Fast Learning Requires Good Memory: A Time-Space Lower Bound for Parity Learning

RAN RAZ, Princeton University

We prove that any algorithm for learning parities requires either a memory of quadratic size or an exponential number of samples. This proves a recent conjecture of Steinhardt et al. (2016) and shows that for some learning problems, a large storage space is crucial.

More formally, in the problem of parity learning, an unknown string $x \in \{0,1\}^n$ was chosen uniformly at random. A learner tries to learn x from a stream of samples $(a_1, b_1), (a_2, b_2) \ldots$, where each a_t is uniformly distributed over $\{0,1\}^n$ and b_t is the inner product of a_t and x, modulo 2. We show that any algorithm for parity learning that uses less than $\frac{n^2}{25}$ bits of memory requires an exponential number of samples.

Previously, there was no non-trivial lower bound on the number of samples needed for any learning problem, even if the allowed memory size is $O(n)$ (where n is the space needed to store one sample).

We also give an application of our result in the field of bounded-storage cryptography. We show an encryption scheme that requires a private key of length n, as well as time complexity of n per encryption/decryption of each bit, and is provably and unconditionally secure as long as the attacker uses less than $\frac{n^2}{25}$ memory bits and the scheme is used at most an exponential number of times. Previous works on bounded-storage cryptography assumed that the memory size used by the attacker is at most linear in the time needed for encryption/decryption.

CCS Concepts: • **Theory of computation** → **Computational complexity and cryptography**;

Additional Key Words and Phrases: Lower bounds, bounded storage cryptography, branching program, learning, time-space tradeoff

ACM Reference format:
Ran Raz. 2018. Fast Learning Requires Good Memory: A Time-Space Lower Bound for Parity Learning. *J. ACM* 66, 1, Article 3 (December 2018), 18 pages.
https://doi.org/10.1145/3186563

A preliminary version of the article appeared at the 57th Annual IEEE Symposium on Foundations of Computer Science (FOCS 2016) (Raz 2016).

Research supported by the Israel Science Foundation grant No. 1402/14, by the I-CORE Program of the Planning and Budgeting Committee and the Israel Science Foundation, by the Simons Collaboration on Algorithms and Geometry, by the Fund for Math at IAS, and by the National Science Foundation grants No. CCF-1412958 and CCF-1714779. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author and do not necessarily reflect the views of the National Science Foundation.

Authors' address: R. Raz, Department of Computer Science, Princeton University, 35 Olden Street, Princeton, NJ 08540, USA; email: ran.raz.mail@gmail.com.

0004-5411/2018/12-ART3 $15.00
https://doi.org/10.1145/3186563

1 INTRODUCTION

Parity learning can be solved in polynomial time, by Gaussian elimination, using $O(n)$ samples and $O(n^2)$ memory bits. On the other hand, parity learning can be solved by trying all the possibilities, using $n + o(n)$ memory bits and an exponential number of samples.

We prove that any algorithm for parity learning requires either $\frac{n^2}{25}$ memory bits or an exponential number of samples. Our result may be of interest from the points of view of learning theory, computational complexity, and cryptography.

1.1 Learning Theory

The main message of this article from the point of view of learning theory is that for some learning problems, access to a relatively large memory is crucial. In other words, in some cases, learning is infeasible due to memory constraints. We show that there exist concept classes that can be efficiently learned from a polynomial number of samples, if the learner has access to a quadratic-size memory, but require an exponential number of samples if the memory used by the learner is of less than quadratic size. This gives a formally stated and mathematically proved example for the intuitive feeling that a "good" memory may be very helpful in learning processes.

Many works studied the resources needed for learning, under certain information, communication, or memory constraints (see, in particular, Shamir (2014) and Steinhardt et al. (2016) and the many references given there). However, there was no previous non-trivial lower bound on the number of samples needed, for any learning problem, even when the allowed memory size is bounded by the length of one sample (where we don't count the space taken by the current sample that is being read).

The starting point of our work is the intriguing recent work of Steinhardt et al. (2016). Steinhardt, Valiant, and Wager asked whether there exist concept classes that can be efficiently learned from a polynomial number of samples, but cannot be learned from a polynomial number of samples if the allowed memory size is linear in the length of one sample. They conjectured that the problem of parity learning provides such a separation. Our main result proves that conjecture.

Remark 1. Conjecture 1.1 of Steinhardt et al. (2016) conjectures that for any constant $\epsilon > 0$, any algorithm for parity learning requires either at least $(\frac{1}{4} - \epsilon)n^2$ bits of memory, or at least an exponential number of samples. Our main result qualitatively proves this conjecture, but with a constant of $\frac{1}{20}$ rather than $\frac{1}{4}$.

1.2 Computational Complexity

Time-space tradeoffs have been extensively studied in the field of computational complexity, in many works and various settings. Two brilliant lines of research were particularly successful in establishing time-space lower bounds for computation.

The first line of works (Ajtai 1999a, 1999b; Beame et al. 1998, 2000) gives explicit examples for polynomial-time computable Boolean functions $f : \{0,1\}^n \to \{0,1\}$ such that any algorithm for computing f requires either at least $n^{1-\epsilon}$ memory bits, where $\epsilon > 0$ is an arbitrarily small constant, or time complexity of at least $\Omega(n\sqrt{\log n/\log\log n})$. These bounds are proved for any *branching program* that computes f. Branching programs are the standard and most general computational model for studying time-space tradeoffs in the *non-uniform* setting (which is the more general setting), and is also the computational model that we use in the current work.

The second line of works (Fortnow 1997; Fortnow et al. 2005; Williams 2006, 2007) (and other works) studies time-space tradeoffs for SAT (and other NP problems), in the *uniform* setting, and proves that any algorithm for SAT requires either at least $n^{1-\epsilon}$ memory bits, or time complexity of at least $n^{1+\delta}$ (where $0 < \epsilon, \delta < 1$ are constants). For an excellent survey, see van Melkebeek (2007).

Both lines of works obtain less than quadratic lower bounds on the time needed for computation, under memory constraints. Quadratic lower bounds on the time needed for computation are not known, even if the allowed memory-size is logarithmic. Comparing these results to our work, one may ask what makes it possible to prove exponential lower bounds on the time needed for parity learning, under memory constraints, while the known time-space lower bounds for computations are significantly weaker? The main point to keep in mind is that when studying time-space trade-offs for computing a function, one assumes that the input for the function can always be accessed, and the space needed to store the input doesn't count as memory that is used by the algorithm. Thus, the input is stored for free. In our learning problem, it is assumed that after the learner saw a sample, the learner cannot access that sample again, unless the sample was stored in the learner's memory. The learner can always get a new sample that is "as good as the old one," but she cannot access the same sample that she saw before (without storing it in the memory).

Finally, let us note that by Barrington's celebrated result, any function in NC^1 can be computed by a polynomial-length branching program of width 5 (Barrington 1986). Hence, proving super-polynomial lower bounds on the time needed for computing a function, by a branching program of width 5, would imply super-polynomial lower bounds for formula size.

1.3 Cryptography

Assume that a group of (two or more) users share a (random) secret key $x \in \{0,1\}^n$. Assume that user Alice wants to send an encrypted bit $M \in \{0,1\}$ to user Bob. Let a be a string of n bits, uniformly distributed over $\{0,1\}^n$, and assume that both Alice and Bob know a (we can think of a as taken from a shared random string and if a shared random string is not available, Alice can just choose a randomly and send it to Bob). Let b be the inner product of a and x, modulo 2. Thus, b is known to both Alice and Bob and can be used as a one-time pad to encrypt/decrypt M; that is, Alice encrypts by computing $M \oplus b$ and Bob decrypts by computing $M = (M \oplus b) \oplus b$.

Assume that this protocol is used $m + 1$ times, with the same secret key x, where m is less than exponential. Denote by a_t, b_t the string a and bit b used at time t. Suppose that during all that time, an attacker could see $(a_1, b_1), \ldots, (a_m, b_m)$, but the attacker has less than $\frac{n^2}{25}$ bits of memory. Our main result shows that the attacker cannot guess the secret key x, with better than exponentially small probability. Therefore, using the fact that the inner product is a strong extractor (with exponentially small error), even if the attacker sees a_{m+1}, the attacker cannot predict b_{m+1}, with better than exponentially small advantage over a random guess.

Thus, if the attacker has less than $\frac{n^2}{25}$ bits of memory, the encryption remains secure as long as it is used less than an exponential number of times.

Bounded-storage cryptography, first introduced by Maurer (Maurer 1992) and extensively studied in many works, studies cryptographical protocols that are secure under the assumption that the memory used by the attacker is limited (see, for example, Cachin and Maurer (1997), Aumann and Rabin (1999), Aumann et al. (2002), Vadhan (2003), and Dziembowski and Maurer (2004), and many other works). Previous works on bounded-storage cryptography assumed the existence of a high-rate source of randomness that streams random bits to all parties. The main idea is that the attacker doesn't have sufficiently large memory to store all random bits, and, hence, a shared secret key can be used to randomly select (or extract) bits from the random source that the attacker has very little information about.

In previous works, the number of random bits transmitted during the encryption was assumed to be larger than the memory-size of the attacker. Thus, the time needed for encryption/decryption was at least linear in the memory-size of the attacker. In contrast, the time needed for encryption/decryption in our protocol is n, while the encryption is secure against attackers with memory of size $\frac{n^2}{25}$.

Remark 2. If Alice and Bob want to transmit encrypted messages of length m, where $m \geq n$ (and the attacker has $O(n^2)$ bits of memory), our protocol has no advantage over previous ones, as the time needed for encryption/decryption in our protocol is mn. The advantage of our protocol is in situations where the users want to securely transmit many shorter messages.

1.4 Our Result

Parity Learning. In the problem of parity learning, there is an unknown string $x \in \{0,1\}^n$ that was chosen uniformly at random. A learner tries to learn x from samples (a, b), where $a \in_R \{0,1\}^n$ and $b = a \cdot x$ (where $a \cdot x$ denotes inner product modulo 2). That is, the learning algorithm is given a stream of samples, $(a_1, b_1), (a_2, b_2) \ldots,$ where each a_t is uniformly distributed over $\{0,1\}^n$ and for every t, $b_t = a_t \cdot x$.

Main Result.

THEOREM 1.1. *For any $c < \frac{1}{20}$, there exists an $\alpha > 0$ such that the following holds: Let x be uniformly distributed over $\{0,1\}^n$. Let $m \leq 2^{\alpha n}$. Let A be an algorithm that is given as input a stream of samples, $(a_1, b_1), \ldots, (a_m, b_m)$, where each a_t is uniformly distributed over $\{0,1\}^n$ and for every t, $b_t = a_t \cdot x$. Assume that A uses at most cn^2 memory bits and outputs a string $\tilde{x} \in \{0,1\}^n$. Then, $\Pr[\tilde{x} = x] \leq O(2^{-\alpha n})$.*

Theorem 1.1 is restated, in a stronger[1] and more formal[2] form, as Theorem 7.2 in Section 7, and the proof of Theorem 7.2 is given there.

1.5 Subsequent Work

A line of subsequent works further studies the resources needed for learning, under memory constraints:

Valiant and Valiant (2016) studied the problem of learning sparse parities under information constraints . Their work focuses on the case where the learner can extract only r bits of information from each sample, where $r < n$. In particular, they obtain a lower bound on the number of samples needed, when the memory size of the learner is at most n.

Independently of Valiant and Valiant (2016), building on our techniques, Kol et al. (2017) proved that even if the parity is known to be of sparsity ℓ, parity learning remains infeasible under memory constraints that are super-linear in n, as long as $\ell \geq \omega(\log n / \log\log n)$. Consequently, learning linear-size DNF Formulas, linear-size Decision Trees and logarithmic-size Juntas were all proved to be infeasible under super-linear memory constraints (Kol et al. 2017).

Moshkovitz and Moshkovitz (2017) proved that if a learning problem, when viewed as a bipartite graph between hypotheses and labeled examples, has a (sufficiently strong) mixing property, then any learning algorithm for that learning problem requires either a memory of size at least $1.25 \cdot n$ or at least $2^{\Omega(n)}$ samples, where n is the logarithm (base 2) of the number of hypotheses.

Independently of Moshkovitz and Moshkovitz (2017), in Raz (2017), we proved a general time-space lower bound that applies for a large class of learning problems and shows that for every

[1]Theorem 7.2 allows the algorithm to output an affine subspace of dimension $\leq \frac{3}{5}n$, and bounds by $2^{-\alpha n}$ the probability that x belongs to that affine subspace.

[2]Theorem 7.2 models the algorithm by a branching program, which is more formal and clarifies that the theorem holds also in the (more general) non-uniform setting.

problem in that class, any learning algorithm requires either a memory of quadratic size or an exponential number of samples. The result is stated in terms of the norm of the adjacency matrix of the bipartite graph that corresponds to the learning problem (which can also be viewed as a mixing property). As a special case, the main theorem of Raz (2017) gives an alternative proof for the time-space lower bound for parity learning, presented here, with a completely different set of techniques.

Subsequently to Raz (2017), Moshkovitz and Moshkovitz (2018) built on Moshkovitz and Moshkovitz (2017) and gave a different proof for the main result of Raz (2017).

Two independent (very related) recent works, by Beame et al. (2018) and Garg et al. (2018), build on the the techniques of Raz (2017) and give more general time-space lower bounds for learning, which imply all previous time-space lower bounds (with super-linear lower-bound on the size of the memory and super-polynomial lower-bound on the number of samples), as well as a number of new applications.

Finally, a recent work by Dagan and Shamir (2018) studies the problem of identifying correlations in multivariate data under memory constraints and proves time-space lower bounds for practical estimation problems.

2 PRELIMINARIES

For an integer n, denote by $[n]$ the set $\{1, \ldots, n\}$. For $a, x \in \{0,1\}^n$, denote by $a \cdot x$ their inner product modulo 2.

For a function $P : \Omega \to \mathbb{R}$, we denote by $|P|_1$ its ℓ_1 norm. In particular, for two distributions, $P, Q : \Omega \to [0,1]$, we denote by $|P - Q|_1$ their ℓ_1 distance.

For a random variable X and an event E, we denote by $\mathbb{P}_X$ the distribution of the random variables X, and we denote by $\mathbb{P}_{X|E}$ the distribution of the random variable X conditioned on the event E.

Denote by $\mathcal{U}_n$ the uniform distribution over $\{0,1\}^n$. For an affine subspace (over the field $\mathbb{F}_2$) $w \subseteq \{0,1\}^n$, denote by $\mathcal{U}_w$ the uniform distribution over w.

For $n \in \mathbb{N}$, denote by $\mathcal{A}(n)$ the set of all affine subspaces of $\{0,1\}^n$ (over the field $\mathbb{F}_2$).

3 PROOF OUTLINE

Computational Model. We model the learning algorithm by a *branching program.* A branching program of length m and width d, for parity learning, is a directed (multi) graph with vertices arranged in $m+1$ layers containing at most d vertices each. Intuitively, each layer represents a timestep and each vertex represents a memory state of the learner. In the first layer, which we think of as layer 0, there is only one vertex, called the start vertex. A vertex of outdegree 0 is called a leaf. Every non-leaf vertex in the program has 2^{n+1} outgoing edges, labeled by elements $(a, b) \in \{0,1\}^n \times \{0,1\}$, with exactly one edge labeled by each such (a, b), and all these edges going into vertices in the next layer. Intuitively, these edges represent the action when reading (a_t, b_t). The samples $(a_1, b_1), \ldots, (a_m, b_m) \in \{0,1\}^n \times \{0,1\}$ that are given as input define a computation-path in the branching program by starting from the start vertex and following at step t the edge labeled by (a_t, b_t), until reaching a leaf.

Each leaf v in the program is labeled by an affine subspace $w(v) \in \mathcal{A}(n)$, that we think of as the output of the program on that leaf. The program outputs the label $w(v)$ of the leaf v reached by the computation-path. We interpret the output of the program as a guess that $x \in w(v)$.

We also consider *affine branching programs*, where every vertex v (not necessarily a leaf) is labeled by an affine subspace $w(v) \in \mathcal{A}(n)$ such that the start vertex is labeled by the space $\{0,1\}^n \in \mathcal{A}(n)$, and for any edge (u, v) labeled by (a, b), we have $w(u) \cap \{x' \in \{0,1\}^n : a \cdot x' = b\} \subseteq w(v)$.

These properties guarantee that if the computation-path reaches a vertex v, then $x \in w(v)$. Thus, we can interpret $w(v)$ as an affine subspace that is known to contain x.

An affine branching program is called *accurate* if for (almost) all vertices v, the distribution of x, conditioned on the event that the computation-path reached v, is close to the uniform distribution over $w(v)$.

For exact definitions, see Section 5.

The High-Level Approach. The proof has two parts. We prove lower bounds for affine branching programs, and we reduce general branching programs to affine branching programs. The hard part is the reduction from general branching programs to affine branching programs. We note that this reduction is very wasteful and expands the width of the branching program by a factor of $2^{\Theta(n^2)}$. Nevertheless, since we allow our branching program to be of width up to $2^{O(n^2)}$, this is still affordable (as long as the exact constant in the exponent is relatively small). We have to make sure, when proving time-space lower bounds for affine branching programs, that the upper bounds that we assume on the width of the affine branching programs are larger than the expansion of the width caused by the reduction.

Lower Bounds for Affine Branching Programs. Assume that we have an affine branching program of length at most 2^{cn} and width at most 2^{cn^2}, for a small enough constant c. Fix $k = \frac{4}{5}n$. We prove that the probability that the computation-path reaches some vertex that is labeled with an affine subspace of dimension $\leq k$ is at most $2^{-\Omega(n^2)}$.

Without loss of generality, we can assume that all vertices in the program are labeled with affine subspaces of dimension $\geq k$. Other vertices can just be removed as the computation-path must reach a vertex labeled with a subspace of dimension k before it reaches a vertex labeled with a subspace of dimension $< k$ (because the dimension can decrease by at most 1 along an edge).

We define the "orthogonal" to an affine subspace as the vector space orthogonal to the vector space that defines that affine subspace (that is, the vector space that the affine subspace is given as its translation).

Let v be a vertex in the program such that $w(v)$ is of dimension k. It's enough to prove that the probability that the computation-path reaches v is at most $2^{-\Omega(n^2)}$.

To prove this, we consider the vector spaces "orthogonal" to the affine subspaces that label the vertices along the computation-path, and for each of them, we consider its intersection with the vector space "orthogonal" to $w(v)$. We note that, in each step, the probability that the dimension of the intersection increases is exponentially small (as it requires that the a_t currently being read is contained in some small vector space). Since the dimension of the intersection must increase a linear number of times, in order for the computation-path to reach v, a simple union bound shows that the probability to reach v is at most $2^{-\Omega(n^2)}$.

The full details are given in Lemma 7.1.

From Branching Programs to Affine Branching Programs. In Section 6, we show how to simulate a branching program by an accurate affine branching program. We do that layer after layer. Assume that we are already done with layer $j-1$, so every vertex in layer $j-1$ is already labeled by an affine subspace, and the distribution of x, conditioned on the event that the computation-path reached a vertex, is close to the uniform distribution over the affine subspace that labels that vertex.

Now, take a vertex v in layer j, and consider the distribution of x, conditioned on the event that the computation-path reached the vertex v. By the property that we already know on layer $j-1$, this distribution is close to a convex combination of uniform distributions over affine subspaces of $\{0,1\}^n$.

One could split v into a large number of vertices, one vertex for each affine subspace in the combination. However, this practically means that we would have a vertex for any affine subspace. We would like to keep the number of vertices somewhat smaller. This is done by grouping many affine subspaces into one group. The group will be labeled by an affine subspace that contains all the affine subspaces in the group. Moreover, we will have the property that for each such group, the uniform distribution over the affine subspace that labels the group is close to the relevant weighted average of the uniform distributions over the affine subspaces in the group. Thus, practically, we can replace all the affine subspaces in the group by one affine subspace that represents all of them.

Lemma 4.3 shows that it is possible to group all the affine subspaces into a relatively small number of groups.

We note that the entire inductive argument is delicate, as we cannot afford deteriorating the error multiplicatively in each step and need to make sure that all errors are additive.

4 DISTRIBUTIONS OVER AFFINE SUBSPACES

In this section, we study convex combinations of uniform distributions over affine subspaces of $\{0,1\}^n$. Lemma 4.3 is the only result, proved in this section, that is used outside the section.

In the following lemmas, we have a random variable $W \in \mathcal{A}(n)$ and we consider the distribution $\mathbf{E}_W[\mathcal{U}_W]$. This distribution is a convex combination of uniform distributions over affine subspaces of $\{0,1\}^n$.

The first lemma identifies a condition that implies that the distribution $\mathbf{E}_W[\mathcal{U}_W]$ is close to the uniform distribution over $\{0,1\}^n$.

Lemma 4.1. *Let $W \in \mathcal{A}(n)$ be a random variable. Let $r \geq \frac{n}{2}$. Assume that for every $a \in \{0,1\}^n$ such that $a \neq \vec{0}$, and every $b \in \{0,1\}$,*

$$\Pr_W[\forall x \in W : a \cdot x = b] \leq 2^{-r}.$$

Then,

$$\left| \mathop{\mathbf{E}}_W[\mathcal{U}_W] - \mathcal{U}_n \right|_1 < 2^{-(r-\frac{n}{2})}.$$

Proof. The proof uses Fourier analysis. For any affine subspace $w \subseteq \{0,1\}^n$, the Fourier coefficients of $\mathcal{U}_w$ are:

$$\widehat{\mathcal{U}_w}(a) = \begin{cases} 2^{-n} & \text{if } \forall x \in w : a \cdot x = 0 \\ -2^{-n} & \text{if } \forall x \in w : a \cdot x = 1 \\ 0 & \text{otherwise} \end{cases}$$

Hence, the Fourier coefficients of $\mathbf{E}_W[\mathcal{U}_W]$ are:

$$\widehat{\mathop{\mathbf{E}}_W[\mathcal{U}_W]}(a) = 2^{-n} \cdot \left(\Pr_W[\forall x \in W : a \cdot x = 0] - \Pr_W[\forall x \in W : a \cdot x = 1] \right),$$

and note that this also implies that

$$\widehat{\mathop{\mathbf{E}}_W[\mathcal{U}_W]}(\vec{0}) = 2^{-n}.$$

The Fourier coefficients of $\mathcal{U}_n$ are:

$$\widehat{\mathcal{U}_n}(a) = \begin{cases} 2^{-n} & \text{if } a = \vec{0} \\ 0 & \text{if } a \neq \vec{0} \end{cases}$$

Thus,

$$\sum_{a \in \{0,1\}^n} \left(\widehat{\mathop{\mathbf{E}}_W[\mathcal{U}_W]}(a) - \widehat{\mathcal{U}_n}(a) \right)^2 < 2^n \cdot (2^{-n} \cdot 2^{-r})^2 = 2^{-n-2r}.$$

By Cauchy-Schwarz and Parseval,

$$\left(\mathop{\mathbf{E}}_{x\in_R\{0,1\}^n}\left|\mathop{\mathbf{E}}_{W}[\mathcal{U}_W](x)-\mathcal{U}_n(x)\right|\right)^2 \le \mathop{\mathbf{E}}_{x\in_R\{0,1\}^n}\left(\mathop{\mathbf{E}}_{W}[\mathcal{U}_W](x)-\mathcal{U}_n(x)\right)^2$$
$$= \sum_{a\in\{0,1\}^n}\left(\widehat{\mathop{\mathbf{E}}_{W}[\mathcal{U}_W]}(a)-\widehat{\mathcal{U}_n}(a)\right)^2 < 2^{-n-2r}.$$

Therefore,

$$\left|\mathop{\mathbf{E}}_{W}[\mathcal{U}_W]-\mathcal{U}_n\right|_1 = 2^n \mathop{\mathbf{E}}_{x\in_R\{0,1\}^n}\left|\mathop{\mathbf{E}}_{W}[\mathcal{U}_W](x)-\mathcal{U}_n(x)\right| < 2^n\cdot\sqrt{2^{-n-2r}} = 2^{-(r-n/2)}. \qquad \square$$

The next lemma shows that there always exists an affine subspace $s \subseteq \{0,1\}^n$ such that the distribution $\mathbf{E}_{W|(W\subseteq s)}[\mathcal{U}_W]$ is close to the uniform distribution over s, and the event $W \subseteq s$ occurs with non-negligible probability.

Lemma 4.2. *Let $W \in \mathcal{A}(n)$ be a random variable. Let $r \ge \frac{n}{2}$. There exists an affine subspace $s \subseteq \{0,1\}^n$ such that:*

(1)
$$\Pr_W[W\subseteq s] \ge 2^{-\sum_{i=0}^{n-\dim(s)-1}\left(r-\frac{i}{2}\right)}.$$

(2)
$$\left|\mathop{\mathbf{E}}_{W|(W\subseteq s)}[\mathcal{U}_W]-\mathcal{U}_s\right|_1 < 2^{-\left(r-\frac{n}{2}\right)}.$$

Proof. The proof is by induction on n. The base case, $n = 0$, is trivial, because, in this case, the only element of $\mathcal{A}(n)$ is $\{\vec{0}\}$, so the lemma follows with $s = \{\vec{0}\}$.

Let $n \ge 1$. If for every $a \in \{0,1\}^n$ such that $a \ne \vec{0}$, and every $b \in \{0,1\}$, we have $\Pr_W[\forall x \in W : a\cdot x = b] \le 2^{-r}$. The proof follows by Lemma 4.1, with $s = \{0,1\}^n$. Otherwise, there exists $a \ne \vec{0}$, and $b \in \{0,1\}$ such that $\Pr_W[\forall x \in W : a \cdot x = b] > 2^{-r}$. Denote by u the $(n-1)$-dimensional affine subspace

$$u = \{x \in \{0,1\}^n : a\cdot x = b\}.$$

Thus,

$$\Pr_W[W\subseteq u] > 2^{-r}.$$

Consider the random variable $W' = W \mid (W \subseteq u)$. Since u is an $(n-1)$-dimensional affine subspace, we can identify u with $\{0,1\}^{n-1}$ and think of W' as a random variable over $\mathcal{A}(n-1)$. Hence, by the inductive hypothesis (applied with $n-1$ and $r-\frac{1}{2}$), there exists an affine subspace $s \subseteq u$ such that

(1)
$$\Pr_{W'}[W'\subseteq s] \ge 2^{-\sum_{i=1}^{n-\dim(s)-1}\left(r-\frac{i}{2}\right)}.$$

(2)
$$\left|\mathop{\mathbf{E}}_{W'|(W'\subseteq s)}[\mathcal{U}_{W'}]-\mathcal{U}_s\right|_1 < 2^{-\left(r-\frac{n}{2}\right)}.$$

We will show that s satisfies the two properties claimed in the statement of the lemma.

For the first property, note that since $s \subseteq u$,

$$\Pr[W\subseteq s] = \Pr[W\subseteq u]\cdot\Pr[W\subseteq s \mid W\subseteq u] = \Pr[W\subseteq u]\cdot\Pr[W'\subseteq s]$$
$$> 2^{-r}\cdot 2^{-\sum_{i=1}^{n-\dim(s)-1}\left(r-\frac{i}{2}\right)} = 2^{-\sum_{i=0}^{n-\dim(s)-1}\left(r-\frac{i}{2}\right)}.$$

For the second property, note that since $s \subseteq u$,

$$\mathop{\mathbf{E}}_{W|(W \subseteq s)}[\mathcal{U}_W] = \mathop{\mathbf{E}}_{W'|(W' \subseteq s)}[\mathcal{U}_{W'}]. \qquad \square$$

The next lemma is the main result of this section.

Lemma 4.3. *Let $W \in \mathcal{A}(n)$ be a random variable. Let $r \geq \frac{n}{2}$. There exists a partial function $\sigma : \mathcal{A}(n) \to \mathcal{A}(n)$ such that:*

(1) $\Pr_W[W \notin \text{domain}(\sigma)] \leq 2^{-2n}$.
(2) *For every $w \in \text{domain}(\sigma)$, $w \subseteq \sigma(w)$.*
(3) *For every $s \in \text{image}(\sigma)$,*

$$\left| \mathop{\mathbf{E}}_{W|(\sigma(W)=s)}[\mathcal{U}_W] - \mathcal{U}_s \right|_1 < 2^{-(r-\frac{n}{2})}.$$

(4) *For every $k \in \mathbb{N}$, there are at most*

$$4n \cdot 2^{\sum_{i=0}^{n-k-1}(r-\frac{i}{2})}$$

elements $s \in \text{image}(\sigma)$, with $\dim(s) \geq k$.

Proof. The proof is by repeatedly applying Lemma 4.2. We start with the random variable $W_0 = W$, and apply Lemma 4.2 on W_0. We obtain a subspace s_0 (the subspace s whose existence is guaranteed by Lemma 4.2). For every $w \subseteq s_0$, we define $\sigma(w) = s_0$.

We then define the random variable $W_1 = W_0 \mid (W_0 \not\subseteq s_0)$ and apply Lemma 4.2 on W_1. We obtain a subspace s_1 (the subspace s whose existence is guaranteed by Lemma 4.2). For every $w \subseteq s_1$ on which σ was still not defined, we define $\sigma(w) = s_1$.

In the same way, in step i, we define the random variable $W_i = W_{i-1} \mid (W_{i-1} \not\subseteq s_{i-1})$. Note that $W_i = W \mid (W \not\subseteq s_0) \wedge \ldots \wedge (W \not\subseteq s_{i-1})$; that is, W_i is the restriction of W to the part of $\mathcal{A}(n)$ where σ was still not defined. We apply Lemma 4.2 on W_i and obtain a subspace s_i (the subspace s whose existence is guaranteed by Lemma 4.2). For every $w \subseteq s_i$ on which σ was still not defined, we define $\sigma(w) = s_i$.

We repeat this until $\Pr_W[W \notin \text{domain}(\sigma)] \leq 2^{-2n}$.

Note that for $i' < i$, $s_{i'} \neq s_i$, because the support of W_i doesn't contain any element $w \subseteq s_{i'}$. Hence, the subspaces $s_0, s_1, \ldots$ are all different.

It remains to show that the four properties in the statement of the lemma hold.

The first property is obvious because we continue to define σ on more and more elements repeatedly, until the first property holds. The second property is obvious because we mapped w to s_i only if $w \subseteq s_i$. The third property holds by the second property guaranteed by Lemma 4.2.

The fourth property holds because by the first property guaranteed by Lemma 4.2, in each step where we obtain a subspace s_i of dimension at least k, we define σ on a fraction of at least $2^{-\sum_{i=0}^{n-k-1}(r-\frac{i}{2})}$ of the space that still remains. Thus, after at most $4n \cdot 2^{\sum_{i=0}^{n-k-1}(r-\frac{i}{2})}$ such steps, we have $\Pr[W \notin \text{domain}(\sigma)] \leq 2^{-2n}$, and we stop. Thus, the number of elements s_i, of dimension at least k that we obtain in the process, is at most $4n \cdot 2^{\sum_{i=0}^{n-k-1}(r-\frac{i}{2})}$. $\square$

5 BRANCHING PROGRAMS FOR PARITY LEARNING

In this section, we formally define the parity learning problem and the models of computation that we consider.

Recall that in the problem of parity learning, there is a string $x \in \{0,1\}^n$ that was chosen uniformly at random. A learner tries to learn x from a stream of samples, $(a_1, b_1), (a_2, b_2) \ldots$, where each a_t is uniformly distributed over $\{0,1\}^n$ and for every t, $b_t = a_t \cdot x$.

5.1 General Branching Programs for Parity Learning

In the following definition, we model the learner by a *branching program.* We allow the branching program to output an affine subspace $w \in \mathcal{A}(n)$. We interpret the output of the program as a guess that $x \in w$. Obviously, the output w is more meaningful when $\dim(w)$ is relatively small.

Definition 5.1. **Branching Program for Parity Learning:** A branching program of length m and width d, for parity learning, is a directed (multi) graph with vertices arranged in $m+1$ layers containing at most d vertices each. In the first layer, that we think of as layer 0, there is only one vertex, called the start vertex. A vertex of outdegree 0 is called a leaf. All vertices in the last layer are leaves (but there may be additional leaves). Every non-leaf vertex in the program has 2^{n+1} outgoing edges, labeled by elements $(a,b) \in \{0,1\}^n \times \{0,1\}$, with exactly one edge labeled by each such (a,b), and all these edges going into vertices in the next layer. Each leaf v in the program is labeled by an affine subspace $w(v) \in \mathcal{A}(n)$, that we think of as the output of the program on that leaf.

Computation-Path: The samples $(a_1,b_1),\ldots,(a_m,b_m) \in \{0,1\}^n \times \{0,1\}$ that are given as input define a computation-path in the branching program, by starting from the start vertex and following at step t the edge labeled by (a_t,b_t), until reaching a leaf. The program outputs the label $w(v)$ of the leaf v reached by the computation-path.

Success Probability: The success probability of the program is the probability that $x \in w$, where w is the affine subspace that the program outputs, and the probability is over $x, a_1, \ldots, a_m$ (where $x, a_1, \ldots, a_m$ are uniformly distributed over $\{0,1\}^n$, and for every t, $b_t = a_t \cdot x$).

5.2 Affine Branching Programs for Parity Learning

Next, we define a special type of a branching program for parity learning, that we call an *affine branching program for parity learning*. In an affine branching program for parity learning, every vertex v (not necessarily a leaf) is labeled by an affine subspace $w(v) \in \mathcal{A}(n)$. We will have the property that if the computation-path reaches v, then $x \in w(v)$. Thus, we can interpret $w(v)$ as an affine subspace that is known to contain x.

Definition 5.2. **Affine Branching Program for Parity Learning:** A branching program for parity learning is affine if each vertex v in the program is labeled by an affine subspace $w(v) \in \mathcal{A}(n)$, and the following properties hold:

(1) **Start vertex:** The start vertex is labeled by the space $\{0,1\}^n \in \mathcal{A}(n)$.
(2) **Soundness:** For an edge $e = (u,v)$ labeled by (a,b), denote

$$w(e) = w(u) \cap \{x' \in \{0,1\}^n : a \cdot x' = b\}.$$

Then,

$$w(e) \subseteq w(v).$$

Given an affine branching program for parity learning and samples $(a_1,b_1),\ldots,(a_m,b_m)$ such that for every t, $b_t = a_t \cdot x$, it follows by induction that for every vertex v in the program, if the computation-path reaches v, then $x \in w(v)$. In particular, the output w of the program always satisfies $x \in w$, and, thus, the success probability of an affine program is always 1.

5.3 Accurate Affine Branching Programs for Parity Learning

For a vertex v in a branching program for parity learning, we denote by $\mathbb{P}_{x|v}$ the distribution of the random variable x, conditioned on the event that the vertex v was reached by the computation-path.

Definition 5.3. **ϵ-Accurate Affine Branching Program for Parity Learning:** An affine branching program of length m for parity learning is ϵ-accurate if all the leaves are in the last layer, and the following additional property holds (where $x, a_1, \ldots, a_m$ are uniformly distributed over $\{0,1\}^n$, and for every t, $b_t = a_t \cdot x$):

(3) **Accuracy:** Let $0 \le t \le m$. Let V_t be the vertex in layer t, reached by the computation-path. Let y_t be a random variable uniformly distributed over the subspace $w(V_t)$. Then,

$$\left|\mathbb{P}_{V_t,x} - \mathbb{P}_{V_t,y_t}\right|_1 \le \epsilon,$$

or, equivalently,

$$\mathop{\mathbf{E}}_{V_t} \left|\mathbb{P}_{x|V_t} - \mathcal{U}_{w(V_t)}\right|_1 \le \epsilon.$$

Intuitively, accuracy means that what the program remembers on x, given that it reached some vertex v, is that $x \in w(v)$ and (almost) nothing more than that.

6 FROM BRANCHING PROGRAMS TO AFFINE BRANCHING PROGRAMS

In this section, we show that any branching program B for parity learning can be simulated by an affine branching program P for parity learning. Roughly speaking, each vertex of the simulated program B will be represented by a set of vertices of the simulating program P. Note that the width of P will typically be significantly larger than the width of B.

More precisely, a branching program B for parity learning is simulated by a branching program P for parity learning if there exists a mapping Γ from the vertices of P to the vertices of B, and the following properties hold:

(1) **Preservation of structure:** For every i, Γ maps layer i of P to layer i of B. Moreover, Γ maps leaves to leaves and non-leaf vertices to non-leaf vertices. Note that Γ is not necessarily one-to-one.
(2) **Preservation of functionality:** For every edge (u,v), labeled by (a,b), in P, there is an edge $(\Gamma(u),\Gamma(v))$, labeled by (a,b), in B.

Lemma 6.1. *Let $k' < n$. Assume that there exists a length m and width d branching program B for parity learning (of size n) such that: all leaves of B are in the last layer; the output of B is always an affine subspace of dimension $\le k'$; and the success probability of B is β.*

Let $\frac{n}{2} \le r \le n$. Let $\epsilon = 4m \cdot 2^{-(r-\frac{n}{2})}$. Then, there exists an ϵ-accurate length m affine branching program P for parity learning (of size n) such that:

(1) For every $k < n$, the number of vertices in P that are labeled with an affine subspace of dimension k is at most

$$4n \cdot 2^{\sum_{i=0}^{n-k-1}(r-\frac{i}{2})} \cdot dm.$$

(2) For every k such that $k' < k < n$, the output of P is an affine subspace of dimension $< k$, with probability of at least

$$\beta - \epsilon - 2^{-(k-k')}.$$

Proof. For every $0 \le j \le m$, let $\epsilon_j = 4j \cdot 2^{-(r-\frac{n}{2})}$. We will use Lemma 4.3 to turn, inductively, the layers of B, one by one, into layers of an ϵ-accurate affine branching program, P. In step j of the induction, we will turn layer j of B into layer j of P, and define the label $w(v) \in \mathcal{A}(n)$ for every vertex v in that layer of P. Formally, we will construct, inductively, a sequence of programs $B, P_0, \ldots, P_m = P$, where each program is of length m, and for every j, the program P_j differs from the previous program only in layer j (and in the edges going into layer j and out of layer j). After step j of the induction, we will have a branching program P_j such that layers 0 to j of P_j form an affine branching program for parity learning. In addition, the following inductive hypothesis will hold:

Inductive Hypothesis: Let $\mathcal{L}_j$ be the set of vertices in layer j of P_j. Let V_j be the vertex in $\mathcal{L}_j$, reached by the computation-path of P_j. Note that V_j is a random variable that depends on $x, a_1, \ldots, a_j$ (and recall that $x, a_1, \ldots, a_m$ are uniformly distributed over $\{0,1\}^n$, and for every t, $b_t = a_t \cdot x$). The inductive hypothesis is that there exists a random variable U_j over $\mathcal{L}_j$ such that if y_j is a random variable uniformly distributed over the subspace $w(U_j)$, then

$$\left|\mathbb{P}_{V_j,x} - \mathbb{P}_{U_j,y_j}\right|_1 \le \tfrac{\epsilon_j}{2}. \tag{1}$$

The inductive hypothesis is equivalent to the *accuracy* requirement (see Definition 5.3) for layer j of P_j, up to a small multiplicative constant in the accuracy; but we need to assume it in this slightly different form in order to avoid deteriorating the accuracy by a multiplicative factor in each step of the induction.

Base Case: In the base case of the induction, $j = 0$, we define P_0 by just labeling the start vertex of B by $\{0,1\}^n \in \mathcal{A}(n)$. Thus, the *start vertex* property in the definition of an affine branching program is satisfied. The *soundness* property is trivially satisfied because the restriction of P_0 to layer 0 contains no edges. Since we always start from the start vertex, the distribution of the random variable x, conditioned on the event that we reached the start vertex, is just $\mathcal{U}_n$, and, hence, the inductive hypothesis (Equation (1)) holds with $U_0 = V_0$.

Inductive Step: Assume that we already turned layers 0 to $j-1$ of B into layers 0 to $j-1$ of P. That is, we already defined the program P_{j-1}, and layers 0 to $j-1$ of P_{j-1} satisfy the *start vertex* property, the *soundness* property, and the inductive hypothesis (Equation (1)). We will now show how to define P_j from P_{j-1}, that is, how to turn layer j of B into layer j of P.

Let $U_{j-1} \in \mathcal{L}_{j-1}$ be the random variable that satisfies the inductive hypothesis (Equation (1)) for layer $j-1$ of P_{j-1}. Let y_{j-1} be a random variable uniformly distributed over the subspace $w(U_{j-1})$. Let $a \in_R \{0,1\}^n$. Let $b = a \cdot y_{j-1}$. Let $E = (U_{j-1}, V)$ be the edge labeled by (a,b) outgoing U_{j-1} in P_{j-1}. Thus, V is a vertex in layer j of P_{j-1}. Let $W = w(E)$, where $w(E)$ is defined as in the *soundness* property in Definition 5.2. That is,

$$w(E) = w(U_{j-1}) \cap \{x' \in \{0,1\}^n : a \cdot x' = b\},$$

where (a,b) is the label of E, and $w(U_{j-1})$ is the label of U_{j-1} in P_{j-1}.

Let v be a vertex in layer j of P_{j-1} (and note that v is also a vertex in layer j of B). Let

$$W_v = W|(V = v).$$

Let $\sigma_v : \mathcal{A}(n) \to \mathcal{A}(n)$ be the partial function whose existence is guaranteed by Lemma 4.3, when applied on the random variable W_v. Extend $\sigma_v : \mathcal{A}(n) \to \mathcal{A}(n)$ so that it outputs the special value $*$ on every element where it was previously undefined.

In the program P_j, we will split the vertex v into $|\text{image}(\sigma_v)|$ vertices (where $\text{image}(\sigma_v)$ already contains the additional special value $*$). For every $s \in \text{image}(\sigma_v)$, we will have a vertex (v,s). If $s \neq *$, we label the vertex (v,s) by the affine subspace s, and we label the additional vertex $(v,*)$ by $\{0,1\}^n$. For every $s \in \text{image}(\sigma_v)$, the edges going out of (v,s) (in P_j) will be the same as the edges going out of v in P_{j-1}. That is, for every edge (v,v') (from layer j to layer $j+1$) in the program P_{j-1}, and every $s \in \text{image}(\sigma_v)$, we will have an edge $((v,s),v')$ with the same label, (from layer j to layer $j+1$) in the program P_j.

We will now define the edges going into the vertices (v,s) in the program P_j. For every edge $e = (u,v)$, labeled by (a,b), (from layer $j-1$ to layer j), in the program P_{j-1}, consider the affine subspace $w = w(e) = w(u) \cap \{x' \in \{0,1\}^n : a \cdot x' = b\}$ (as in the *soundness* property in Definition 5.2), where $w(u)$ is the label of u in P_{j-1}. Let $s = \sigma_v(w)$. Then, in P_j, we will have the edge $(u, (v, s))$ (labeled by (a, b)), from layer $j-1$ to layer j; that is, we connect u to (v, s). Note that the edge $(u, (v, s))$ satisfies the *soundness* property in the definition of an affine branching program: If $s \neq *$, the vertex (v, s) is labeled by $s = \sigma_v(w)$ and by Property 2 of Lemma 4.3, $w \subseteq \sigma_v(w)$. If $s = *$, the vertex (v, s) is labeled by $\{0, 1\}^n$ and, hence, the *soundness* property is trivially satisfied.

Proof of the Inductive Hypothesis: Next, we will prove the inductive hypothesis (Equation (1)), for P_j. We will define the random variable $U_j \in \mathcal{L}_j$ as follows:

As before, let $U_{j-1} \in \mathcal{L}_{j-1}$ be the random variable that satisfies the inductive hypothesis (Equation (1)) for layer $j-1$ of P_{j-1}. Let y_{j-1} be a random variable uniformly distributed over the subspace $w(U_{j-1})$. Let $a \in_R \{0, 1\}^n$. Let $b = a \cdot y_{j-1}$. Let $E = (U_{j-1}, V)$ be the edge labeled by (a, b) outgoing U_{j-1} in P_{j-1}. Thus, V is a vertex in layer j of P_{j-1}. As before, let $W = w(E) = w(U_{j-1}) \cap \{x' \in \{0, 1\}^n : a \cdot x' = b\}$. As before, for a vertex v in layer j of P_{j-1}, let $\sigma_v : \mathcal{A}(n) \to \mathcal{A}(n)$ be the partial function whose existence is guaranteed by Lemma 4.3, when applied on the random variable $W_v = W|(V = v)$, and extend $\sigma_v : \mathcal{A}(n) \to \mathcal{A}(n)$ so that it outputs the special value $*$ on every element where it was previously undefined.

We define $U_j = (V, \sigma_V(W)) \in \mathcal{L}_j$. Let y_j be a random variable uniformly distributed over the subspace $w(U_j)$, and let V_j be the vertex in $\mathcal{L}_j$, reached by the computation-path of P_j. We need to prove that

$$\left|\mathbb{P}_{V_j, x} - \mathbb{P}_{U_j, y_j}\right|_1 \leq 2j \cdot 2^{-(r-\frac{n}{2})}. \tag{2}$$

Let y'_j be a random variable uniformly distributed over the subspace W. Equation (2) follows by the following two equations and by the triangle inequality:

$$\left|\mathbb{P}_{U_j, y'_j} - \mathbb{P}_{U_j, y_j}\right|_1 \leq 2 \cdot 2^{-(r-\frac{n}{2})}. \tag{3}$$

$$\left|\mathbb{P}_{V_j, x} - \mathbb{P}_{U_j, y'_j}\right|_1 \leq 2(j-1) \cdot 2^{-(r-\frac{n}{2})}. \tag{4}$$

Thus, it is sufficient to prove Equations (3) and (4). We will start with Equation (3).

By Property 3 of Lemma 4.3, for every v in layer j of P_{j-1}, and every $s \in \text{image}(\sigma_v) \setminus \{*\}$,

$$\left|\mathop{\mathbf{E}}_{W|(V=v), (\sigma_v(W)=s)}[\mathcal{U}_W] - \mathcal{U}_s\right|_1 < 2^{-(r-\frac{n}{2})}.$$

By the definitions of y'_j and U_j,

$$\mathop{\mathbf{E}}_{W|(V=v), (\sigma_v(W)=s)}[\mathcal{U}_W] = \mathop{\mathbf{E}}_{W|(U_j=(v,s))}[\mathcal{U}_W] = \mathbb{P}_{y'_j|(U_j=(v,s))}.$$

By the definition of y_j,

$$\mathcal{U}_s = \mathbb{P}_{y_j|(U_j=(v,s))}$$

Hence,

$$\left|\mathbb{P}_{y'_j|(U_j=(v,s))} - \mathbb{P}_{y_j|(U_j=(v,s))}\right|_1 < 2^{-(r-\frac{n}{2})}.$$

Taking expectation over U_j, and taking into account that, by Property 1 of Lemma 4.3, for every v, $\Pr(\sigma_v(W) = *) \leq 2^{-2n}$, we obtain

$$\left|\mathbb{P}_{U_j, y'_j} - \mathbb{P}_{U_j, y_j}\right|_1 = \mathop{\mathbf{E}}_{U_j}\left|\mathbb{P}_{y'_j|U_j} - \mathbb{P}_{y_j|U_j}\right|_1 < 2^{-(r-\frac{n}{2})} + 2^{-2n} \cdot 2,$$

which proves Equation (3).

We will now prove Equation (4). Let $\mathcal{T}$ be the following probabilistic transformation from $\mathcal{L}_{j-1} \times \{0, 1\}^n$ to $\mathcal{L}_j \times \{0, 1\}^n$. Given $(u, z) \in \mathcal{L}_{j-1} \times \{0, 1\}^n$, the transformation $\mathcal{T}$ chooses $a \in_R \{0, 1\}^n$ and $b = a \cdot z$, and outputs (V, z), where $V \in \mathcal{L}_j$ is the vertex obtained by following the edge labeled by (a, b) outgoing u in P_j.

By the definition of the computation-path, $\mathcal{T}(V_{j-1}, x)$ has the same distribution as (V_j, x). By the definition of U_j, y_j, y'_j, we have that $\mathcal{T}(U_{j-1}, y_{j-1})$ has the same distribution as (U_j, y'_j). Hence, by the inductive hypothesis and since the transformation $\mathcal{T}$ cannot increase ℓ_1 distance,

$$\left|\mathbb{P}_{V_j,x} - \mathbb{P}_{U_j,y'_j}\right|_1 = \left|\mathbb{P}_{\mathcal{T}(V_{j-1},x)} - \mathbb{P}_{\mathcal{T}(U_{j-1},y_{j-1})}\right|_1 \leq \left|\mathbb{P}_{V_{j-1},x} - \mathbb{P}_{U_{j-1},y_{j-1}}\right|_1 \leq 2(j-1)\cdot 2^{-(r-\frac{n}{2})},$$

which gives Equation (4).

Since, by induction, layers 0 to $j-1$ of P_{j-1} form an affine branching program for parity learning, and since we already saw that all the edges between layer $j-1$ and layer j of P_j satisfy the *soundness* property in the definition of an affine branching program, we have that layers 0 to j of P_j form an affine branching program for parity learning.

P is ϵ-Accurate: We will now prove that the final branching program $P = P_m$, which we obtained, satisfies the requirements of the lemma. We already know that P is an affine branching program for parity learning.

We will start by proving that P is ϵ-accurate. Let $0 \leq t \leq m$. Let V_t be the vertex in layer t of P, reached by the computation-path of P. Let z_t be a random variable uniformly distributed over the subspace $w(V_t)$. We need to prove that

$$\left|\mathbb{P}_{V_t,x} - \mathbb{P}_{V_t,z_t}\right|_1 \leq \epsilon. \tag{5}$$

Recall that by the inductive hypothesis (Equation (1)), there exists a random variable U_t over layer t of P such that if y_t is a random variable uniformly distributed over the subspace $w(U_t)$, then

$$\left|\mathbb{P}_{V_t,x} - \mathbb{P}_{U_t,y_t}\right|_1 \leq \tfrac{\epsilon}{2}, \tag{6}$$

and this also implies that

$$\left|\mathbb{P}_{V_t} - \mathbb{P}_{U_t}\right|_1 \leq \tfrac{\epsilon}{2}.$$

By the last inequality and since for every v in layer t of P, it holds that $\mathbb{P}_{z_t|(V_t=v)} = \mathbb{P}_{y_t|(U_t=v)}$ (since they are both uniformly distributed over $w(v)$), we have

$$\left|\mathbb{P}_{V_t,z_t} - \mathbb{P}_{U_t,y_t}\right|_1 = \left|\mathbb{P}_{V_t} - \mathbb{P}_{U_t}\right|_1 \leq \tfrac{\epsilon}{2}. \tag{7}$$

Equation (5) follows by Equation (6), Equation (7), and the triangle inequality.

P Satisfies the Additional Properties: We will now prove that P satisfies the two additional properties claimed in the statement of the lemma. The first property holds since Property 4 of Lemma 4.3 ensures that for every vertex in layers 1 to m of the branching program B, we obtain at most $4n \cdot 2^{\sum_{i=0}^{n-k-1}(r-\frac{i}{2})}$ vertices in the branching program P that are labeled with affine subspaces of dimension k.

It remains to prove the second property. Let $V_m = (V, S)$ be the vertex in layer m of P, reached by the computation-path of P. Note that V_m is a random variable that depends on $x, a_1, \ldots, a_m$ (and recall that $x, a_1, \ldots, a_m$ are uniformly distributed over $\{0,1\}^n$ and for every t, $b_t = a_t \cdot x$).

Note that V is the vertex in layer m of B, reached by the computation-path of B (on the same $x, a_1, \ldots, a_m$). This is true since P simulates B. More precisely, by the construction, if on $x, a_1, \ldots, a_m$, the program P reaches (V, S), then, on the same $x, a_1, \ldots, a_m$, the program B reaches V.

Since the success probability of B is β,

$$\Pr[x \in w(V)] = \beta,$$

where $w(V)$ is the label of V in B. Let y_m be a random variable uniformly distributed over the subspace $w(V_m)$, where $w(V_m)$ is the label of V_m in P. Since P is ϵ-accurate,

$$\left|\mathbb{P}_{V,x} - \mathbb{P}_{V,y_m}\right|_1 \leq \left|\mathbb{P}_{V,S,x} - \mathbb{P}_{V,S,y_m}\right|_1 = \left|\mathbb{P}_{V_m,x} - \mathbb{P}_{V_m,y_m}\right|_1 \leq \epsilon.$$

Thus,

$$\Pr[y_m \in w(V)] \geq \Pr[x \in w(V)] - \epsilon = \beta - \epsilon.$$

Let $k > k'$. Recall that $w(V)$ is of dimension $\leq k'$. Thus, if $w(V_m)$ is of dimension $\geq k$, the (conditional) probability that $y_m \in w(V)$ is at most $2^{k'-k}$. Thus,

$$\beta - \epsilon \leq \Pr[y_m \in w(V)] \leq \Pr[\dim(w(V_m)) < k] + 2^{k'-k}.$$

That is,

$$\Pr[\dim(w(V_m)) < k] \geq \beta - \epsilon - 2^{-(k-k')}. \qquad \square$$

7 TIME-SPACE LOWER BOUNDS FOR PARITY LEARNING

In this section, we will use Lemma 6.1 to prove Theorem 7.2, our main result. Recall that Theorem 7.2 (stated below) is stronger than Theorem 1.1, and, hence, Theorem 1.1 follows as well. We start by a lemma that will be used in the proof of Theorem 7.2 to obtain time-space lower bounds for *affine* branching programs.

Lemma 7.1. *Let $k < n$. Let P be a length m affine branching program for parity learning (of size n) such that for every vertex u of P, $\dim(w(u)) \geq k$. Let v be a vertex of P such that $\dim(w(v)) = k$. Then, the probability that the computation-path of P reaches v is at most*

$$m^{n-k} \cdot 2^{\sum_{j=0}^{n-k-1}(n-2k-j)}.$$

Proof. Let s be the vector space "orthogonal" to $w(v)$ in $\{0,1\}^n$. That is,

$$s = \{a \in \{0,1\}^n : \exists b \in \{0,1\}\ \forall x' \in w(v) : a \cdot x' = b\}.$$

Let $V_0, \ldots, V_m$ be the vertices on the computation-path of P. Note that $V_0, \ldots, V_m$ are random variables that depend on $x, a_1, \ldots, a_m$. For every $0 \leq i \leq m$, let S_i be the vector space "orthogonal" to $w(V_i)$ in $\{0,1\}^n$. That is,

$$S_i = \{a \in \{0,1\}^n : \exists b \in \{0,1\}\ \forall x' \in w(V_i) : a \cdot x' = b\}.$$

By the *soundness* property in Definition 5.2, for every $1 \leq i \leq m$,

$$S_i \subseteq \operatorname{span}(S_{i-1} \cup a_i). \tag{8}$$

For every $0 \leq i \leq m$, let $Z_i = \dim(S_i \cap s)$. Note that $Z_0 = 0$, and by Equation (8), for every $1 \leq i \leq m$, $Z_i \leq Z_{i-1} + 1$. If the computation-path of P reaches v, then for some $1 \leq i \leq m$, $Z_i = n - k$. Thus, if the computation-path of P reaches v, then there exist $n - k$ indices $i_1 < \cdots < i_{n-k} \in [m]$, such that the following event denoted by $E_{i_1,\ldots,i_{n-k}}$ occurs:

$$E_{i_1,\ldots,i_{n-k}} = \bigwedge_{j \in [n-k]} (Z_{i_j - 1} = j - 1) \wedge (Z_{i_j} = j).$$

(In particular, $E_{i_1,\ldots,i_{n-k}}$ occurs if for every j, we take i_j to be the first i such that $Z_i = j$). We will bound the probability that the computation-path of P reaches v by bounding $\Pr[E_{i_1,\ldots,i_{n-k}}]$ and taking the union bound over (less than) m^{n-k} possibilities for $i_1, \ldots, i_{n-k} \in [m]$.

Fix $i_1 < \cdots < i_{n-k} \in [m]$. For $r \in \{0, \ldots, n-k\}$, let

$$E_{i_1,\ldots,i_r} = \bigwedge_{j \in [r]} (Z_{i_j - 1} = j - 1) \wedge (Z_{i_j} = j).$$

Thus,

$$\Pr[E_{i_1,\ldots,i_{n-k}}] = \prod_{j\in[n-k]} \Pr[E_{i_1,\ldots,i_j} \mid E_{i_1,\ldots,i_{j-1}}].$$

We will show how to bound $\Pr[E_{i_1,\ldots,i_j} \mid E_{i_1,\ldots,i_{j-1}}]$.

$$\begin{aligned}\Pr[E_{i_1,\ldots,i_j} \mid E_{i_1,\ldots,i_{j-1}}] &= \Pr[(Z_{i_j-1} = j-1) \wedge (Z_{i_j} = j) \mid E_{i_1,\ldots,i_{j-1}}]\\ &= \Pr[(Z_{i_j-1} = j-1) \wedge (Z_{i_j-1} < Z_{i_j}) \mid E_{i_1,\ldots,i_{j-1}}]\\ &\le \Pr[(Z_{i_j-1} < Z_{i_j}) \mid E_{i_1,\ldots,i_{j-1}} \wedge (Z_{i_j-1} = j-1)]. \qquad (9)\end{aligned}$$

Note that the event $E_{i_1,\ldots,i_{j-1}} \wedge (Z_{i_j-1} = j-1)$ that we condition on, on the right-hand side, depends only on $x, a_1, \ldots, a_{i_j-1}$. We will bound the probability for the event $(Z_{i_j-1} < Z_{i_j})$, conditioned on any event that fixes Z_{i_j-1} and depends only on $x, a_1, \ldots, a_{i_j-1}$.

More generally, fix $1 \le i \le m$, and let E'_i be the event $(Z_{i-1} < Z_i)$. Let E' be any event that fixes Z_{i-1} and depends only on $x, a_1, \ldots, a_{i-1}$. Without loss of generality, we can assume that the event E' just fixes the values of $x, a_1, \ldots, a_{i-1}$. We will show how to bound $\Pr[E'_i \mid E']$.

Thus, we fix $x, a_1, \ldots, a_{i-1}$ and we will bound $\Pr[E'_i]$ (conditioned on $x, a_1, \ldots, a_{i-1}$). By Equation (8), if E'_i occurs, then $\dim(S_{i-1} \cap s) < \dim(S_i \cap s) \le \dim(\mathrm{span}(S_{i-1} \cup a_i) \cap s)$; and, hence, $S_{i-1} \cap s \subsetneq \mathrm{span}(S_{i-1} \cup a_i) \cap s$, which implies that there exists $a \in S_{i-1}$ such that $a \oplus a_i \in s$. For every fixed $a \in S_{i-1}$, the event $a \oplus a_i \in s$ occurs with probability $2^{\dim(s)-n} = 2^{(n-k)-n} = 2^{-k}$ (since a_i is uniformly distributed and independent of $x, a_1, \ldots, a_{i-1}$). We will bound the probability for E'_i by taking a union bound over all possibilities for a; but doing so, we take into account that $a \in S_{i-1}$ satisfies $a \oplus a_i \in s$ if and only if every $a' \in a \oplus (S_{i-1} \cap s)$ satisfies $a' \oplus a_i \in s$. Thus, we can take a union bound over $2^{\dim(S_{i-1})-Z_{i-1}} \le 2^{n-k-Z_{i-1}}$ possibilities (where we assume that Z_{i-1} is fixed). Hence, by the union bound

$$\Pr[E'_i \mid E'] \le 2^{n-k-Z_{i-1}} \cdot 2^{-k} = 2^{n-2k-Z_{i-1}}.$$

Thus, in particular, by Equation (9),

$$\Pr[E_{i_1,\ldots,i_j} \mid E_{i_1,\ldots,i_{j-1}}] \le 2^{n-2k-(j-1)}.$$

Hence,

$$\Pr[E_{i_1,\ldots,i_{n-k}}] \le \prod_{j\in[n-k]} 2^{n-2k-(j-1)} = 2^{\sum_{j=0}^{n-k-1}(n-2k-j)}.$$

By the union bound, the probability that the computation-path of P reaches v is at most

$$m^{n-k} \cdot 2^{\sum_{j=0}^{n-k-1}(n-2k-j)}.$$ □

Theorem 7.2. *For any $c < \frac{1}{20}$, there exists $\alpha > 0$ such that the following holds: Let B be a branching program of length at most $2^{\alpha n}$ and width at most 2^{cn^2} for parity learning (of size n) such that the output of B is always an affine subspace of dimension $\le \frac{3}{5}n$. Assume for simplicity, and without loss of generality, that all leaves of B are in the last layer. Then, the success probability of B (that is, the probability that x is contained in the subspace that B outputs) is at most $O(2^{-\alpha n})$.*

Proof. Let $0 < \alpha < \frac{1}{5}$ be a sufficiently small constant (to be determined later on). Let B be a branching program of length $m = 2^{\alpha n}$ and width $d = 2^{cn^2}$ for parity learning (of size n) such that, the output of B is always an affine subspace of dimension $\le \frac{3}{5}n$. Assume for simplicity and without loss of generality that all leaves of B are in the last layer. Denote by β the success probability of B.

Let $r = (\frac{1}{2} + 2\alpha) \cdot n$. Let $k = \frac{4}{5}n$. By Lemma 6.1, there exists a length m affine branching program P for parity learning (of size n), such that:

(1) The number of vertices in P that are labeled with an affine subspace of dimension k is at most
$$4n \cdot 2^{\sum_{i=0}^{n-k-1}(r-\frac{i}{2})} \cdot dm.$$

(2) The output of P is an affine subspace of dimension $\leq k$, with probability of at least
$$\beta - 4 \cdot 2^{-\alpha n} - 2^{-\frac{1}{5}n} \geq \beta - 5 \cdot 2^{-\alpha n}.$$

Assume without loss of generality that every vertex u of P such that $\dim(w(u)) = k$ is a leaf. (Otherwise, we can just redefine u to be a leaf by removing all the edges going out of it). Assume without loss of generality that for every vertex u of P, $\dim(w(u)) \geq k$. (Otherwise, we can just remove u as it is unreachable from the start vertex, since we defined all vertices labeled by subspaces of dimension k to be leaves and since, by the *soundness* property in Definition 5.2, the dimensions along the computation-path can only decrease by 1 in each step).

By Lemma 7.1, and by substituting the values of m, d, k, r, the probability that the computation-path of P reaches some vertex that is labeled with an affine subspace of dimension k is at most
$$\begin{aligned}
&\left(4n \cdot 2^{\sum_{i=0}^{n-k-1}(r-\frac{i}{2})} \cdot dm\right) \cdot \left(m^{n-k} \cdot 2^{\sum_{i=0}^{n-k-1}(n-2k-i)}\right)\\
&= 4nm \cdot 2^{cn^2} \cdot \left(2^{\sum_{i=0}^{n-k-1}(\frac{1}{2}n+2\alpha n-\frac{i}{2})}\right) \cdot \left(2^{\alpha n(n-k)} \cdot 2^{\sum_{i=0}^{n-k-1}(-\frac{3}{5}n-i)}\right)\\
&= 4nm \cdot 2^{cn^2} \cdot 2^{(n-k)(3\alpha n-\frac{1}{10}n)} \cdot \left(2^{\sum_{i=0}^{n-k-1}(-\frac{3}{2}i)}\right)\\
&= 4nm \cdot 2^{cn^2} \cdot 2^{(n-k)(3\alpha n-\frac{1}{10}n)} \cdot 2^{-\frac{3}{4}(n-k)\cdot(n-k-1)}\\
&= 4nm \cdot 2^{cn^2} \cdot 2^{\frac{1}{5}n(3\alpha n-\frac{1}{10}n-\frac{3}{20}n+\frac{3}{4})}\\
&= 4nm \cdot 2^{n^2(c+\frac{3}{5}\alpha-\frac{1}{20}+\frac{3}{20n})}.
\end{aligned}$$

Thus, if $\alpha < \frac{5}{3}(\frac{1}{20} - c)$, this probability is at most $2^{-\Omega(n^2)}$, and, hence,
$$\beta - 5 \cdot 2^{-\alpha n} \leq 2^{-\Omega(n^2)}.$$

That is,
$$\beta \leq O(2^{-\alpha n}).$$

□

ACKNOWLEDGMENTS

This research was done while the author was a faculty member at the Weizmann Institute of Science, Israel, and a visiting Professor at the Institute for Advanced Study, Princeton, NJ.

REFERENCES

Miklós Ajtai. 1999a. Determinism versus non-determinism for linear time RAMs. In *STOC*. 632–641.

Miklós Ajtai. 1999b. A non-linear time lower bound for boolean branching programs. In *FOCS*. 60–70.

Yonatan Aumann, Yan Zong Ding, and Michael O. Rabin. 2002. Everlasting security in the bounded storage model. *IEEE Trans. Inf. Theory* 48, 6 (2002), 1668–1680.

Yonatan Aumann and Michael O. Rabin. 1999. Information theoretically secure communication in the limited storage space model. In *CRYPTO*. 65–79.

David A. Mix Barrington. 1989. Bounded-width polynomial-size branching programs recognize exactly those languages in NC1. *J. Comput. Syst. Sci.* 38, 1 (1989), 150–164. (also in STOC 1986)

Paul Beame, Shayan Oveis Gharan, and Xin Yang. 2018. Time-space tradeoffs for learning finite functions from random evaluations, with applications to polynomials. In *COLT*. 843–856. (also in Electronic Colloquium on Computational Complexity (ECCC) 24: 120 (2017))

Paul Beame, T. S. Jayram, and Michael E. Saks. 2001. Time-space tradeoffs for branching programs. *J. Comput. Syst. Sci. (JCSS)* 63, 4 (2001), 542–572. (also in FOCS 1998)

Paul Beame, Michael E. Saks, Xiaodong Sun, and Erik Vee. 2003. Time-space trade-off lower bounds for randomized computation of decision problems. *J. ACM (JACM)* 50, 2 (2003), 154–195. (also in FOCS 2000)

Christian Cachin and Ueli M. Maurer. 1997. Unconditional security against memory-bounded adversaries. In *CRYPTO.* 292–306.

Stefan Dziembowski and Ueli M. Maurer. 2004. On generating the initial key in the bounded-storage model. In *EUROCRYPT.* 126–137.

Yuval Dagan and Ohad Shamir. 2018. Detecting correlations with little memory and communication. *COLT* (2018), 1145–1198.

Lance Fortnow. 2000. Time-space tradeoffs for satisfiability. *J. Comput. Syst. Sci.* 60, 2 (2000), 337–353. (also in CCC 1997)

Lance Fortnow, Richard J. Lipton, Dieter van Melkebeek, and Anastasios Viglas. 2005. Time-space lower bounds for satisfiability. *J. ACM* 52, 6 (2005), 835–865.

Sumegha Garg, Ran Raz, and Avishay Tal. 2018. Extractor-based time-space lower bounds for learning. In *STOC.* 990–1002. (also in Electronic Colloquium on Computational Complexity (ECCC) 24: 121 (2017))

Gillat Kol, Ran Raz, and Avishay Tal. 2017. Time-space hardness of learning sparse parities. In *STOC.* 1067–1080. (also in Electronic Colloquium on Computational Complexity (ECCC) 23: 113 (2016))

Ueli M. Maurer. 1992. Conditionally-perfect secrecy and a provably-secure randomized cipher. *J. Cryptology* 5, 1 (1992), 53–66.

Dieter van Melkebeek. 2007. A survey of lower bounds for satisfiability and related problems. *Found. Trends Theor. Comput. Sci.* 2 (2007), 197–303.

Dana Moshkovitz and Michal Moshkovitz. 2017. Mixing implies lower bounds for space bounded learning. In *COLT.* 1516–1566. (also in Electronic Colloquium on Computational Complexity (ECCC) 24: 17 (2017))

Dana Moshkovitz and Michal Moshkovitz. 2018. Entropy samplers and strong generic lower bounds for space bounded learning. In *ITCS.* 28:1–28:20. (also in Electronic Colloquium on Computational Complexity (ECCC) 24: 116 (2017))

Ran Raz. 2016. Fast learning requires good memory: A time-space lower bound for parity learning. In *FOCS.* 266–275.

Ran Raz. 2017. A time-space lower bound for a large class of learning problems. In *FOCS.* 732–742. (also in Electronic Colloquium on Computational Complexity (ECCC) 24: 20 (2017))

Ohad Shamir. 2014. Fundamental limits of online and distributed algorithms for statistical learning and estimation. In *NIPS.* 163–171.

Jacob Steinhardt, Gregory Valiant, and Stefan Wager. 2016. Memory, communication, and statistical queries. In *COLT.* 1490–1516. (also in Electronic Colloquium on Computational Complexity (ECCC) 22: 126 (2015))

Salil P. Vadhan. 2003. Constructing locally computable extractors and cryptosystems in the bounded-storage model. *J. Cryptology* 17, 1 (2004), 43–77. (also in Crypto 2003)

Gregory Valiant and Paul Valiant. 2016. Information theoretically secure databases. *Electronic Colloquium on Computational Complexity (ECCC)* 23 (2016), 78.

Ryan Williams. 2006. Inductive time-space lower bounds for sat and related problems. *Comput. Complex.* 15, 4 (2006), 433–470.

Ryan Williams. 2007. Time-space tradeoffs for counting np solutions modulo integers. In *IEEE Conference on Computational Complexity* (2007), 70–82.

Received June 2017; revised August 2018; accepted August 2018