JMLR: Workshop and Conference Proceedings vol 49:1–22, 2016

A Guide to Learning Arithmetic Circuits

Ilya Volkovich ILYAVOL@UMICH.EDU
Department of EECS, CSE Division, University of Michigan, Ann Arbor, MI.

Abstract

An *arithmetic circuit* is a directed acyclic graph in which the operations are $\{+, \times\}$. In this paper, we exhibit several connections between learning algorithms for arithmetic circuits and other problems. In particular, we show that:

- Efficient learning algorithms for arithmetic circuit classes imply explicit exponential lower bounds.
- General circuits and formulas can be learned efficiently with membership and equivalence queries iff they can be learned efficiently with membership queries only.
- Low-query learning algorithms for certain classes of circuits imply explicit rigid matrices.
- Learning algorithms for multilinear depth-3 and depth-4 circuits must compute square roots.

1. Introduction

Revealing a hidden function from a set of examples is a natural and basic problem referred to as learning. The purpose of this paper is to serve as a guide for learning arithmetic circuits by pointing out the obstacles along the path of coming up with efficient learning algorithms for these classes of functions. We achieve this goal by showing a relation between learning and other problems related to arithmetic circuits.

1.1. Arithmetic Circuits

Arithmetic circuits are the standard computational model for computing polynomials. A circuit C in the variables $X = \{x_1, \ldots, x_n\}$ over the field $\mathbb{F}$ is a labelled directed acyclic graph. The inputs (nodes of in-degree zero) are labelled by variables from X or by constants from the field. The internal nodes are labelled by $+$ or $\times$, computing the sum and product, resp., of the polynomials on the tails of incoming edges (subtraction is obtained using the constant -1). A *formula* is a circuit whose nodes have out-degree one (namely, a tree). The output of a circuit (formula) is the polynomial computed at the output node. The *size* of a circuit (formula) is the number of gates in it. The *depth* of the circuit (formula) is the length of a longest path between an input node and the output node.

A polynomial is *multilinear* if its linear in each of its variables. In other words, no variable appears in a power of two or higher. A circuit is *multilinear* if every gate in the circuit computes a multilinear polynomial (not just the output gate). This implies that each multiplication gate in the circuit must compute a product of two polynomials defined over disjoint sets of variables (in short: variable-disjoint polynomials). A circuit is *set-multilinear*

if there exists a partition of the variables $X_1 \cup \ldots \cup X_\ell$ such that no multiplication gate computes a product of two polynomials containing variables for the same set. This property is a relaxation of multilinearity, as a multilinear polynomial is a set-multilinear polynomials w.r.t to the singletons (i.e. $X_i = \{x_i\}$). Particular examples of set-multilinear polynomials are the Permanent and the Determinant.

Sometimes we shall think of an arithmetic circuit and of the polynomial that it computes as the same objects. In particular, the degree of a circuit is defined as the degree of the polynomial computed by the circuit. Finally, we shall say that C is an (n, s, d)-*circuit* if it is an n-variate arithmetic circuit of size s and of degree at most d.

1.2. Arithmetic Circuit Classes

In this paper we will usually refer to a class of arithmetic circuits $\mathcal{C}$. It should be thought of as either the class of general circuits or formulas, of as some restricted class. In particular, we will be interested in the following classes:

- s-sparse polynomial is a polynomial that has at most s (non-zero) monomials.
- Depth-3 $\Sigma\Pi\Sigma(k)$ circuits computes polynomials of the form $C(\bar{x}) = \sum_{i=1}^{k} \prod_{j=1}^{d_i} L_{ij}(\bar{x})$ where L_{ij}-s are linear forms. That is, $L_{ij}(\bar{x}) = \alpha_1 x_1 + \alpha_2 x_2 + ... + \alpha_n x_n + \alpha_0$.
- Depth-4 $\Sigma\Pi\Sigma\Pi(k)$ circuits computes polynomials of the form $C(\bar{x}) = \sum_{i=1}^{k} \prod_{j=1}^{d_i} P_{ij}(\bar{x})$ where P_{ij}-s are sparse polynomials.
- Read-once Oblivious Algebraic Branching Programs (ABPs) compute polynomials of the form $C(\bar{x}) = \bar{v}^t \cdot D_n(x_{\sigma(n)}) \cdot D_{n-1}(x_{\sigma(n-1)}) \cdot \ldots \cdot D_1(x_{\sigma(1)}) \cdot \bar{u}$ when $\sigma : [n] \to [n]$ is permutation, $\bar{v}$ and $\bar{u}$ are vectors of field elements, and each $D_i(z)$ is a matrix with entries being polynomials in the variable z.

1.3. Exact Learning

Angluin's model of Exact Learning Angluin (1988) consists of a (computationally-bounded) learner and an (all-powerful) teacher. The learner's goal is to output a target function f from a given function class $\mathcal{C}$. To do so, the learner is allowed to query the value $f(\bar{x})$ on any input $\bar{x}$ (membership query). In addition, the learner is also allowed to propose a hypothesis $\hat{f}$ and ask the teacher whether it is equivalent to f (equivalence query). If this is indeed the case, the learner has achieved his goal. Otherwise, the teacher presents the learner with a counterexample $\bar{a}$ for which $f(\bar{a}) \neq \hat{f}(\bar{a})$. We say that a function class $\mathcal{C}$ is *exactly learnable* if there exists a learner which given any $f \in \mathcal{C}$, in time polynomial in n and $|f|$ (the size of f in the representation scheme) outputs a hypothesis $\hat{f}$ such that $f(\bar{x}) = \hat{f}(\bar{x})$ for all $\bar{x}$, using membership and equivalence queries.

1.4. Polynomial Identity Testing

Let $\mathcal{C}$ be a class of arithmetic circuits defined over some field $\mathbb{F}$. The polynomial identity testing problem (PIT for short) for $\mathcal{C}$ is the question of deciding whether a given circuit from $\mathcal{C}$ computes the identically zero polynomial. This question can be considered both in the white-box and the black-box models. In the white-box model, the circuit is given to

us as an input. In the black-box model, we can only access the polynomial computed by the circuit using (membership) queries. It is easy to see that a deterministic black-box PIT algorithm is equivalent to a "hitting set". That is, a set of points $\mathcal{H}$ such that for every $C \in \mathcal{C}$: $C \equiv 0$ iff $C|_{\mathcal{H}} \equiv 0$. In other words, C is identically zero iff it evaluates to zero on every point in $\mathcal{H}$.

The importance of this fundamental problem stems from its many applications. For example, the deterministic primality testing algorithm of Agrawal et al. (2004) and the fast parallel algorithm for perfect matching of Mulmuley et al. (1987) are based on solving PIT problems.

PIT has a well-known randomized algorithm Schwartz (1980); Zippel (1979); DeMillo and Lipton (1978) by evaluating the circuit at a random point. However, we are interested in the problem of obtaining efficient *deterministic* algorithms for it. This question has received a lot of attention recently but its deterministic complexity is still far from being well-understood. For a more detailed discussion, we refer the reader to the excellent survey Shpilka and Yehudayoff (2010).

1.5. Polynomial Reconstruction

The *reconstruction* problem for arithmetic circuits can be seen as an algebraic specialization of the learning problem. Given a black-box (i.e. oracle), access to a degree d polynomial $P \in \mathbb{F}[x_1, x_2, \ldots, x_n]$ computable by an arithmetic circuit of size s from a circuit class $\mathcal{C}$, output a circuit from $C \in \mathcal{C}$ that computes P. More generally, the algorithm is allowed to output a circuit of size $\text{poly}(s)$. Similar to the learning scenario, an efficient reconstruction algorithm must run in time polynomial in n, s, d.

A key difference between learning and reconstruction is that in the reconstruction problem we seek to output a circuit that computes the target polynomial as a formal sum of monomials and not just as a function. This poses an implicit requirement that the underlying field $\mathbb{F}$ is larger than the degree of the polynomial it is trying to reconstruct. Otherwise, the algorithm is allowed to query the polynomial on a sufficiently (polynomially) large extension field. Particular examples include reconstruction algorithms for sparse polynomials Klivans and Spielman (2001); Lipton and Vishnoi (2003), $\Sigma\Pi\Sigma(k)$ circuits Karnin and Shpilka (2009) and read-once formulas Bshouty et al. (1995); Shpilka and Volkovich (2014).

In this regard, a natural question is the dependence of the running time on the bit-complexity of the coefficients in the underlying polynomials. For finite fields, this bit-complexity corresponds with the log of the field size. Therefore, for simplicity of notation, we will denote by $\log|\mathbb{F}|$ the bit-complexity of the coefficients even if the field is infinite. Going back to the question, ideally, we would like the running time of the algorithm to be polynomial in the bit-complexity, that is $\text{poly}(\log|\mathbb{F}|)$. Unfortunately, this is not always the case (see e.g. Karnin and Shpilka (2009)). For a more detailed discussion, we refer the reader to the excellent survey Shpilka and Yehudayoff (2010).

1.6. Elimination of Equivalence Queries in the Arithmetic Setting

Unlike the Boolean setting, where the equivalence queries are essential (see Angluin (1987)), in the arithmetic setting, an efficient black-box PIT algorithm can be used to eliminate them. More specifically, let $\mathcal{H}$ be a hitting set for $\mathcal{C}$ and let $C \in \mathcal{C}$ be the target circuit. Given an

equivalence query $\hat{C}$, the learning algorithm tests if $\hat{C}|_{\mathcal{H}} \equiv C|_{\mathcal{H}}$, using membership queries to C. If $\hat{C}(\bar{a}) \neq C(\bar{a})$ for some $\bar{a} \in \mathcal{H}$, the algorithm answers the query with $\bar{a}$. Otherwise, the algorithm stops and outputs $\hat{C}$.

Of course, the above works if $\mathcal{C}$ is closed under subtraction, that is $C - \hat{C} \in \mathcal{C}$, since by definition $C \equiv \hat{C}$ iff $C|_{\mathcal{H}} \equiv \hat{C}|_{\mathcal{H}}$. This is typically the case when the learning algorithm is proper[1]. For example, Beimel et al. Beimel et al. (2000) gave a proper exact learning algorithm for read-once oblivious ABPs. Later, Forbes & Shpilka Forbes and Shpilka (2013) devised an efficient black-box PIT algorithm for that class of circuits. As read-once oblivious ABPs are closed under subtraction, this results in a learning algorithm for read-once oblivious ABPs that uses membership queries only. In the case of general circuits, one can compare $\hat{C}$ and C by evaluating them at random point $\bar{a}$, applying the Schwartz-Zippel Lemma (see Lemma 19). Yet, this will result in a randomized learning algorithm.

1.7. Circuit Lower Bounds

Attaining explicit lower bounds for both arithmetic and Boolean circuits is one of the biggest challenges in the theory of computation and has been the focus of much research Baur and Strassen (1983); Raz et al. (2008); Håstad (1998); Lachish and Raz (2001); Raz (2009). Counting arguments show that the vast majority of the functions require circuits of high complexity. Yet, laying our hands on a specific function remains elusive. Attaining strong lower bounds have direct implications to the P vs NP question, as well as to questions related to derandomization.

Let n denote the number of variables. It is not hard to show lower bounds for polynomials of high degree (i.e. $d = 2^n$). Therefore, the goal is to prove lower bounds for polynomials of low degree, (typically $d = \text{poly}(n)$). More precisely, what we are after is a low-degree polynomial computable in time $2^{\mathcal{O}(n)}$ that requires large arithmetics circuits.

So far, the best known lower bounds for arithmetic circuits are $\Omega(n \log n)$, due to Baur & Strassen Baur and Strassen (1983). When restricting ourselves to multilinear circuits, we can get better bounds: $\Omega(n^{4/3}/\log^2 n)$ for multilinear circuits due to Raz et al. Raz et al. (2008), and $n^{\Omega(\log n)}$ for multilinear formulas due to Raz Raz (2009). It is an interesting open question to improve any of these bounds.

1.8. Matrix Rigidity

The notion of matrix rigidity was introduced by Valiant in Valiant (1977). For $r, s \in \mathbb{N}$, we say that a matrix S is *s-sparse* if each row of the matrix contains at most s non-zero entries. A matrix $A \in \mathbb{F}^{m \times n}$ is (r,s)-*rigid* if A cannot be written as a sum of two matrices $A = L + S$ such that $\text{rank}(L) \leq r$ and S is s-sparse. In the same paper, Valiant showed that if a matrix A with $m = \mathcal{O}(n)$ is $(\Omega(n), n^{\varepsilon})$-rigid then the linear map $A \cdot \bar{x}$, induced by A, cannot be computed by a circuit of size $\mathcal{O}(n)$ and depth $\mathcal{O}(\log n)$. While it is easy to see that a random matrix satisfies the above conditions, a big body of research was invested in attempts to construct rigid matrices explicitly. Yet, the best known result is due to Saraf & Yekhanin Saraf and Yekhanin (2011) that gives a construction of $(n/2, s)$-rigid matrices with $m = \exp(s) \cdot n$. For a further discussion of the work on matrix rigidity, we refer the reader to the excellent survey Lokam (2009).

1. i.e. when the hypotheses of the learning algorithm come from the class it is trying to learn.

1.9. Square Root Extraction

Given $\beta \in \mathbb{F}$, output $\alpha \in \mathbb{F}$ such that $\alpha^2 = \beta$. In addition to being a natural problem, square root extraction plays an import role in cryptography Goldwasser and Micali (1984); Rabin (2005). The best known *deterministic* root extraction algorithms over the finite fields have polynomial dependence on $|\mathbb{F}|$, i.e. exponential in the bit-complexity of the coefficients (see e.g. Shoup (1991); Gathen and Gerhard (1999); Gao et al. (2004); Kayal (2007))[2]. In other words, the best known algorithm is roughly as efficient as the naive brute-force algorithm. While in the *randomized* setting, this dependence is polynomial in $\log |\mathbb{F}|$. In particular, there is no known efficient deterministic root extraction algorithm when $\mathbb{F}$ is large. One reason for that is the two-way connection between modular square root extraction and finding quadric non-residues. Over fields with characteristic 0 (e.g. $\mathbb{Q}$), both the *deterministic* and the *randomized* complexities are polynomial in $\log |\mathbb{F}|$ (see e.g. Lenstra et al. (1982)).

1.10. Our Results

We begin with our main result: exact learning algorithms for arithmetic circuit classes imply explicit exponential lower bounds. The result is obtained by combining and extending the techniques of Klivans et al. (2013) with the techniques of Heintz and Schnorr (1980); Agrawal (2005). In Appendix A, we show a folklore result that PAC and Exact learning models are essentially equivalent for arithmetic circuits. For that reason, we are not considering the PAC model in our paper.

Theorem 1 *Let $\mathcal{C}$ be an arithmetic circuit class over the field $\mathbb{F}$. If $\mathcal{C}$ is exactly learnable then for every $n \in \mathbb{N}$, there exists an explicit[3] multilinear polynomial $P \in \mathbb{F}[x_1, x_2, \ldots, x_n]$ that requires circuits of size $2^{\Omega(n)}$ from $\mathcal{C}$.*

We remark that the learning algorithm need not be proper. The only implicit assumption is that the functions posed as equivalence queries by the learning algorithm can be evaluated efficiently from their description. This is a typical situation with all the existing exact learning algorithms.

Using the connection between lowers bounds and PIT algorithms Kabanets and Impagliazzo (2004), as well as depth reduction techniques Agrawal and Vinay (2008), we are able to convert efficient learning algorithms for certain (relatively general) circuit classes into efficient PIT algorithms.

Theorem 2 *Let $\mathcal{C} \in \{$general circuits, general formulas, $\Sigma\Pi\Sigma\Pi$ circuits$\}$ be an arithmetic circuit class over the field $\mathbb{F}$. If $\mathcal{C}$ is exactly learnable then there exists a quasi-polynomial time black-box PIT algorithm for the class of general arithmetic circuits.*

In the case when the learning algorithm is proper or, more generally, outputs circuits of polynomial size and degree as its hypotheses, we can eliminate the equivalence queries by implementing the procedure outlined in Section 1.6.

2. In fact, for fields of size p^e this dependence is polynomial in p.
3. Computable in time $2^{\mathcal{O}(n)}$.

Theorem 3 *Let $\mathcal{C} \in \{$general circuits, general formulas, $\Sigma\Pi\Sigma\Pi$ circuits$\}$ be an arithmetic circuit class over the field $\mathbb{F}$. If $\mathcal{C}$ is exactly learnable with hypotheses being general circuits of polynomial size and degree then $\mathcal{C}$ is learnable with membership queries only in quasi-polynomial time. In addition, if the algorithm is proper, then $\mathcal{C}$ can be reconstructed in quasi-polynomial time.*

We see this theorem as evidence that reconstruction is, perhaps, a more natural framework for learning arithmetic circuits.

Zeev Dvir pointed out that the same techniques can be used to show that "low-query" exact learning algorithms for a certain class of multilinear polynomials give rise to explicit rigid matrices.

Theorem 4 *Let $r, s \in \mathbb{N}$. Let $\mathcal{C}_{(r,s)}$ be a class of multilinear polynomials of the form:*

$$\mathcal{C}_{(r,s)} = \left\{ P(\bar{x}, \bar{y}) = \bar{x}^t \cdot L \cdot \bar{y} + \bar{x}^t \cdot S \cdot \bar{y} \;\middle|\; L, S \in \mathbb{F}^{m \times n},\ \mathrm{rank}(L) \leq r,\ S \text{ is } s\text{-sparse.} \right\}$$

If $\mathcal{C}_{(r,s)}$ is exactly learnable with $< mn$ queries, then there exist explicit (r, s)-rigid matrices.

Combined with Valiant (1977), this result has the same flavor as the results of Fortnow and Klivans (2009); Klivans et al. (2013) and Theorem 1: learning algorithms imply lower bounds. Using the same techniques, one could show that devising a learning algorithm with $< nm$ queries for the class of linear maps computable by circuits of size $\mathcal{O}(n)$ and depth $\mathcal{O}(\log n)$ would entail the same lower bounds. We would like to point out that while the learning algorithms can output the polynomials in any representation, the bound on the number of queries is tight since any $m \times n$ matrix is learnable using mn membership queries.

On a different note, we study the question of the dependence of the running time of the learning/reconstruction algorithm on the field size ($|\mathbb{F}|$). In Karnin and Shpilka (2009), a deterministic reconstruction algorithm for $\Sigma\Pi\Sigma(k)$ circuits was given. The running time of that algorithm has polynomial dependence on the field size. In addition, in Gupta et al. (2012), a randomized reconstruction algorithm for multilinear $\Sigma\Pi\Sigma\Pi(2)$ circuits was given with polylogarithmic dependence on the field size. Recently, this algorithm was derandomized in Volkovich (2015) preserving the polylogarithmic dependence on the field size. The only additional assumption was that the algorithm is given a square root oracle[4]. As we can implement such an oracle in deterministic $\mathrm{poly}(|\mathbb{F}|)$ time (see Section 1.9), this results in a deterministic reconstruction algorithm with a polynomial dependence on $|\mathbb{F}|$. To the best of our knowledge, there is no known deterministic reconstruction algorithm for multilinear $\Sigma\Pi\Sigma\Pi(2)$ circuits with a sub-polynomial dependence on $|\mathbb{F}|$. Moreover, this still holds true for multilinear $\Sigma\Pi\Sigma(2)$ circuits and even set-multilinear $\Sigma\Pi\Sigma(2)$ circuits[5]. Our next theorem suggests an explanation for this state of affairs. We show that any proper learning/reconstruction algorithm for these classes of circuits must compute square roots along its execution.

Theorem 5 *Let $k \in \mathbb{N}$. Suppose the class of (set)-multilinear $\Sigma\Pi\Sigma(2)$ or $\Sigma\Pi\Sigma\Pi(2)$ circuits over the field $\mathbb{F}$ is exactly learnable with hypotheses being multilinear $\Sigma\Pi\Sigma\Pi(k)$ circuits*

4. An algorithm that can extract square roots of field elements efficiently.
5. The learning algorithm of Beimel et al. (2000); Klivans and Shpilka (2006) for set-multilinear $\Sigma\Pi\Sigma$ circuits outputs read-once oblivious ABPs as hypotheses.

of polynomial size. Then in time polynomial in $\log|\mathbb{F}|$ *the exact learning algorithm can be deterministically converted into a square root oracle over* $\mathbb{F}$*.*

Given the state of our current techniques (see Section 1.9), the theorem suggests that any reconstruction algorithm for these classes of circuits must be either randomized or have a polynomial dependence on the field size. This partially answers an open question of Gupta et al. (2012) that asks whether it is possible to remove the polynomial dependence on the field size from the reconstruction algorithm of Karnin and Shpilka (2009) even for $\Sigma\Pi\Sigma(2)$ circuits.

1.11. Techniques

In this section, we outline the techniques we use to prove our results.

1.11.1. From Exact Learning to Lower Bounds

We combine and extend the techniques of Klivans et al. (2013) with the techniques of Heintz and Schnorr (1980); Agrawal (2005). Given an exact learner $\mathcal{A}$, the lower bounds of Klivans et al. (2013) are obtained by running $\mathcal{A}$ and diagonalizing against its queries, thus producing a function that disagrees with the output of $\mathcal{A}$ on at least one point. Applying the same idea naively in the arithmetic setting may result in a polynomial of a high degree. Our goal would be to produce a multilinear polynomial P that disagrees with the output of $\mathcal{A}$.

In order to maintain the multilinearity of P we use a version of the evaluation matrix that was defined in Heintz and Schnorr (1980); Agrawal (2005). Given an assignment $\bar{a} \in \mathbb{F}^n$, the corresponding row $M(\bar{a})$ of the evaluation matrix is the 2^n-length vector containing the evaluations of all the multilinear monomials on $\bar{a}$ (see Definition 4). This matrix represents the linear map of evaluating a polynomial on a given set of assignments. Specifically, taking the product of $M(S)$ with a coefficient vector of a multilinear polynomial results in the vector of evaluations of the polynomial on the assignments in S. On the other hand, a full-rank $M(S)$ determines a unique multilinear polynomial with the coefficients being the solution vector of the corresponding system of linear equations. We use the evaluation matrix to ensure that the diagonalizing values of the polynomial do not violate the multilinearity condition by keeping that the rows of $M(S)$ independent throughout the execution of the algorithm.

Given a membership query $\bar{a}$ of $\mathcal{A}$, the algorithm first checks if the value of the polynomial P on $\bar{a}$ is predefined by the answers to the previous queries. This is done by checking the linear dependence between the row vector $M(\bar{a})$ and the rows of $M(S)$. If the value is predefined, the algorithm computes $P(\bar{a})$ an returns it to $\mathcal{A}$. Otherwise, the algorithm declares $P(\bar{a}) = 0$ and adds the row vector $M(\bar{a})$ to $M(S)$. Given an equivalence query h of $\mathcal{A}$, the algorithms finds a point $\bar{a} \in \mathbb{F}^n$ such that the row vector $M(\bar{a})$ is independent of the rows of $M(S)$. Given such $\bar{a}$, the algorithm diagonalizes against h by declaring $P(\bar{a}) = h(\bar{a}) - 1$ and returning $\bar{a}$ to $\mathcal{A}$ as a counterexample.

We note that the rank of the matrix $M(S)$ equals to the number q of queries posed by $\mathcal{A}$. Therefore, as long as $q < 2^n$, there always exists $\bar{a} \in \mathbb{F}^n$ such that row vector $M(a)$ is independent of the rows $M(S)$. Moreover, we show that such $\bar{a}$ actually exists in $\{0,1\}^n$. This allows the algorithm to be independent of the field size and, in particular, handle infinite fields. For more details, see Section 3.

Using the connection between lowers bounds and PIT algorithms Kabanets and Impagliazzo (2004), as well as depth reduction techniques Agrawal and Vinay (2008), we are able to convert efficient exact learning algorithms into efficient black-box PIT algorithms. Later on, we use the PIT algorithms to eliminate the equivalence queries.

1.11.2. From Exact Learning to Square Roots

For every $\alpha \in \mathbb{F}$, we construct a family of polynomials $\{P^n_\alpha(\bar{x})\}_{n\in\mathbb{N}}$ computable by set-multilinear $\Sigma\Pi\Sigma(2)$ circuits, that can be evaluated efficiently given α^2 (without the knowledge of α). Indeed, for every $\alpha \in \mathbb{F}$ and $n \in \mathbb{N}$ the polynomials $P^n_\alpha(\bar{x})$ and $P^n_{-\alpha}(\bar{x})$ will be identical. However, for a constant k and sufficiently large n, every (set)-multilinear $\Sigma\Pi\Sigma(k)$ or even polynomial-size $\Sigma\Pi\Sigma\Pi(k)$ circuit that computes P^n_α must have a factor of the form $x_j \pm \alpha$ in one of its multiplication gates. Consequently, any proper exact learning/reconstruction algorithm $\mathcal{A}$ for these classes of circuits must extract square roots. Moreover, we show how to use such an $\mathcal{A}$ as a subroutine, in order to actually compute a square root given $\alpha^2 = \beta \in \mathbb{F}$.

1.12. Related Work

Fortnow & Klivans Fortnow and Klivans (2009) initiated a body of work dedicated to the study of the relations between learning algorithms and circuit lower bounds. In the same paper it was shown that an efficient exact learning algorithm for a Boolean circuit class $\mathcal{C}$ implies that the complexity class $\mathsf{EXP}^{\mathsf{NP}}$ requires circuits of super-polynomial size from $\mathcal{C}$. Following a line of improvements, Klivans et al. Klivans et al. (2013) replaced $\mathsf{EXP}^{\mathsf{NP}}$ by $\mathsf{DTIME}(n^{\omega(1)})$. In the arithmetic setting, the best result Fortnow and Klivans (2009) gives super-polynomial lower bounds against the complexity class $\mathsf{ZPEXP}^{\mathsf{RP}}$.

Our improvement, therefore, is two-fold: First, we obtain a quantitative improvement replacing $\mathsf{ZPEXP}^{\mathsf{RP}}$ by a subclass EXP and getting an exponential lower bound as opposed to super-polynomial. Second, our lower bound is more structured and constructive since we give an explicit multilinear polynomial, while the result of Fortnow and Klivans (2009) is existential.

Bshouty et al. Bshouty et al. (1995) gave deterministic exact learning and randomized reconstruction algorithms for read-once formulas with addition, multiplication and division gates. Their algorithm is assumed to be given a square root oracle. They also show that any reconstruction algorithm for read-once formulas with these operations must compute square roots. Recently, Shpilka & Volkovich Shpilka and Volkovich (2014) gave a deterministic reconstruction algorithm for read-once formulas with addition and multiplication gates (no division) which has a polynomial dependence on $\log(|\mathbb{F}|)$. This demonstrates that the need for square root extraction is not necessary if we choose to deal with polynomials rather than rational functions. However, using the ideas of Section 5 with the techniques of Anderson et al. (2015); Shpilka and Volkovich (2015), one could show that any reconstruction algorithm for read-twice formulas or even sums of read-once formulas with addition and multiplication gates must compute square roots.

Several other results Valiant (1984); Kearns and Valiant (1994); Klivans and Sherstov (2009); Gentry and Halevi (2011) exhibit a two-way connection between learning and cryptography: basing the hardness of learning on cryptography and constructing cryptographic

primitives based on the hardness of learning. In addition, Daniely and Shalev-Shwartz (2014); Daniely (2015) show hardness of learning DNF formulas and halfspaces based on another complexity theoretic assumption regarding hardness of refutation of random k-SAT formulas. These results are incomparable as the hardness of showing lower bounds is independent of the question whether one-way functions exist and/or the assumption above. For a more detailed discussion, we refer the reader to Impagliazzo (1995).

A recent result of Carmosino et al. (2016) establishes a connection between learning and circuit lower bounds in the opposite direction. It is shown that lower bounds of a certain kind (i.e. natural properties) for a class of functions imply randomized learning algorithms for that same class.

1.13. Organization

The paper is organized as follows: We begin by some basic definitions and notation in Section 2, where we also introduce our main tool. In Section 3, we prove our main result (Theorem 1): exact learning algorithms for arithmetic circuit classes imply explicit exponential lower bounds, as well as Theorems 2 and 3. Next in Section 4, we show how the techniques can be modified to prove Theorem 4. In Section 5, we show that proper learning algorithms for certain circuit classes must compute square roots, thus proving Theorem 5. We conclude with discussion & open questions in Section 6. In Appendix A, we show that PAC and Exact learning models are essentially equivalent for arithmetic circuits.

2. Preliminaries

For a positive integer n, we denote $[n] = \{1, \ldots, n\}$. For a polynomial $P(x_1, \ldots, x_n)$, a variable x_i and a field element α, we denote with $P|_{x_i=\alpha}$ the polynomial resulting from setting $x_i = \alpha$. Given a subset $I \subseteq [n]$ and an assignment $\bar{a} \in \mathbb{F}^n$, we define $P|_{\bar{x}_I=\bar{a}_I}$ to be the polynomial resulting from setting $x_i = a_i$ for every $i \in I$. We often denote variables interchangeably by their index or by their label: i versus x_i. For $\bar{v} \in \mathbb{N}^n$ we define $\bar{a}^{\bar{v}} \triangleq \prod_{i=1}^{n} a_i^{v_i}$, where $0^0 \triangleq 1$.

2.1. Matrices

In this section we give definitions related to matrices and prepare the ground for our main tool. For more details about matrices, we refer the reader to Horn and Johnson (1991).

Lemma 1 *Let $A \in \mathbb{F}^{n \times m}$ and $\bar{v} \in \mathbb{F}^n$. There exists an algorithm that given A and $\bar{v}$ solves the system $A \cdot \bar{b} = \bar{v}$ using $\mathrm{poly}(n, m)$ field operations. That is, the algorithm returns a solution if one exists, or $\bar{b} = \perp$ otherwise.*

Definition 2 (Tensor Product) *Let $A \in \mathbb{F}^{m \times n}$ and $B \in \mathbb{F}^{k \times \ell}$. Then the* tensor product of A and B, *$A \otimes B \in \mathbb{F}^{mk \times n\ell}$ is defined as:*

$$A \otimes B = \begin{bmatrix} a_{11}B & \ldots & a_{1n}B \\ \vdots & & \vdots \\ a_{1m}B & \ldots & a_{mn}B \end{bmatrix}.$$

Fact 3 $\text{rank}(A \otimes B) = \text{rank}(A) \cdot \text{rank}(B)$.

We now introduce our main tool - the evaluation matrix. This tool has been previous used in Heintz and Schnorr (1980) and Agrawal (2005) to show that efficient black-box PIT algorithms imply circuit lower bounds. In this paper we will focus on evaluation matrices for multilinear polynomials. Let $n \in \mathbb{N}$ and $\bar{a} \in \mathbb{F}^n$. A row of such matrix is a (row) vector of length 2^n listing the evaluations of all n-variate multilinear monomials to $\bar{a}$. Formally:

Definition 4 (Evaluation Matrix) *Let $\bar{a} \in \mathbb{F}^n$ and $\bar{v} \subseteq \{0,1\}^n$. We define $M(\bar{a}) \in \mathbb{F}^{1 \times 2^n}$ as $M(\bar{a})_{1,\bar{v}} = \bar{a}^{\bar{v}}$. We now extend the definition to sets of assignments.*
Let $S \subseteq \mathbb{F}^n$. We define $M(S) \in \mathbb{F}^{|S| \times 2^n}$ as $M(S)_{\bar{a}} = M(\bar{a})$ for $\bar{a} \in S$, when $M(\emptyset) \in \mathbb{F}^{1 \times 2^n} \triangleq \bar{0}$.
We say that $\bar{a}$ is independent of S *if* $\text{rank}(M(S \cup \{\bar{a}\})) > \text{rank}(M(S))$.

To provide additional intuition, we list several important and useful properties of $M(S)$ which we will apply in our proofs.

Lemma 5 *Let $n \in \mathbb{N}$ and $S \subseteq \mathbb{F}^n$. Then we have the following:*

1. *Let $\bar{a} \in \mathbb{F}^n$ and $P \in \mathbb{F}[x_1, x_2, \ldots, x_n]$ be a multilinear polynomial. Denote by $\bar{c}$ the vector of coefficients of P. Then $P(\bar{a}) = M(\bar{a}) \cdot \bar{c}$.*
2. *$\bar{a}$ is independent of S iff $M(S)^t \cdot \bar{b} = M(\bar{a})^t$ has no solution.*
3. *Suppose $S = S_1 \times S_2 \times \cdots \times S_n$ when $S_1, S_2, \ldots S_n \subseteq \mathbb{F}$ are of size $|S_i| = 2$ for $i \in [n]$. Then $M(S) = M(S_1) \otimes M(S_2) \otimes \cdots \otimes M(S_n)$.*
4. $\text{rank}\left(M\left(\{0,1\}^n\right)\right) = 2^n$.
5. *Suppose $|S| < 2^n$. Then there exists $\bar{a} \in \{0,1\}^n$ such that $\bar{a}$ is independent of S. Moreover, given S such $\bar{a}$ can be found using $\text{poly}(n, |S|)$ field operations.*

Proof

1. By inspection.
2. $\text{rank}(M(S \cup \{\bar{a}\})) > \text{rank}(M(S))$ iff the (row) vector $M(\bar{a})$ can not be we written as a linear combination of the rows of $M(S)$.
3. By inspection. In terms of the dimensions, note that for each i: $S_i \subseteq \mathbb{F}^1$. Therefore, by definition $M(S_i) \in \mathbb{F}^{2 \times 2}$. While $S \subseteq \mathbb{F}^n$ and thus $M(S) \in \mathbb{F}^{2^n \times 2^n}$.
4. $M(\{0,1\}) = \begin{bmatrix} 1 & 0 \\ 1 & 1 \end{bmatrix}$. By the previous part: $\text{rank}(M(\{0,1\}^n)) = \text{rank}(M(\{0,1\}))^n = 2^n$.
5. Since $\text{rank}(M(S)) < 2^n$ and $\text{rank}(M(\{0,1\}^n)) = 2^n$, the existence of such $\bar{a} \in \{0,1\}^n$ follows from the exchange property of the rank. Therefore, we can find such $\bar{a}$, by going over all the assignments in $\{0,1\}^n$ and checking if $M(S)^t \cdot \bar{b} = M(\bar{a})^t$ has a solution (using Lemma 1).

■

3. From Learning to Lower Bounds

In this section we prove our main result (Theorem 1): exact learning algorithms for arithmetic circuit classes imply explicit exponential lower bounds, as well as Theorems 2 and 3. We begin with our main result.

Algorithm 1 Hard Polynomial from Exact Learner

Input: $n \in \mathbb{N}, \bar{\alpha} \in \mathbb{F}^n$, an exact learner $\mathcal{A}$, ℓ as in the statement of Lemma 6.
Output: $P(\bar{\alpha})$.

1 $S \leftarrow \emptyset$, $\bar{u}$ - a vector indexed by elements of S. That is, $\bar{u} \in \mathbb{F}^S$
Run $\mathcal{A}$ with $n, s = 2^{n/4\ell}$.
`/* Answering a membership query.` `*/`
2 Given a membership query point $\bar{a} \in \mathbb{F}^n$ by $\mathcal{A}$:
Compute $\bar{b}$ such that $M(S)^t \cdot \bar{b} = M(\bar{a})^t$ (using Lemma 1) **if** $\bar{b} = \perp$ **then** ($\bar{a}$ is independent of S)
3 $\quad S \leftarrow S \cup \{\bar{a}\}$; $\bar{u}(\bar{a}) \leftarrow 0$ Answer $\bar{u}(\bar{a})$ to $\mathcal{A}$
4 **else**
5 $\quad$ Answer $\bar{b}^t \cdot \bar{u}$ to $\mathcal{A}$
`/* Answering an equivalence query.` `*/`
6 Given an equivalence query hypothesis $h(\bar{x}) : \mathbb{F}^n \to \mathbb{F}$ by $\mathcal{A}$:
Find an assignment $\bar{a} \in \{0,1\}^n$ independent of S (using by Lemma 5) $S \leftarrow S \cup \{\bar{a}\}$; $\bar{u}(\bar{a}) \leftarrow h(\bar{a}) - 1$ Answer $\bar{a}$ to $\mathcal{A}$ `/* After the interaction with` $\mathcal{A}$`.` `*/`
7 **while** $|S| < 2^n$ **do** (complete the matrix to full rank)
8 $\quad$ Find an assignment $\bar{a} \in \{0,1\}^n$ independent of S (using by Lemma 5) $S \leftarrow S \cup \{\bar{a}\}$; $\bar{u}(\bar{a}) \leftarrow 0$
9 Compute $\bar{c}$ such that $M(S) \cdot \bar{c} = \bar{u}$ Output $M(\bar{\alpha}) \cdot \bar{c}$

Lemma 6 *Let $\mathcal{C}$ be an arithmetic circuit class over the field $\mathbb{F}$. Suppose there exist an exact leaner $\mathcal{A}$ for $\mathcal{C}$ that learns circuits from $\mathcal{C}$ in time s^ℓ for $\ell \in \mathbb{N}$ when $n \leq s$. Then for every $n \in \mathbb{N}$ in time $2^{\mathcal{O}(n)}$ Algorithm 1 computes a multilinear polynomial $P \in \mathbb{F}[x_1, x_2, \ldots, x_n]$ that requires circuits of size $2^{\Omega(n)}$ from $\mathcal{C}$, invoking $\mathcal{A}$ as a subroutine.*

Proof First, observe that throughout the execution of the algorithm the rows of $M(S)$ are linearly independent. Indeed, in the beginning S is empty. Going forward, the algorithm adds elements to S and thus rows to $M(S)$ (in Lines 3, 6 and 7) making sure the rank is increasing (Lemma 5). In addition, the algorithm adds elements to S only when $|S| < 2^n$. Indeed, elements are added following a query from $\mathcal{A}$. Yet, the number of queries it at most $s^\ell \leq 2^{n/4}$. Consequently, in Line 9, $|S| = 2^n$ and $M(S)$ is a full rank $2^n \times 2^n$ matrix. Note the that algorithm computes the same (final) set S and vector $\bar{u}$ in every execution. Let us denote them by S_f and u_f, respectively. Therefore, the exists a unique vector $\bar{c} \in \mathbb{F}^{2^n}$ such that $M(S_f) \cdot \bar{c} = \bar{u}_f$.

Next, we claim consistency with the queries to $\mathcal{A}$. Let $\bar{a} \in \mathbb{F}^n$ be a membership query point by $\mathcal{A}$. Consider S and $\bar{u}$ at the time of the query. If $\bar{a}$ was independent of S, then

$\bar{a} \in S_f$ and $\bar{u}_f(\bar{a}) = 0$. Therefore, $M(S_f) \cdot \bar{c} = \bar{u}_f$ implies that

$$P(\bar{a}) = M(\bar{a}) \cdot \bar{c} = \bar{u}_f(\bar{a}) = 0.$$

Otherwise, $M(S)^t \cdot \bar{b} = M(\bar{a})^t$ for some $\bar{b} \in \mathbb{F}^{|S|}$. As $S \subseteq S_f$, we can extend the vector $\bar{b}$ to $\bar{b}_e \in \mathbb{F}^{2^n}$ by adding zeros in the entries indexed by $S_f \setminus S$. Observe that

$$M(S_f)^t \cdot \bar{b}_e = M(S)^t \cdot \bar{b} \text{ and } \bar{b}_e^t \cdot \bar{u}_f = \bar{b}^t \cdot \bar{u}.$$

Therefore, we obtain:

$$P(\bar{a}) = M(\bar{a}) \cdot \bar{c} = \bar{b}^t \cdot M(S) \cdot \bar{c} = \bar{b}_e^t \cdot M(S_f) \cdot \bar{c} = \bar{b}_e^t \cdot \bar{u}_f = \bar{b}^t \cdot \bar{u}.$$

Let $h(\bar{x}) : \mathbb{F}^n \to \mathbb{F}$ be a hypothesis by $\mathcal{A}$ and let $\bar{a} \in \{0,1\}^n$ be the assignment computed in Line 6. Repeating the previous reasoning we get that:

$$P(\bar{a}) = M(\bar{a}) \cdot \bar{c} = \bar{u}_f(\bar{a}) = h(\bar{a}) - 1 \neq h(\bar{a}).$$

Finally, suppose P was computable by a circuit of size s from $\mathcal{C}$. By the definition, $\mathcal{A}$ should output a hypothesis h such that $h \equiv P$. However, as we saw previously, for every hypothesis h by $\mathcal{A}$ there exists an assignment $\bar{a} \in \{0,1\}^n$ such that $P(\bar{a}) \neq h(\bar{a})$ leading to a contradiction.

For the running time, by Lemmas 1 and 5, the algorithm uses $2^{\mathcal{O}(n)}$ field operations. Note that we implicitly assume that $h(\bar{a})$ can be computed efficiently given the description of $h(\bar{x})$, which is the typical case with all the existing algorithms. ■

Theorem 1 follows as a corollary of the Lemma. Next, we use the connection between lower bounds and PIT algorithms, and depth reduction techniques to obtain an efficient PIT algorithm, thus proving Theorem 2. We require the following results.

Lemma 7 (Kabanets and Impagliazzo (2004)) *Let $\mathbb{F}$ be a field. Suppose that for every $n \in \mathbb{N}$ there exists an explicit multilinear polynomial $P \in \mathbb{F}[x_1, x_2, \ldots, x_n]$ that requires general circuits of size $2^{\Omega(n)}$. Then there exists an algorithm that given $n, d, s \in \mathbb{N}$ in time $(nd)^{\mathcal{O}(\log s)}$ outputs a hitting set $\mathcal{H}$ for general (n, s, d)-circuits.*

Lemma 8 (Agrawal and Vinay (2008)) *Let $\mathbb{F}$ be a field. Suppose that a multilinear polynomial $P \in \mathbb{F}[x_1, x_2, \ldots, x_n]$ requires $\Sigma\Pi\Sigma\Pi$ circuits of size $2^{\Omega(n)}$. Then P also requires general circuits of size $2^{\Omega(n)}$.*

Theorem 2 follows as corollary from the following Lemma.

Lemma 9 *Suppose that the class $\mathcal{C} \in \{$general circuits, general formulas, $\Sigma\Pi\Sigma\Pi$ circuits$\}$ is exactly learnable. Then there exists an algorithm that given $n, d, s \in \mathbb{N}$ in time $(nd)^{\mathcal{O}(\log s)}$ outputs a hitting set $\mathcal{H}$ for general (n, s, d)-circuits.*

Proof By Theorem 1, for every $n \in \mathbb{N}$ there exists an explicit multilinear polynomial $P \in \mathbb{F}[x_1, x_2, \ldots, x_n]$ that requires circuits of size $2^{\Omega(n)}$ from $\mathcal{C}$. Since $\mathcal{C} \in \{$general circuits, general formulas, $\Sigma\Pi\Sigma\Pi$ by Lemma 8 each such $P \in \mathbb{F}[x_1, x_2, \ldots, x_n]$ actually requires general circuits of size $2^{\Omega(n)}$.

Finally, by Lemma 7 there exists an algorithm that given $n, d, s \in \mathbb{N}$ in time $(nd)^{\mathcal{O}(\log s)}$ outputs a hitting set $\mathcal{H}$ for general (n, s, d)-circuits. ■

We prove Theorem 3 by implementing the procedure outlined in Section 1.6.

Proof [Proof of Theorem 3.] Let $\mathcal{A}$ be an exact learner for $\mathcal{C}$ and let $C \in \mathcal{C}$ be a (n, s, d)-circuit from $\mathcal{C}$ given via oracle access. We run $\mathcal{A}$ on C. Given a membership query $\bar{a} \in \mathbb{F}^n$, we simply answer by $C(\bar{a})$. Now, suppose that we are given an equivalence query hypothesis $\hat{C}$. By the properties of $\mathcal{A}$, $\hat{C}$ is a $(n, \text{poly}(s), \text{poly}(d))$-circuit. As all the classes being considered are closed under subtraction, the same holds for true $C - \hat{C}$ as well. Let $\mathcal{H}$ be the hitting set from Lemma 9 with the appropriate parameters. We test if $\hat{C}|_{\mathcal{H}} \equiv C|_{\mathcal{H}}$ using membership queries to C. If $\hat{C}(\bar{a}) \neq C(\bar{a})$ for some $\bar{a} \in \mathcal{H}$, then we answer the query with $\bar{a}$. Otherwise, the we stop and output $\hat{C}$. ■

4. From Learning to Rigidity

In this section we discuss the proof Theorem 4. In similar fashion to Definition 4, we can define a slightly different evaluation matrix.

Definition 10 *Let $\bar{a} \in \mathbb{F}^m$, $\bar{b} \in \mathbb{F}^n$ and $i \in [m], j \in [n]$. Define $\hat{M}(\bar{a}, \bar{b}) \in \mathbb{F}^{1 \times mn}$ as $\hat{M}(\bar{a}, \bar{b})_{1,(i,j)} = \bar{a}_i \cdot \bar{b}_j$. With the extension: for $S \subseteq \mathbb{F}^n \times \mathbb{F}^m$, $\hat{M}(S) \in \mathbb{F}^{|S| \times mn}$ when $\hat{M}(S)_{(\bar{a},\bar{b})} = \hat{M}(\bar{a}, \bar{b})$ for $(\bar{a}, \bar{b}) \in S$.*

In the spirit of Lemma 5, we observe that:

1. Given $\bar{a} \in \mathbb{F}^m$, $\bar{b} \in \mathbb{F}^n$ and a matrix $A \in \mathbb{F}^{m \times n}$ we have that $\bar{a}^t \cdot A \cdot \bar{b} = \hat{M}(\bar{a}, \bar{b}) \cdot v(A)$, when $v(A)$ is a vector of length mn containing the entries of A indexed by (i, j).
2. Let $S = \{(e_i, e_j)\}_{i \in [m], j \in [n]}$ when e_i represents the i-th vector of the standard basis. Then $\text{rank}(\hat{M}(S)) = mn$.

Given the above, one can use an argument similar to the one in Lemma 6 and Algorithm 1 to show that if $\mathcal{C}_{(r,s)}$ is exactly learnable with $q < mn$ queries, then the rank of the corresponding evaluation matrix will be q. Therefore, the algorithm could diagonalize against the output hypothesis (Lines 6 and 6), thus producing a vector which corresponds to a (r, s)-rigid matrix. We leave the formalization of the proof of Theorem 4 as an exercise for the reader.

5. From Learning to Square Roots

In this section, we establish a connection between learning and square root extraction, thus proving Theorem 5.

We begin by formally presenting the models of depth-4 and depth-3 multilinear circuits and some related definitions. Similar definitions were given in Dvir and Shpilka (2006); Saraf and Volkovich (2011); Karnin et al. (2013).

Definition 11 *A depth-4 $\Sigma\Pi\Sigma\Pi(k)$ circuit C has four layers of alternating Σ and Π gates (the top Σ gate is at level one) and it computes a polynomial of the form*

$$C(\bar{x}) = \sum_{i=1}^{k} F_i(\bar{x}) = \sum_{i=1}^{k} \prod_{j=1}^{d_i} P_{ij}(\bar{x})$$

where the $P_{ij}(\bar{x})$-s are polynomials computed by the last two layers of $\Sigma\Pi$ gates of the circuit and are the inputs to the Π gates at the second level.

A *multilinear* $\Sigma\Pi\Sigma\Pi(k)$ circuit is a $\Sigma\Pi\Sigma\Pi(k)$ circuit in which each multiplication gate F_i computes a multilinear polynomial. The requirement that the F_i-s compute multilinear polynomials implies that for each $i \in [n]$ the polynomials $\{P_{ij}\}_{j\in[d_i]}$ are variable-disjoint. In other words, each F_i induces a partition of the input variables. Yet in general, different multiplication gates may induce different partitions. A *set-multilinear* $\Sigma\Pi\Sigma\Pi(k)$ circuit is a multilinear $\Sigma\Pi\Sigma\Pi(k)$ circuit in which all the F_i-s induce the same partition.

Note that if the circuit is of size s then each P_{ij} is s-sparse. A depth-3 $\Sigma\Pi\Sigma(k)$ circuit can be a seen a special case of $\Sigma\Pi\Sigma\Pi(k)$ circuit in which the polynomials P_{ij} are linear forms (in particular, n-sparse).

We say that a circuit is *minimal* if no proper subsum of the F_i-s computes the zero polynomial. In other words, C cannot be "shrunk". We say that the circuit C is *simple* if $\gcd(F_1, \ldots, F_k) = 1$. Saraf & Volkovich showed that the multiplication gates of a multilinear depth-4 computing the zero polynomial must compute sparse polynomials. This is referred to as the "Sparsity Bound" for multilinear $\Sigma\Pi\Sigma\Pi(k)$ circuits. Previously, Dvir & Shpilka Dvir and Shpilka (2006) proved a similar statement for $\Sigma\Pi\Sigma(k)$ circuits. (See Dvir and Shpilka (2006) for more details).

Lemma 12 (Saraf and Volkovich (2011)) *There exists an non-decreasing function $\varphi(k, s)$ such that if $C(\bar{x}) = \sum_{i=1}^{k} F_i(\bar{x})$ is a simple and minimal, multilinear $\Sigma\Pi\Sigma\Pi(k)$ circuit of size s computing the zero polynomial, then the polynomial computed by each F_i in the circuit is $\varphi(k, s)$-sparse, where $\varphi(k, s) \leq s^{5k^2}$.*

We now move the proof of Theorem 5.

Definition 13 *For $n \in \mathbb{N}$ and $\alpha \in \mathbb{F}$ we define the polynomials $\Phi_n^{\alpha}(x_1, \ldots, x_n) \triangleq \prod_{i=1}^{n}(x_i+\alpha)$, $P_n^{\alpha}(\bar{x}) \triangleq \Phi_n^{\alpha}(\bar{x}) + \Phi_n^{-\alpha}(\bar{x})$, $Q_n^{\alpha}(\bar{x}) \triangleq \alpha \cdot \Phi_n^{\alpha}(\bar{x}) - \alpha \cdot \Phi_n^{-\alpha}(\bar{x})$ and $B_{\alpha}(z) \triangleq \begin{bmatrix} z & 1 \\ \alpha^2 & z \end{bmatrix}$.*

The following lemma ties the above together:

Lemma 14 *Let $n \in \mathbb{N}$ and $\alpha \in \mathbb{F}$. Then*

$$B_{\alpha}(x_n) \cdot B_{\alpha}(x_{n-1}) \cdot \ldots \cdot B_{\alpha}(x_1) \cdot \begin{bmatrix} 2 \\ 0 \end{bmatrix} = \begin{bmatrix} P_n^{\alpha} \\ Q_n^{\alpha} \end{bmatrix}.$$

Before giving the proof, we observe that P_n^α has even symmetry w.r.t α (i.e. $P_n^\alpha = P_n^{-\alpha}$) while Q_n^α has odd symmetry w.r.t α (i.e. $-Q_n^\alpha = Q_n^{-\alpha}$). In addition, we remark that the lemma actually shows that the polynomials P_n^α and Q_n^α are computable by read-once oblivious ABPs (see Section 1.2 for definition). This is not a coincidence. The results of Beimel et al. (2000); Klivans and Shpilka (2006) show that set-multilinear $\Sigma\Pi\Sigma$ circuits can be simulated by read-once oblivious ABPs of small width.

Proof Fix $\alpha \in \mathbb{F}$. The proof is by induction on n. For $n = 1$ we get $B_\alpha(x_1) = \begin{bmatrix} 2x_1 \\ 2\alpha^2 \end{bmatrix} = \begin{bmatrix} P_1^\alpha \\ Q_1^\alpha \end{bmatrix}$. Suppose $n \geq 2$. By the induction hypothesis:

$$B_\alpha(x_{n-1}) \cdot \ldots \cdot B_\alpha(x_1) \cdot \begin{bmatrix} 2 \\ 0 \end{bmatrix} = \begin{bmatrix} P_{n-1}^\alpha \\ Q_{n-1}^\alpha \end{bmatrix}.$$

Therefore:

$$B_\alpha(x_n) \cdot B_\alpha(x_{n-1}) \cdot \ldots \cdot B_\alpha(x_1) \cdot \begin{bmatrix} 2 \\ 0 \end{bmatrix} = B_\alpha(x_n) \cdot \begin{bmatrix} P_{n-1}^\alpha \\ Q_{n-1}^\alpha \end{bmatrix} = \begin{bmatrix} x_n \cdot P_{n-1}^\alpha + Q_{n-1}^\alpha \\ \alpha^2 \cdot P_{n-1}^\alpha + x_n \cdot Q_{n-1}^\alpha \end{bmatrix} = \begin{bmatrix} P_n^\alpha \\ Q_n^\alpha \end{bmatrix}.$$

■

Corollary 15 *Let $n \in \mathbb{N}$ and $\alpha^2 = \beta \in \mathbb{F}$. Then the polynomial $P_n^\alpha(\bar{x})$ is computable by a set-multilinear $\Sigma\Pi\Sigma\Pi(2)$ circuit of size $4n$. In addition, given an assignment $\bar{a} \in \mathbb{F}^n$, $P_n^\alpha(\bar{a})$ can be evaluated using $\mathcal{O}(n)$ field operations given β (without the knowledge of α).*

Before giving the proof of the main claim of the section, we require the following result that gives an efficient deterministic factorization algorithm for multilinear sparse polynomials.

Lemma 16 (Shpilka and Volkovich (2010)) *There is a deterministic algorithm that given a multilinear s-sparse polynomial $P \in \mathbb{F}[x_1, x_2, \ldots, x_n]$ outputs the* irreducible *factors, $h_1, \ldots, h_k$ of P using* $\mathrm{poly}(n, s)$ *field operations. Furthermore, each h_i is s-sparse.*

Lemma 17 *Suppose there exists an exact leaner $\mathcal{A}$ that learns multilinear $\Sigma\Pi\Sigma\Pi(2)$ circuits of size s over $\mathbb{F}$ in time $T(s, |\mathbb{F}|)$ when $n \leq s$, with hypotheses being multilinear $\Sigma\Pi\Sigma\Pi(k)$ circuits of size at most s^ℓ for some $k, \ell \in \mathbb{N}$. Then given $\beta \in \mathbb{F}$ in time $T(\mathrm{poly}(\ell, k), |\mathbb{F}|) \cdot 2^{\mathrm{poly}(\ell,k)}$ Algorithm 2 outputs $\alpha \in \mathbb{F}$ such that $\alpha^2 = \beta$, using $\mathcal{A}$ as a subroutine.*

Proof First, we show that the algorithm can successfully answer the queries of $\mathcal{A}$. Corollary 15 allows the algorithm to answer a membership query. In addition, recall that two multilinear polynomials are equivalent iff they agree on the Boolean cube $\{0,1\}^n$. As $\hat{C}$ and $P_n^\alpha(\bar{a})$ are both multilinear polynomials, this take care of an equivalence query. Consequently, $\mathcal{A}$ will output a multilinear $\Sigma\Pi\Sigma\Pi(k)$ circuit of size at most $(4n)^\ell$ computing P_n^α. Therefore, the correctness of Algorithm 2 boils down to the following claim: there exist F_i and x_j such that $x_j \pm \alpha$ is an irreducible factor of F_i. Assume for the contradiction that this

Algorithm 2 Square Root Extraction from Exact Learner for multilinear $\Sigma\Pi\Sigma\Pi(2)$ circuits

Input: $\beta \in \mathbb{F}$, exact learner $\mathcal{A}$, k and ℓ as in the statement of Lemma 17
Output: α such that $\alpha^2 = \beta$

10 Run $\mathcal{A}$ with $n = 10\ell(k+2)^2 \cdot \log(\ell(k+2)^2), s = 4n$ on P_n^α. `/*` $2^n > \varphi\left(k+2,(8n)^\ell\right)$ `*/`
`/* Answering a membership query.` `*/`
11 Given a membership query point $\bar{a} \in \mathbb{F}^n$ by $\mathcal{A}$:
Use Corollary 15 to evaluate $P_n^\alpha(\bar{a})$ `/* Answering an equivalence query.` `*/`
12 Given an equivalence query hypothesis $\hat{C}$:
Test if $\hat{C}|_{\{0,1\}^n} \equiv P_n^\alpha|_{\{0,1\}^n}$ `/* as outlined in Section 1.6.` `*/`
`/* After the interaction with` $\mathcal{A}$. `*/`
13 Let $C = \sum_{i=1}^k F_i$ be the output of $\mathcal{A}$ Use Lemma 16 to factor all the F_i-s into irreducible factors For each factor of the form $x_j + \gamma$: **if** $\gamma^2 = \beta$ **then** Output γ;

is not the case. Consider the following $\Sigma\Pi\Sigma\Pi(k+2)$ circuit: $C' \triangleq \Phi_n^\alpha + \Phi_n^{-\alpha} - F_1 - \ldots - F_k$. By construction, $C' \equiv 0$. Let C'' be the minimal zero subcircuit of C' that contains Φ_n^α. As $\Phi_n^\alpha + \Phi_n^{-\alpha} \not\equiv 0$, C'' must contain at least one (non-zero) F_i. We now claim that C'' is simple. Indeed, the only irreducible factors of Φ_n^α are of the form $x_j + \alpha$. However, by assumption none of them is a factor of F_i. Finally, observe that size of C'' is at most $(4n)^\ell + (4n) \leq (8n)^\ell$. By Lemma 12, Φ_n^α must be $\varphi\left(k+2,(8n)^\ell\right)$-sparse. That is, $2^n \leq \varphi\left(k+2,(8n)^\ell\right)$, in contradiction to the choice of n. ■

Theorem 5 follows as a corollary.

6. Discussion & Open Questions

In this paper we show several consequences from efficient learnability of various classes of arithmetic circuits. In particular, we show that efficient exact learning algorithms for general circuits and formulas imply efficient black-box PIT algorithms for these circuits. The proof goes via constructing explicit lower bounds and applying the hardness-randomness paradigm of Kabanets and Impagliazzo (2004), which is so far limited to general circuit/formulas only. Can one give a more direct proof of this implication? In particular, a proof that will cover a larger family of arithmetic circuit classes? Further more, as outlined in Section 1.6, efficient black-box PIT algorithms can be used to eliminate equivalence queries. As a consequence, one might be able to show that a "rich enough" arithmetic circuit class is effiently learnable iff it is efficiently reconstructible. We note that there is a long standing open problem to transform explicit lower bounds for a class $\mathcal{C}$ into an efficient PIT algorithm for that class (see e.g. Shpilka and Yehudayoff (2010)). As we show that efficient exact learning algorithms imply explicit lower bounds, our question might be easier.

Another natural question is: can one devise or explain the lack of progress for reconstruction algorithms for multilinear formulas with bounded read? We note that there no efficient reconstruction algorithms even for a sum of two read-once formulas. We would like to point out that using the ideas of Section 5 with the techniques of Anderson et al.

(2015); Shpilka and Volkovich (2015), one could show that any reconstruction algorithm for read-twice formulas or even sums of read-once formulas must compute square roots.

Finally, can one prove other consequences from efficient learnability of arithmetic circuits? In particular, can one show that certain learning algorithms imply integer factorization?

Acknowledgment

The author would like to thank Zeev Dvir for pointing out that the techniques used in the paper could show that efficient learning algorithm imply explicit rigid matrices, which resulted in Theorem 4. The author would also like to thank Michael Forbes and Rocco Servedio for useful conversations. Finally, the author would like to thank the anonymous referees for useful comments that improved the presentation of the paper.

References

M. Agrawal. Proving lower bounds via pseudo-random generators. In *Proceedings of the 25th FSTTCS*, volume 3821 of *LNCS*, pages 92–105, 2005.

M. Agrawal and V. Vinay. Arithmetic circuits: A chasm at depth four. In *Proceedings of the 49th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 67–75, 2008.

M. Agrawal, N. Kayal, and N. Saxena. Primes is in P. *Annals of Mathematics*, 160(2): 781–793, 2004.

M. Anderson, D. van Melkebeek, and I. Volkovich. Derandomizing polynomial identity testing for multilinear constant-read formulae. *Computational Complexity*, 24(4):695–776, 2015.

D. Angluin. Learning regular sets from queries and counterexamples. *Inf. Comput.*, 75(2): 87–106, 1987.

D. Angluin. Queries and concept learning. *Machine Learning*, 2:319–342, 1988.

W. Baur and V. Strassen. The complexity of partial derivatives. *Theoretical Comp. Sci.*, 22:317–330, 1983.

D. Beaver and J. Feigenbaum. Hiding instances in multioracle queries. In *STACS*, pages 37–48, 1990. doi: 10.1007/3-540-52282-4_30. URL http://dx.doi.org/10.1007/3-540-52282-4_30.

A. Beimel, F. Bergadano, N. H. Bshouty, E. Kushilevitz, and S. Varricchio. Learning functions represented as multiplicity automata. *J. ACM*, 47(3):506–530, 2000.

A. Blum. Separating distribution-free and mistake-bound learning models over the boolean domain. *SIAM J. Comput.*, 23(5):990–1000, 1994. doi: 10.1137/S009753979223455X. URL http://dx.doi.org/10.1137/S009753979223455X.

N. H. Bshouty, T. R. Hancock, and L. Hellerstein. Learning arithmetic read-once formulas. *SIAM J. on Computing*, 24(4):706–735, 1995.

M. L. Carmosino, R. Impagliazzo, V. Kabanets, and A. Kolokolova. Algorithms from natural lower bounds. *Electronic Colloquium on Computational Complexity (ECCC)*, 23:8, 2016. URL http://eccc.hpi-web.de/report/2016/008.

A. Daniely. Complexity theoretic limitations on learning halfspaces. *CoRR*, abs/1505.05800, 2015. URL http://arxiv.org/abs/1505.05800.

A. Daniely and S. Shalev-Shwartz. Complexity theoretic limitations on learning dnf's. *CoRR*, abs/1404.3378, 2014. URL http://arxiv.org/abs/1404.3378.

R. A. DeMillo and R. J. Lipton. A probabilistic remark on algebraic program testing. *Inf. Process. Lett.*, 7(4):193–195, 1978.

Z. Dvir and A. Shpilka. Locally decodable codes with 2 queries and polynomial identity testing for depth 3 circuits. *SIAM J. on Computing*, 36(5):1404–1434, 2006.

M. Forbes and A. Shpilka. Quasipolynomial-time identity testing of non-commutative and read-once oblivious algebraic branching programs. In *Proceedings of the 54th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 243–252, 2013. Full version at http://eccc.hpi-web.de/report/2012/115.

L. Fortnow and A. R. Klivans. Efficient learning algorithms yield circuit lower bounds. *J. Comput. Syst. Sci.*, 75(1):27–36, 2009.

S. Gao, E. Kaltofen, and A. G. B. Lauder. Deterministic distinct-degree factorization of polynomials over finite fields. *J. Symb. Comput.*, 38(6):1461–1470, 2004.

J. von zur Gathen and J. Gerhard. *Modern computer algebra*. Cambridge University Press, 1999.

C. Gentry and Sh. Halevi. Fully homomorphic encryption without squashing using depth-3 arithmetic circuits. In *Proceedings of the 52nd Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 107–109, 2011.

Sh. Goldwasser and S. Micali. Probabilistic encryption. *J. Comput. Syst. Sci.*, 28(2):270–299, 1984. doi: 10.1016/0022-0000(84)90070-9.

A. Gupta, N. Kayal, and S. V. Lokam. Reconstruction of depth-4 multilinear circuits with top fanin 2. In *Proceedings of the 44th Annual ACM Symposium on Theory of Computing (STOC)*, pages 625–642, 2012. Full version at http://eccc.hpi-web.de/report/2011/153.

J. Håstad. The shrinkage exponent is 2. *SIAM J. on Computing*, 27:48–64, 1998.

J. Heintz and C. P. Schnorr. Testing polynomials which are easy to compute (extended abstract). In *Proceedings of the 12th Annual ACM Symposium on Theory of Computing (STOC)*, pages 262–272, 1980.

R.A. Horn and C.R. Johnson. *Topics in Matrix Analysis*. Cambridge University Press, 1991.

R. Impagliazzo. A personal view of average-case complexity. In *Proceedings of the Tenth Annual Structure in Complexity Theory Conference, Minneapolis, Minnesota, USA, June 19-22, 1995*, pages 134–147, 1995. URL http://dx.doi.org/10.1109/SCT.1995.514853.

V. Kabanets and R. Impagliazzo. Derandomizing polynomial identity tests means proving circuit lower bounds. *Computational Complexity*, 13(1-2):1–46, 2004.

Z. S. Karnin and A. Shpilka. Reconstruction of generalized depth-3 arithmetic circuits with bounded top fan-in. In *Proceedings of the 24th Annual IEEE Conference on Computational Complexity (CCC)*, pages 274–285, 2009. Full version at http://www.cs.technion.ac.il/ shpilka/publications/KarninShpilka09.pdf.

Z. S. Karnin, P. Mukhopadhyay, A. Shpilka, and I. Volkovich. Deterministic identity testing of depth 4 multilinear circuits with bounded top fan-in. *SIAM J. on Computing*, 42(6): 2114–2131, 2013.

N. Kayal. *Derandomizing some number-theoretic and algebraic algorithms*. PhD thesis, Indian Institute of Technology, Kanpur, India, 2007.

M. J. Kearns and L. G. Valiant. Cryptographic limitations on learning boolean formulae and finite automata. *J. ACM*, 41(1):67–95, 1994.

A. Klivans and A. Shpilka. Learning restricted models of arithmetic circuits. *Theory of computing*, 2(10):185–206, 2006.

A. Klivans and D. Spielman. Randomness efficient identity testing of multivariate polynomials. In *Proceedings of the 33rd Annual ACM Symposium on Theory of Computing (STOC)*, pages 216–223, 2001.

A. Klivans, P. Kothari, and I. Oliveira. Constructing hard functions from learning algorithms. In *Proceedings of the 28th Annual IEEE Conference on Computational Complexity (CCC)*, pages 86–97, 2013.

A. R. Klivans and A. A. Sherstov. Cryptographic hardness for learning intersections of halfspaces. *J. Comput. Syst. Sci.*, 75(1):2–12, 2009.

O. Lachish and R. Raz. Explicit lower bound of $4.5n - o(n)$ for boolean circuits. In *Proceedings of the 33rd Annual ACM Symposium on Theory of Computing (STOC)*, pages 399–408, 2001.

A.K. Lenstra, H.W. Lenstr, and L. Lovász. Factoring polynomials with rational coefficients. *Mathematische Annalen,*, 261(4):515–534, 1982.

R. J. Lipton and N. K. Vishnoi. Deterministic identity testing for multivariate polynomials. In *Proceedings of the 14th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 756–760, 2003.

S. V. Lokam. Complexity lower bounds using linear algebra. *Foundations and Trends in Theoretical Computer Science*, 4(1-2):1–155, 2009. doi: 10.1561/0400000011. URL http://dx.doi.org/10.1561/0400000011.

K. Mulmuley, U. Vazirani, and V. Vazirani. Matching is as easy as matrix inversion. *Combinatorica*, 7(1):105–113, 1987.

M. O. Rabin. How to exchange secrets with oblivious transfer. *IACR Cryptology ePrint Archive*, 2005:187, 2005. URL http://eprint.iacr.org/2005/187.

R. Raz. Multi-linear formulas for permanent and determinant are of super-polynomial size. *J. ACM*, 56(2), 2009.

R. Raz, A. Shpilka, and A. Yehudayoff. A lower bound for the size of syntactically multilinear arithmetic circuits. *SIAM J. on Computing*, 38(4):1624–1647, 2008.

S. Saraf and I. Volkovich. Blackbox identity testing for depth-4 multilinear circuits. In *Proceedings of the 43rd Annual ACM Symposium on Theory of Computing (STOC)*, pages 421–430, 2011. Full version at http://eccc.hpi-web.de/report/2011/046.

S. Saraf and S. Yekhanin. Noisy interpolation of sparse polynomials, and applications. In *Proceedings of the 26th Annual IEEE Conference on Computational Complexity CCC*, pages 86–92, 2011. doi: 10.1109/CCC.2011.38. URL http://dx.doi.org/10.1109/CCC.2011.38.

J. T. Schwartz. Fast probabilistic algorithms for verification of polynomial identities. *J. ACM*, 27(4):701–717, 1980.

V. Shoup. A fast deterministic algorithm for factoring polynomials over finite fields of small characteristic. In *ISSAC*, pages 14–21, 1991.

A. Shpilka and I. Volkovich. On the relation between polynomial identity testing and finding variable disjoint factors. In *Automata, Languages and Programming, 37th International Colloquium (ICALP)*, pages 408–419, 2010. Full version at http://eccc.hpi-web.de/report/2010/036.

A. Shpilka and I. Volkovich. On reconstruction and testing of read-once formulas. *Theory of Computing*, 10:465–514, 2014.

A. Shpilka and I. Volkovich. Read-once polynomial identity testing. *Computational Complexity*, 24(3):477–532, 2015.

A. Shpilka and A. Yehudayoff. Arithmetic circuits: A survey of recent results and open questions. *Foundations and Trends in Theoretical Computer Science*, 5(3-4):207–388, 2010.

L. G. Valiant. Graph-theoretic arguments in low-level complexity. In *Lecture notes in Computer Science*, volume 53, pages 162–176. Springer, 1977.

L. G. Valiant. A theory of the learnable. *Communications of the ACM*, 27(11):1134–1142, 1984.

I. Volkovich. On some computations on sparse polynomials. *Manuscript*, 2015. (submitted).

R. Zippel. Probabilistic algorithms for sparse polynomials. In *Proceedings of the International Symposium on Symbolic and Algebraic Computation*, pages 216–226, 1979.

Appendix A. PAC and Exact Learning are essentially equivalent for Arithmetic Circuits

In this section we give a formal proof to a folklore result. We begin by formally defining the model.

In Valiant's PAC Learning model, we have a (computationally bounded) learner that is given a set of samples of the form $(\bar{x}, f(\bar{x}))$ from some fixed function $f \in \mathcal{C}$, where $\bar{x}$ is chosen according to some unknown distribution D. Given $\varepsilon > 0$ and $\delta > 0$, the learner's goal is to output a hypothesis "functionally close" to f w.h.p. Formally, we say that a function class $\mathcal{C}$ is PAC *learnable* if there exists a learner which given any $f \in \mathcal{C}$, $\varepsilon > 0$ and $\delta > 0$ in time polynomial in $n, 1/\varepsilon, 1/\delta, |f|$ outputs with probability $1 - \varepsilon$ a hypothesis $\hat{f}$ such that $\hat{f}$ is a $1 - \delta$ close to f under D. In a more general model, the learner is allowed membership queries (as in the exact learning model). In this case, we say that $\mathcal{C}$ is PAC *learnable* with *membership queries*.

We show that both PAC and Exact learning models are, essentially, equivalent in the arithmetic setting. In fact we show that if an arithmetic circuit class $\mathcal{C}$ is PAC learnable with *membership queries* then $\mathcal{C}$ is learnable with membership queries only. In addition, if the learning algorithm is proper and $\mathbb{F}$ is sufficiently large then the algorithm actually outputs a circuit from $\mathcal{C}$. The celebrated result of Blum (1994) shows that if one-way functions exist, then this is not the case for Boolean functions.

It is also known Angluin (1987) that both randomized and exact learners can be used to obtain a PAC learner with membership queries.

Definition 18 (Distance) *Let $f, g : \mathbb{F}^n \to \mathbb{F}$ be functions. We define their (relative)* distance *as $\Delta(f, g) \stackrel{\Delta}{=} \Pr_{\bar{a} \in \mathbb{F}^n}[f(\bar{a}) \neq g(\bar{a})]$. For $\delta > 0$ we say that f is δ-*far *from g if $\Delta(f, g) > \delta$; otherwise we say that f is δ-*close *to g.*

We give the Schwartz-Zippel Lemma which provides a bound on the number of zeros of a polynomial.

Lemma 19 (Zippel (1979); Schwartz (1980)) *Let $P \in \mathbb{F}[x_1, x_2, \ldots, x_n]$ be a non-zero polynomial of degree at most d and let $V \subseteq \mathbb{F}$. Then $\Pr_{\bar{a} \in V^n}[P(\bar{a}) = 0] \leq \frac{d}{|V|}$.*

The following is an important property of low-degree polynomials: Self-Correctability.

Lemma 20 (Beaver and Feigenbaum (1990)) *Let $n, d \in \mathbb{N}$ and $\mathbb{F}$ be a field of size $|\mathbb{F}| > d + 2$. Let $f : \mathbb{F}^n \to \mathbb{F}$ be a function that is $1/d$-close to a degree d polynomial $P \in \mathbb{F}[x_1, x_2, \ldots, x_n]$. Then there exists a randomized algorithm* CORRECT *that uses* $\mathrm{poly}(n, d)$ *field operations and oracle calls to f such that for any $\bar{a} \in \mathbb{F}^n$:* $\Pr[\mathsf{CORRECT}^f(\bar{a}) \neq P(\bar{a})] \leq 2^{-10n}$.

The following is obtained by standard techniques:

Observation 21 *Let $f : \mathbb{F}^n \to \mathbb{F}$ be a function that is $1/d$-close to a degree d polynomial $P \in \mathbb{F}[x_1, x_2, \ldots, x_n]$ and let C be a Boolean circuit of size s computing f. Then there exists a circuit $\hat{C}$ of size $\mathrm{poly}(s, n, d)$ that computes P. Moreover, there is randomized algorithm that given C output $\hat{C}$ with high probability in time $\mathrm{poly}(s, n, d)$.*

We now give our main claim:

Lemma 22 *Let $\mathcal{C}$ be an arithmetic circuit class over the field $\mathbb{F}$. If $\mathcal{C}$ is* PAC *learnable with membership queries then, $\mathcal{C}$ is learnable with membership queries only. In addition, if the learning algorithm is proper and $\mathbb{F} = \mathrm{poly}(d)$ is sufficiently large, then $\mathcal{C}$ can be reconstructed efficiently.*

Proof Let $\mathcal{A}$ be an exact learner for $\mathcal{C}$ and let $C \in \mathcal{C}$ be a (n, s, d)-circuit given via oracle access. We run $\mathcal{A}$ on C with $\delta = 1/d$ and $\varepsilon = 1/n$. Whenever $\mathcal{A}$ asks for a random sample point, we pick $\bar{a} \in \mathbb{F}^n$ at random and return $(\bar{a}, C(\bar{a}))$. Let $h(\bar{x}) : \mathbb{F}^n \to \mathbb{F}$ be the output of $\mathcal{A}$. We have that $\Delta(h, C) \leq 1/d$. We can compute a Boolean Circuit C' for h. Applying Observation 21, in randomized time $\mathrm{poly}(n, s, d)$, we can compute a circuit $\hat{C}$ such that $\hat{C} \equiv C$.

If $\mathcal{A}$ is proper, h is a $(n, \mathrm{poly}(s), \mathrm{poly}(d))$-circuit from $\mathcal{C}$. By Lemma 19, if $h \not\equiv C$ then $\Delta(h, C) \geq 1 - \mathrm{poly}(d)/|\mathbb{F}|$ which is larger than $1 > d$ when $\mathbb{F} = \mathrm{poly}(d)$. Therefore, $h \equiv C$. ■