z by taking each row of Y', and generating a row of Z of length $\binom{n}{k/3}$, with each position $z_{i,S}$ corresponding to a set $S \subset [n]$ of $k/3$ distinct indices, and setting $z_{i,S} = \prod_{j \in S} y'_{i,j}$.
- Compute the $\binom{n}{k/3} \times \binom{n}{k/3}$ matrix $C = Z^t Z$. For convenience, we regard the elements of C as being indexed by a pair of sets $S, S' \subset n$ with $|S| = k/3$, thus $c_{S,S'}$ is the entry corresponding to the product of the two columns of Z corresponding to the sets S and S'.
- For every pair of subsets $S, S' \subset [n]$ with S and S' each consisting of $k/3$ distinct elements, if $S \cap S' \neq \emptyset$, set $c_{S,S'} = 0$.
- Let c_{S_1,S_2}, be the largest element of matrix C. Look in the column C_{S_1} corresponding to set S_1, and let S_3 be the set that maximizes c_{S_1,S_3} subject to the constraint that $S_3 \cap S_2 = \emptyset$. Output $S_1 \cup S_2 \cup S_3$.

Theorem 5. *For any fixed $\epsilon > 0$, for sufficiently large n and k given $m = \frac{n^{\frac{2k}{3}(1+\epsilon)}}{(1-2\eta)^{2+\epsilon}}$ examples from an (n, k, η) instance of parity with noise, with probability $1 - o(1/n)$, the algorithm* LEARN PARITY WITH NOISE*, when given as input the $m \times n$ matrix of examples, and length m vector of labels, will correctly return the true set of k parity bits. Additionally, the algorithm will run in time*

$$O\left(\left(\frac{n^{\frac{k}{3}(1+\epsilon)}}{(1-2\eta)^{2+\epsilon}}\right)^{\omega}\right).$$

The crux of the proof of Theorem 5 is that, given that applying MAKE BIASED EXAMPLES to matrix X yields a matrix Y with suitably biased elements, the matrix Z inherits some bias from Y. In particular, the fact that each entry of Z is given as the product of $k/3$ entries of X should not completely erase the bias. While the bias will decrease exponentially in k, the length of the rows of Z are correspondingly larger, and we are only hoping that the bias of each element of Z is roughly $1/\sqrt{|Z|}$. The following basic lemma guarantees this.

Lemma 36. *Let $z = \prod_{i=1}^{s} w_i$, where each $w_i \in \{-1, +1\}$ is chosen independently to be 1 with probability $\frac{1}{2} + \alpha$. Then $\Pr[z = 1] = \frac{1}{2} + 2^{s-1}\alpha^s$.*

Proof. Letting $p = \frac{1}{2} - \alpha$, we have the following:

$$\begin{aligned}\Pr[z = 1] - \Pr[z = -1] &= \sum_{i=0}^{s}(-1)^i p^i (1-p)^{s-i}\binom{s}{i} \\ &= ((1-p) - p)^s \\ &= (1-2p)^s = 2^s\alpha^s,\end{aligned}$$

□

Proof of Theorem 5. Proposition 33 guarantees that with probability at least $1 - o(1/n)$ there will be at least $m' = \frac{n^{\frac{k}{3}(1+\epsilon)}}{(1-2\eta)^{2+\epsilon}}$ examples with which to populate matrix Y. Additionally, Proposition 33 guarantees that the total variational distance between the distribution from which matrix Y is drawn and the distribution defined by choosing each index independently to be 1 with probability $\frac{1}{2} + \frac{1}{\sqrt{n}}$ is $o(1/n)$, hence with the claimed probability the algorithm will perform identically as in the case that the elements of matrix Y were actually generated according to this independent model. For the remainder of the proof, we argue as if matrix Y is generated in that fashion.

We now consider the matrix Z. Let $z_S, z_{S'}$ be two element of the row z of Z corresponding to disjoint sets $S, S' \subset [n]$, and let ℓ denote the label corresponding to row z before any noise has been added. Let $w = \prod_{j \in S \cup S'} y_j$, denote the random variable representing $z_S z_{S'}$. For notational convenience, define

$$F(\beta, h) = \sum_{i=0}^{\lfloor h/2 \rfloor}(\frac{1}{2} - \beta)^{2i}(\frac{1}{2} + \beta)^{h-2i}\binom{h}{2i} = \frac{1}{2}\left(1 + 2^h\beta^h\right),$$

which is the probability computed in Lemma 36 that when h identical independent coins that land heads with probability $\frac{1}{2} + \beta$ are tossed, an even number of heads occurs. Letting s denotes the

number of parity bits in $S \cup S'$, we have the following, where $\alpha = \frac{1}{\sqrt{n}}$:

$$\begin{aligned}\Pr[w = 1|\ell = 1] &= \frac{F(\alpha, s)F(\alpha, \frac{2k}{3} - s)F(\alpha, k - s)}{F(\alpha, k)} \\ &\quad + \frac{(1 - F(\alpha, s))(1 - F(\alpha, \frac{2k}{3} - s))(1 - F(\alpha, k - s))}{F(\alpha, k)} \\ &= \frac{1 + (2\alpha)^{2k/3} + (2\alpha)^k + (2\alpha)^{5k/3-2s}}{2(1 + (2\alpha)^k)},\end{aligned}$$

where the numerator of first line is computing the probability that $w = 1$ and $\ell = 1$.

In the case that $s = 2k/3$, which occurs when both S and S' are subsets of the set of parity bits, then we can lowerbound the above as

$$\Pr[w = 1|\ell = 1] \geq \frac{1 + (2\alpha)^{k/3}}{2(1 + (2\alpha)^k)} \geq \frac{1}{2} + \frac{(2\alpha)^{k/3}}{2} - (2\alpha)^{2k/3} \geq \frac{1}{2} + \frac{(2\alpha)^{k/3}}{3},$$

since $\alpha = o(1)$. In the case that $s \leq 2k/3 - 1$, we upperbound the quantity as follows:

$$\Pr[w = 1|\ell = 1] \leq \frac{1 + 2(2\alpha)^{k/3+2}}{2} \leq \frac{1}{2} + \frac{(2\alpha)^{k/3}}{100},$$

since $\alpha = o(1)$. Letting ℓ^* denote the true label, corrupted with independent noise $\eta < 1/2$, we have

$$\Pr_{s=2k/3}[w = 1|\ell^* = 1] \geq \frac{1}{2} + \frac{(2\alpha)^{k/3}}{3}(1 - 2\eta), \text{ and } \Pr_{s\leq 2k/3-1}[w = 1|\ell^* = 1] \leq \frac{1}{2} + \frac{(2\alpha)^{k/3}}{100}(1 - 2\eta).$$

Putting the pieces together, letting m' denote the number of rows of matrix Z, which is at least $\frac{n^{\frac{k}{3}(1+\epsilon)}}{(1-2\eta)^{2+\epsilon}}$, we have that for any entry $c_{S,S'}$ of matrix C corresponding to two disjoint sets S, S', where S and S' are not *both* subsets of the parity bits, $E[c_{S,S'}] \leq 2\frac{m'\left(2/\sqrt{n}\right)^{k/3}}{100}(1 - 2\eta)$. On the other hand, if S, S' are both subsets of the parity bits, then $E[c_{S,S'}] \geq 2\frac{m'\left(2/\sqrt{n}\right)^{k/3}}{3}(1 - 2\eta)$, and since these quantities have variance at most m', for any constant ϵ, via a union bound over Chernoff bounds, taking n large yields that with probability $1 - o(1/n)$, all the entries of C corresponding to pairs of disjoint sets that are not both subsets of the true parity bits will be smaller than all the entries that correspond to pairs of subsets of the true parity bits. □

5.3 Reducing the Sample Complexity

In order to obtain our desired corollary for learning DNF (Corollary 9), we must reduce the number of examples used in our Learn Parity with Noise algorithm. Intuitively, provided one has a very noise–robust algorithm, such reduction in sample complexity is easy; one simply takes a very small number of examples—in fact, $\min(n^{1+\epsilon}, poly(k) \log n)$ will suffice—and then "manufactures" many examples by XORing together small sets of the actual examples. Provided the initial noise in the labels is η, if we XOR together q examples, then the XOR of the labels will be the correct label with probability at least $\frac{1}{2} + \frac{(1-2\eta)^q}{2}$.

Algorithm 37. Make More Examples

Input: An $m \times n$ matrix X with entries $x_{i,j} \in \{-1, +1\}$, a length m vector $v \in \{-1, +1\}^m$ of labels, a positive integer $q < m$, an integer m'.

Output: An $m' \times n$ matrix Y, and a length m' vector w of labels.

- For each $i \in [m']$, randomly choose a set $T \subset [m]$, with $|T| = q$, create row y_i of Y, by assigning the jth component of y_i to be $\prod_{\ell \in T} x_{\ell,j}$, and letting the ith label be $\prod_{j \in T} v_j$.

Ideally, we would be able to apply the algorithm Learn Parity With Noise in a black-box fashion to the output of running make more examples on a small number of actual examples, as was done in [21]. Unfortunately, because the noise in the generated examples will increase with q in the exponent, we will not be able to take sufficiently large q so as to yield the necessary claim that the distribution of resulting examples is close in total variation distance to the desired uniform distribution.

Instead, we argue that the distribution of a small number (namely, k) of the columns of the generated examples are close to uniform. The idea is that we will argue that the distribution of the values in the k parity columns are close to uniform, which will let us apply our Chernoff bound to argue that with very high probability, the "good" entries $c_{S,S'}$ of the matrix C generated in Learn Parity With Noise, corresponding to S, S' subsets of the true parity set, will be "large". For all the "bad" entries of C, we will not be able to apply Chernoff bounds; however, using the fact that the rows are pairwise independent, we will apply Chebyshev's inequality to argue that with probability at least $1/2$, each "bad" element will be small. Thus after running the whole algorithm $\log(\binom{n}{k/3})$ times, we can argue that with high probability, in *every* run, the "good" coordinates will be large, whereas for a "bad" element, in each run of the algorithm, it will be small with probability at least $1/2$. Thus after $\log(\binom{n}{k/3})$ runs, with high probability the only elements that were never "small" will correspond to entries whose row and column correspond to subsets of the true parity set, as desired. We now make this roadmap rigorous. We begin by defining what it means for a family of hash functions to be *universal*, and state the Leftover Hash Lemma.

Definition 38. *Let $\mathcal{H}$ be a family of hash functions from A to B, and let $H \in \mathcal{H}$ be chosen uniformly at random. $\mathcal{H}$ is a* universal *family of hash functions if for all distinct $a, a' \in A$, $\Pr[H(a) = H(a')] \leq \frac{1}{|B|}$.*

Lemma 39 (Leftover Hash Lemma [17])**.** *For $A \subset \{0,1\}^m$, with $|A| \geq 2^r$, and $|B| = \{-1,+1\}^{r-\ell}$ for some $\ell > 0$, if $\mathcal{H}$ is a universal family of hash functions from A to B, then with probability at least $1 - 2^{-\ell/4}$, a uniformly random $H \in \mathcal{H}$ will satisfy $D_{tv}[H(a), Unif(B)] \leq 2^{-\ell/4}$, where a is a random element of A, $Unif(B)$ denotes the uniform distribution on B, and D_{tv} is the total variation distance.*

The following basic fact will also be useful.

Fact 40. *Given a vector $v \in \{-1,+1\}^m$, such that $m(\frac{1}{2}+p)$ indices of v are $+1$, then for a random set $T \subset [m]$, with $|T| = q$,*

$$\Pr[\prod_{i \in T} v_i = 1] \geq \frac{1}{2}\left(1 + \left(\frac{2mp - q + 1}{m - q + 1}\right)^q\right).$$

Proposition 41. *Given an $m \times n$ matrix X and vector of labels v consisting of m examples from an instance of parity with noise with noise rate η, integer $q \leq \frac{m(1-2\eta)}{4}$, and integer m', for any fixed set $S \subset [n]$ with $|S| = k$, with probability at least $1 - 2^{-\frac{q \log \frac{m}{q} - k}{4}}$, the algorithm* make more examples *on input X, v, q, m', will output a matrix Y such that the $m' \times k$ submatrix Y_S defined as the subset of the columns of Y corresponding to indices in S, will have total variation distance at most $m' 2^{-\frac{q \log \frac{m}{q} - k}{4}}$ from the distribution on matrices given by assigning each element to be ± 1 independently with probability $1/2$.*

Additionally, with probability at least $1 - 2^{-\frac{(q-1) \log \frac{m}{q-1} - k}{4}}$, the distribution of the rows of Y_S corresponding to the set of correct labels, will differ from that corresponding to the set of incorrect

labels by total variation distance at most $2m'2^{-\frac{(q-1)\log\frac{m}{q-1}-k}{4}}$. *Finally, provided* $1-2\eta > 4m^{-0.4}$, *with probability at least* $1-o(1/m)$, *the number of correct labels will be at least* $m'\left(\frac{1}{2}+\frac{1}{2}\left(\frac{1-2\eta}{4}\right)^q\right)-m'^{0.6}$.

Proof. We will first apply the Leftover Hash Lemma (Lemma 39). Note that each choice of matrix X defines a hash function from the set $A := \{T : T \subset [m], |T| = q\}$ to the set $B = \{-1,+1\}^k$, via the mapping that considers the columns of X corresponding to indices in set S, and XORs each coordinate of the rows of X with indices in set T (as described in the algorithm MAKE MORE EXAMPLES). Trivially, this family of hash functions is universal, since for two sets $T \neq T'$, supposing that $i \in T$, $i \notin T'$, the image of T and T' will differ XORing with a uniformly random string (namely, the ith row of X). Next, note that $|A| = \binom{m}{q} \geq 2^{q\log\frac{m}{q}}$ and thus Lemma 39 implies that with probability at least $1-2^{-\frac{q\log\frac{m}{q}-k}{4}}$ over the choice of matrix X, we will have that the distance between each row of Y and the uniform distribution over $\{-1,+1\}^k$ is at most $2^{-\frac{q\log m-2k}{8}}$. A union bound over our m' rows yields the desired claim.

We now reason about labels. With probability at least $1-o(1/m)$, the number of correct labels in the original vector v of labels will be at least $\frac{m}{2}+\frac{m(1-2\eta)}{2}-m^{0.6} > \frac{m}{2}+\frac{m(1-2\eta)}{4}$. Thus by Fact 40, with this probability the expected number of correct labels in vector w will be at least

$$m'\left(\frac{1}{2}\left(1+\left(\frac{m(1-2\eta)/2-q+1}{m-q+1}\right)^q\right)\right) \geq m'\left(\frac{1}{2}+\frac{1}{2}\left(\frac{1-2\eta}{4}\right)^q\right),$$

and thus with probability at least $1-o(1/m)$ over the initial choice of the v labels, and the choice of the sets that generate the m' new examples, at least $m'\left(\frac{1}{2}+\frac{1}{2}\left(\frac{1-2\eta}{4}\right)^q\right)-m'^{0.6}$ of the labels w will be correct.

We now argue that for a given "manufactured" example, the correctness of the label is essentially independent of the values of the chosen set of k indices. We proceed as in [21], and note that, assuming there is at least one incorrectly labelled example in v (if not, then the independence is trivial), letting X_{odd}, X_{even} denote the sets of subsets $T \subset [m]$ with $|T| = q$ for which the number of corresponding label is incorrect (correct). With probability $1-o(1/m)$, $|X_{even}| > |X_{odd}| > \binom{m}{q-1}$, and thus (since the correctness of the original labels are chosen independently of the corresponding example) we may apply Lemma 39 as above, to conclude that the distribution of the values of the k bits is distributed nearly uniformly over the 2^k values. In particular, with probability at least $1-2^{-\frac{(q-1)\log\frac{m}{q-1}-k}{2}}$, the distribution of the k bits conditioned on the label being correct will differ from the distribution conditioned on the label being incorrect by at most total variation distance $1-2^{-\frac{(q-1)\log\frac{m}{q-1}-k}{4}}$. □

We now describe our algorithm for solving instances of parity with noise, that uses few examples.

Algorithm 42. LEARN WITH FEW EXAMPLES

Input: *Positive integers* k, r, q, m' *an* $r \cdot m \times n$ *matrix* X *with entries* $x_{i,j} \in \{-1, +1\}$, *and a length* $r \cdot m$ *vector* $v \in \{-1, +1\}^m$ *of labels.*

Output: *A set* $S \subset [n]$ *with* $|S| = k$.

- *For* $i = 1$ *to* r
  - *Let matrix* X', *and labels* v' *be the output of running algorithm* MAKE MORE EXAMPLES *on input* X^i, v^i, q, m', *where* X^i *is the* $m \times n$ *submatrix of* X *consisting of rows* $i \cdot m + 1$ *through rows* $(i+1)m$, *and* v^i *is the corresponding vector of labels.*
  - *Let matrix* Y *be the result of running* MAKE BIASED EXAMPLES *on matrix* X *with* $\alpha = \frac{1}{\sqrt{n}}$ *and* $t = \frac{n}{2} + \frac{k \log n}{4}\sqrt{n}$.
  - *Remove all rows of* Y *with labels* -1, *and denote the resulting smaller* $m'' \times n$ *matrix* Y'.
  - *Generate the* $m'' \times \binom{n}{k/3}$ *matrix* Z *by taking each row of* Y', *and generating a row of length* $\binom{n}{k/3}$ *with each position* $z_{\ell,S}$ *corresponding to a set* $S \subset [n]$ *of* $k/3$ *(distinct) indices, and setting* $z_{\ell,S} = \prod_{j \in S} y'_{\ell,j}$.
  - *Compute the* $\binom{n}{k/3} \times \binom{n}{k/3}$ *matrix* $C^i = Z^t Z$, *and let* $m^i := m''$.
- *Let the set* $ParityBits$ *be the union of all pairs of disjoint sets of size* $k/3$, S, S', *that have the property that* $c^i_{S,S'} > \frac{m^i (2/\sqrt{n})^{k/3}}{3}$ *for each* $i \in [r]$, *where* $c^i_{S,S'}$ *denotes the index of matrix* C^i *indexed by the sets* S, S', *as in algorithm* LEARN PARITY WITH NOISE.
- *If* $|ParityBits| \neq k$ *output* $FAIL$, *otherwise output the set* $ParityBits$.

Theorem 6. *The algorithm* LEARN WITH FEW EXAMPLES, *when run on input* k, $r := 100k \log n$, q, $m' := \frac{n^{\frac{2k}{3}(1+\epsilon)}}{(1-2\eta)^{3q}}$, *and* $m = 100 \cdot rq \frac{n^{2k/q}}{(1-2\eta)^6}$ *examples from an* (n, k, η) *instance of parity with noise, will return the correct set of* k *parity bits with probability at least* $1 - o(1)$. *Additionally, the number of examples used is* m; *for sufficiently large constant* q, *the number of examples is bounded by* $n^{\epsilon k}$, *and the total runtime of the algorithm is bounded by*

$$poly\left(\frac{1}{(1-2\eta)}\right) \cdot n^{\frac{k}{3}(1+\epsilon)\omega},$$

where $\omega < 2.4$ *is the matrix multiplication exponent.*

Proof. The proof follows from noting first that $m' < \binom{m/r}{q}^4$, and thus with probability $1 - o(1)$, in all r runs, no two choices of random subsets of $[m]$ of size q chosen in the construction of X' will be equal, and thus with this probability, the rows of X' (and thus Y and Y') will all be pairwise independent. Thus, from the proof of Theorem 5 and Chebyshev's inequality, for each pair S, S' of disjoint sets of size $k/3$ that are not both subsets of the true set of parity bits, $c^i_{S,S'} \leq \frac{m^i (2/\sqrt{n})^{k/3}}{3}$ with probability at least $1/2$. Since each of the r runs are independent, the probability that such a bad pair of sets remains after all $r = 100k \log n$ runs is at most $\frac{1}{n^{100k}}$, and thus via a union bound over the at most $\binom{n}{k/3}^2$ such pairs of bad sets, with probability $1 - o(1)$, no such bad pairs of sets will appear in the final output set $ParityBits$.

By Proposition 41, and our choice of parameters, with probability $1 - o(1/n)$, the total variation distance between the assignment of the values to the k true parity columns of matrix X' in a given run, and if they were chosen uniformly is at most $o(1/n)$, and thus with probability $1 - o(1)$, after the r runs, the algorithm must perform identically to the performance in the case that these columns were chosen uniformly at random, and thus the arguments of the proof of Theorem 5, and, in particular,

the Chernoff bound, guarantees that with probability $1 - o(1)$ in *all* r runs, every pair of disjoint sets S, S' of size k that are subsets of the parity bits, will satisfy $c^i_{S,S'} > \frac{m^i(2/\sqrt{n})^{k/3}}{3}$, as desired. □

6 Further Directions: Beyond Fast Matrix Multiplication

Beyond the more obvious open questions posed by these improved algorithms, one very relevant direction for future work is to give algorithms that improve over the brute-force search *in practice, for modest-sized datasets*. For instance:

> *Does there exist an algorithm for finding a pair of* 0.05*-correlated Boolean vectors from among* $n = 100,000$ *uniformly random Boolean vectors that significantly beats brute-force-search,* in practice*?*

There are two natural angles to this question. The first is to try to improve fast matrix multiplication implementations. While the algorithms described in this work rely on fast matrix multiplication, they do not require an especially accurate multiplication. In particular, our algorithms would still succeed if they used a noisy matrix multiplication, or even an algorithm that "misplaced" a constant fraction of the cross-terms. (For example, for $n \times n$ matrices A, B, C, in computing $AB = C$, the entry $c_{i,j}$ should be the sum of n cross terms $a_{i,k} \cdot b_{k,j}$; our algorithms would be fine if only, say, half of these cross terms ended up contributing to $c_{i,j}$.) Tolerating such "sloppiness" seems unlikely to allow for faster asymptotic bounds on the runtime (at least within the Coppersmith–Winograd framework), though it may significantly reduce the overhead on some of the more practically expensive components of the Coppersmith-Winograd framework.

The second approach to yielding a practical algorithm would be to avoid fast matrix multiplication entirely. Our Vector Aggregation algorithm of Section 3 seems natural (if many pairwise inner products are extremely small, we should "bucket" them in such a way that we can process them in bulk, yet still be able to detect which bucket contains the large inner product). Nevertheless, if one replaces the fast matrix multiplication step with the naive cubic-time multiplication, one gets no improvement over the quadratic brute-force search. It seems that no clever bucketing schemes (in the "aggregation" step, one need not simply add the vectors over the reals...), or fancy embeddings can remove the need for fast matrix multiplication.

One intuitive explanation for the difficulty of avoiding fast matrix multiplication is via the connection between finding correlations, and learning parity with noise. The statistical query (SQ) lower bound of Blum et al. [7], informally, implies that any algorithm that will beat brute-force-search must be highly non-SQ; in particular, it must perform nontrivial operations that intertwine at least $\log n$ rows of the matrix whose columns are the given vectors. Fast matrix multiplication is clearly such an algorithm.

Given this intuitive need for a non-SQ component of the algorithm, perhaps the most likely candidate for an off-the-shelf algorithm that might replace fast matrix multiplication, is the Fast Fourier Transform. In a recent paper, Pagh gives an extremely clean and practically viable algorithm for computing or approximating the product of two matrices given the promise that their product is sparse, or has small Frobenius norm after one removes a small number of large entries [27]. The algorithmic core of Pagh's approach is the computation of a Fourier transform. Perplexingly, despite the fact that Pagh's results specifically apply to the type of matrix products that we require for our algorithms that find correlations and parities, it does not seem possible to improve on the trivial brute-force search runtimes by using Pagh's matrix multiplication algorithm.

References

[1] M. Ajtai, R. Kumar, and D. Sivakumar. A sieve algorithm for the shortest lattice vector problem. In *Proceedings of the ACM Symposium on Theory of Computing (STOC)*, pages 601–610, 2001.

[2] N. Alon and A. Naor. Approximating the cut-norm via Grothendiecks inequality. In *Proceedings of the ACM Symposium on Theory of Computing (STOC)*, pages 72–80, 2004.

[3] A. Andoni and P. Indyk. Near-optimal hashing algorithms for approximate nearest neighbor in high dimensions. In *IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 459–468, 2006.

[4] A. Andoni and P. Indyk. Near–optimal hashing algorithms for approximate nearest neighbor in high dimensions. *Communications of the ACM*, 51(1):117–122, 2008.

[5] S. Arora and R. Ge. New algorithms for learning in presence of errors. In *International Colloquium on Automata, Languages and Programming (ICALP)*, pages 403–415, 2011.

[6] J.L. Bentley. Multidimensional binary search trees used for associative searching. *Communications of the ACM*, 18(9):509–517, 1975.

[7] A. Blum, M. Furst, J. Jackson, M. Kearns, Y. Mansour, and S. Rudich. Weakly learning DNF and characterizing statistical query learning using Fourier analysis. In *Proceedings of the ACM Symposium on Theory of Computing (STOC)*, pages 253–262, 1994.

[8] A. Blum, A. Kalai, and H. Wasserman. Noise-tolerant learning, the parity problem, and the statistical query model. *Journal of the ACM (JACM)*, 50(4):507–519, 2003.

[9] Z. Brakerski and V. Vaikuntanathan. Efficient fully homomorphic encryption from (standard) LWE. In *IEEE Symposium on Foundations of Computer Science (FOCS)*, 2011.

[10] K. Clarkson. A randomized algorithm for closest–point queries. *SIAM Journal on Computing*, 17(4):830–847, 1988.

[11] D. Coppersmith. Rectangular matrix multiplication revisited. *Journal of Complexity*, 13(1):42–49, 1997.

[12] M. Datar, N. Immorlica, P. Indyk, and V. Mirrokni. Locality–sensitive hashing scheme based on p-stable distributions. In *Proceedings of the 20th ACM Symposium on Computational Geometry (SoCG)*, pages 253–262, 2004.

[13] M. Dubiner. Bucketing coding and information theory for the statistical high dimensional nearest neighbor problem. *CoRR*, abs/0810.4182, 2008.

[14] V. Feldman, P. Gopalan, S. Khot, and A. Ponnuswami. New results for learning noisy parities and halfspaces. In *IEEE Symposium on Foundations of Computer Science (FOCS)*, 2006.

[15] E. Grigorescu, L. Reyzin, and S. Vempala. On noise-tolerant learning of sparse parities and related problems. In *The 22nd International Conference on Algorithmic Learning Theory (ALT)*, 2011.

[16] N. J. Hopper and A. Blum. Secure human identification protocols. In *ASIACRYPT*, pages 52–66, 2001.

[17] R. Impagliazzo and D. Zuckerman. How to recycle random bits. In *IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 248–253, 1989.

[18] P. Indyk and R. Motwani. Approximate nearest neighbors: towards removing the curse of dimensionality. In *Proceedings of the ACM Symposium on Theory of Computing (STOC)*, 1998.

[19] M. Kearns. Efficient noise-tolerant learning from statistical queries. *Journal of the ACM*, 45(6):983–1006, 1998.

[20] E. Kushilevitz, R. Ostrovsky, and Y. Rabani. Efficient search for approximate nearest neighbor in high dimensional spaces. *SIAM Journal on Computing*, 30(2):457–474, 2000.

[21] V. Lyubashevsky. The parity problem in the presence of noise, decoding random linear codes, and the subset sum problem. In *RANDOM*, pages 378–389, 2005.

[22] J. Marchini, P. Donnelly, and L.R. Cardon. Genome-wide strategies for detecting multiple loci that influence complex diseases. *Nature Genetics*, 37(4):413–417, 2005.

[23] S. Meiser. Point location in arrangements of hyperplanes. *Information and Computation*, 106(2):286–303, 1993.

[24] E. Mossel, R. O'Donnell, and R. Servedio. Learning functions of k relevant variables. *Journal of Computer and System Sciences*, 69(3):421–434, 2004.

[25] R. Motwani, A. Noar, and R. Panigraphy. Lower bounds on locality sensitive hashing. In *Proceedings of the ACM Symposium on Computational Geometry (SoCG)*, pages 154–157, 2006.

[26] R. O'Donnell, Y. Wu, and Y. Zhou. Optimal lower bounds for locality sensitive hashing (except when q in tiny). In *Innovations in Theoretical Computer Science (ITCS)*, pages 275–283, 2011.

[27] R. Pagh. Compressed matrix multiplication. In *Innovations in Theoretical Computer Science (ITCS)*, 2012.

[28] R. Panigraphy. Entropy–based nearest neighbor search in high dimensions. In *Proceedings of the 17th ACM-SIAM Symposium on Discrete Algorithms (SODA)*, 2006.

[29] Ramamohan Paturi, Sanguthevar Rajasekaran, and John H. Reif. The light bulb problem. In *Conference on Learning Theory (COLT)*, pages 261–268, 1989.

[30] C. Peikert. Public–key cryptosystems from the worst-case shortest vector problem. In *Proceedings of the ACM Symposium on Theory of Computing (STOC)*, pages 333–342, 2009.

[31] O. Regev. On lattices, learning with errors, random linear codes, and cryptography. *Journal of the ACM*, 56(6):1–40, 2009.

[32] O. Regev. The learning with errors problem. *Invited survey in IEEE Conference on Computational Complexity (CCC)*, 2010.

[33] T.J. Rivlin. *The Chebyshev Polynomials*. John Wiley and Sons, 1974.

[34] H. Samet. *Foundations of Multidimensional and Metric Data Structures*. Elsevier, 2006.

[35] I.J. Schoenberg. Positive definite functions on spheres. *Duke Mathematical Journal*, 9(1):96–108, 1942.

[36] G. Szegö. Orthogonal polynomials, 4th edition. *American Mathematical Society, Colloquium Publications, 23. Providence, RI*, 1975.

[37] G. Valiant. Finding correlations in subquadratic time, with applications to learning parities and juntas. In *IEEE Symposium on Foundations of Computer Science (FOCS)*, 2012.

[38] L. Valiant. Functionality in neural nets. In *First Workshop on Computational Learning Theory*, pages 28–39, 1988.

[39] K. A. Verbeurgt. Learning DNF under the uniform distribution in quasipolynomial time. In *Conference on Learning Theory (COLT)*, pages 314–326, 1990.

[40] X. Wan, C. Yang, H. Xue, N. Tang, and W. Yu. Detecting two-locus associations allowing for interactions in genome-wide association studies. *Bioinformatics*, 26(20):2517–2525, 2010.

[41] R. Weber, H.J. Schek, and S. Blott. A quantitative analysis and performance study for similarity–search methods in high–dimensional spaces. In *The 24th International Conference on Very Large Databases (VLDB)*, 1998.

[42] V. Vassilevska Williams. Multiplying matrices faster than Coppersmith–Winograd. In *Proceedings of the ACM Symposium on Theory of Computing (STOC)*, 2012.

A Learning Juntas and DNF via Sparse Parities

We formally state the results of Feldman et al. [14] which reduce the problem of learning Juntas and DNF to the problem of learning parity with noise. The main intuition, and proof approach of [14] is that the problem of learning parities with noise is the problem of finding a heavy Fourier coefficient, given the promise that one exists; in the case of learning a k-junta, one knows that there will be at most 2^k significant Fourier coefficients. The reduction proceeds by essentially peppering the labels with random XORs, so that after the peppering process, with some decent probability, exactly *one* Fourier coefficient will have survived, in which case the problem has been successfully transformed into the problem of learning a parity of size k with noise. It is worth stressing that this reduction results in an instance with a very large noise rate—noise $\frac{1}{2} - \frac{1}{2^k}$, thus highlighting the importance of considering the problem of learning parities with noise in the setting in which the noise-rates approach 1/2. Below we give formal statements of these reductions.

Theorem A.1 (Feldman et al. [14])**.** *Given an algorithm that learns parities of size k on length n strings (under the uniform distribution) with noise rate $\eta \in [0, \frac{1}{2})$ that runs in time $T(n, k, \eta)$, there exists an algorithm that learns k-juntas under the uniform distribution with noise rate η' that runs in time*

$$O\left(k2^{2k} \cdot T(n, k, \frac{1}{2} - \frac{1-2\eta'}{2^k})\right).$$

Theorem A.2 (Feldman et al. [14])**.** *Given an algorithm that learns parities of length k on length n strings (under the uniform distribution) with noise rate $\eta \in [0, \frac{1}{2})$ that takes $S(n, k, \eta)$ examples and runs in time $T(n, k, \eta)$, there exists an algorithm that (ϵ, δ)–PAC learns r-term DNF formulae under the uniform distribution that runs in time*

$$\tilde{O}\left(\frac{r^4}{\epsilon^2} \cdot T\left(n, \log\left(\tilde{O}(r/\epsilon)\right), \frac{1}{2} - \tilde{O}(\epsilon/r)\right) \cdot S\left(n, \log\left(\tilde{O}(r/\epsilon)\right), \frac{1}{2} - \tilde{O}(\epsilon/r)\right)^2\right).$$

Additionally, as Feldman observed, an improved algorithm for learning parities of size k can be used, via the reduction of Feldman et al. [14] to yield an improvement in runtime of the approach of Mossel et al. [24] for the problem of learning k-juntas *without* noise. The key observation of Mossel et al. is that either a k-junta has a heavy Fourier coefficient of degree at most d, or, when represented as a polynomial over $\mathbb{F}_2$, has degree at most $k - d$. Their algorithm proceeds by brute force-searching for a heavy Fourier coefficients of order at most αk for some appropriately chosen α; if none are found, then the junta is found by solving a linear system over $n^{(1-\alpha)k}$ variables. Given an improved algorithm for learning parities with noise, using the reduction of Feldman et al., one improves upon the brute-force search component of the algorithm of Mossel et al. [24]. The following corollary quantifies this improvement.

Corollary A.3. *Given an algorithm that learns parities of length j on length n strings (under the uniform distribution) with noise rate $\eta \in [0, \frac{1}{2})$ that runs in time $T(n, j, \eta)$, for any $\alpha \in (0, 1)$, there exists an algorithm that learns k-juntas* without noise *under the uniform distribution in time*

$$\max\left(T(n, \alpha k, \frac{1}{2} - \frac{1}{2^{\alpha k}}), n^{\omega k(1-\alpha)}\right) poly(n).$$