Learning Restricted Models of Arithmetic Circuits

Adam R. Klivans[*]
Department of Computer Science
University of Texas at Austin
Austin, TX 78712-1188
klivans@cs.utexas.edu

Amir Shpilka
Faculty of Computer Science
Technion
Haifa 32000, Israel
shpilka@cs.technion.ac.il

Abstract

We present a polynomial time algorithm for learning a large class of algebraic models of computation. We show that any arithmetic circuit whose partial derivatives induce a low-dimensional vector space is exactly learnable from membership and equivalence queries. As a consequence, we obtain polynomial time algorithms for learning restricted algebraic branching programs as well as noncommutative set-multilinear arithmetic formulae. In addition, we observe that the algorithms of [6, 4] can be used to learn depth-3 set-multilinear arithmetic circuits. Previously only versions of depth-2 arithmetic circuits were known to be learnable in polynomial time. Our learning algorithms can be viewed as solving a generalization of the well known *polynomial interpolation problem* where the unknown polynomial has a succinct representation. We can learn representations of polynomials encoding *exponentially* many monomials. Our techniques combine a careful algebraic analysis of the partial derivatives of arithmetic circuits with the "multiplicity automata" techniques due to [5, 4].

1 Introduction

Let p be an unknown multivariate polynomial over a fixed field. Given random input/output pairs chosen from some distribution D, can a computationally bounded learner output a hypothesis which will correctly approximate p with respect to future random examples chosen from D? This problem, known as the multivariate polynomial learning problem, continues to be a fundamental area of research in computational learning theory. If the learner is allowed to query the unknown polynomial at points of his choosing (instead of receiving random examples) and is required to output the exact polynomial p, then this problem is precisely the well-known polynomial interpolation problem. Both the learning and the interpolation problem have received a great deal of attention from the theoretical computer science community. In a learning context, multivariate polynomials are expressive structures for encoding information (sometimes referred to as the "algebraic" analogue of DNF formulae (see e.g. [4])) while polynomial interpolation has been studied in numerous contexts and has important applications in complexity theory, among other fields [2, 34].

Previous research on this problem has focused on giving algorithms whose running time is polynomial in the number of terms or monomials of the unknown polynomial. This is a natural way to measure the complexity of learning and interpolating polynomials when the unknown polynomial is viewed in the usual "sum of monomials" representation. That is to say, given that the polynomial $p = \sum_{i=1}^{t} m_i$ is the sum of t monomials, we may wish to output a list of these monomials (and their respective coefficients), hence using at least t time steps simply to write down the list of coefficients.

[*] Supported by an NSF Mathematical Sciences Postdoctoral Research Fellowship.

Several researchers have developed powerful interpolation and learning algorithms for a variety of contexts which achieve time bounds polynomial in all the relevant parameters, including t (see for example [4, 11, 16, 20, 23, 31]).

1.1 Arithmetic Circuits

In this paper we are concerned with learning succinct representations of polynomials via alternate algebraic models of computation, most notably *arithmetic circuits*. An arithmetic circuit syntactically represents a multivariate polynomial in the obvious way: a multiplication (addition) gate outputs the product (sum) of the polynomials on its inputs. The input wires to the circuit correspond to the input variables of the polynomial and thus the output of the circuit computes some polynomial of the input variables. We measure the size of an arithmetic circuit as the number of gates. For example, the standard "sum of monomials" representation of a polynomial $p = \sum_{i=1}^{t} \alpha_i x_{i_1} \cdots x_{i_n}$ (α_i is an arbitrary field element) corresponds precisely to a depth-2 arithmetic circuit with a single sum gate at the root and t product gates feeding into the sum gate (each product gate multiplies some subset of the input variables). To rephrase previous results on learning and interpolating polynomials in terms of arithmetic circuits, we could say that depth-2 arithmetic circuits with a sum gate at the root are learnable in time polynomial in the size of the circuit.

Moving beyond the standard "sum of products" representation, we consider the complexity of learning higher depth arithmetic circuits. It is easy to see that there exist polynomial size depth-3 (or even depth-2 with a product gate at the root) arithmetic circuits capable of computing polynomials with *exponentially* many monomials. For example, let $\{L_{i,j}\}, 1 \leq i, j \leq n$ be a family of n^2 distinct linear forms over n variables. Then $\sum_{i=1}^{n} \prod_{i=1}^{n} L_{i,j}$ is a polynomial size depth-3 arithmetic circuit but cannot be written as a sum of polynomially many monomials. Although arithmetic circuits have received a great deal of attention in complexity theory and, more recently, derandomization, the best known result for learning arithmetic circuits in a representation other than the depth-2 sum of products representation is due to Beimel *et al.* [4] who show that depth-2 arithmetic circuits with a product gate of fan in $O(\log n)$ at the root and addition gates[1] of unbounded fan-in in the bottom level are learnable in polynomial time, and that circuits that compute polynomials of the form $\sum_i \prod_j p_{i,j}(x_j)$ ($p_{i,j}$ is a univariate polynomial of polynomial degree) can be learned in polynomial time[2].

1.2 Our Results

We learn various models of algebraic computation capable of encoding exponentially many monomials in their input size. Our algorithms work with respect to any distribution and require membership query access to the concept being learned. More specifically we show that any class of polynomial size arithmetic circuits whose partial derivatives induce a vector space of dimension polynomial in the circuit size is learnable in polynomial time. This characterization generalizes the work of Beimel *et al.* [4] and yields the following results:

- An algorithm for learning general depth-3 arithmetic circuits with m product gates each of fan in at most d in time polynomial in m, 2^d, and n, the number of variables.

[1] Beimel *et al.* actually allow addition gates to sum powers of the input variables, rather than just summing variables.

[2] The latter class of circuit can be viewed as a restricted version of depth-3 circuits where the addition gates at the bottom can only sum powers of a certain variable

- The first polynomial time algorithm for learning polynomial size noncommutative formulae computing polynomials over a fixed partition of the variables (note there are no depth restrictions on the size of the formula).
- The first polynomial time algorithm for learning polynomial size read once, oblivious algebraic branching programs.

As an easy consequence of our results we observe a polynomial time algorithm for learning the class of depth-3 set-multilinear circuits: polynomials $p = \sum_{i=1}^{m} \prod_{j=1}^{n} L_{i,j}(X_j)$ where each $L_{i,j}$ is a linear form and the X_j's are a partition of the input variables. We note that this result also follows as an easy corollary from the work of [4]. Finally we show that, with respect to known techniques, it is hard to learn polynomial size depth-3 homogeneous arithmetic circuits in polynomial time[3]. This indicates that our algebraic techniques give a fairly tight characterization of the learnability of arithmetic circuits with respect to current algorithms.

1.3 Our Techniques

We use as a starting point the work on multiplicity automata due to Beimel *et al.* [4]. A multiplicity automaton is a nondeterministic finite automaton where each transition edge has weight from the underlying field (for a precise definition see Section 2). On input x, $f(x)$ is equal to the sum, over all accepting paths of the automaton on input x, of the product of the edge weights on that accepting path. In [7, 4], the authors, building on work due to [27], show that multiplicity automata can be learned in polynomial time and that these multiplicity automata can compute polynomials in their standard sum of products representation (actually, as mentioned earlier, they can learn any polynomial p of the form $p = \sum_{i=1}^{n} \prod_{j=1}^{m} p_{ij}(x_j)$ where $p_{ij}(x_j)$ is a univariate polynomial of polynomial degree). Their analysis centers on the Hankel matrix of a multiplicity automaton (see section 2 for a definition).

We give a new characterization of learnability in terms of partial derivatives. In particular we show that any polynomial whose partial derivatives induce a low dimensional vector space has a low rank Hankel matrix. We conclude that any arithmetic circuit or branching program whose partial derivatives form a low dimensional vector space can be computed by polynomial size multiplicity automaton and are amenable to the learning algorithms developed in [7, 4]. As such, we output a multiplicity automaton as our learner's hypothesis.

Our next task is to show which circuit classes have partial derivatives that induce low dimensional vector spaces. At this point we build on work due to Nisan [25] and Nisan and Wigderson [26] (see also [33]) who analyzed the partial derivatives of certain arithmetic circuit classes in the context of proving lower bounds and show that a large class of algebraic models have "well-behaved" partial derivatives. For example we show that the dimension of the vector space of partial derivatives induced by a set-multilinear depth-3 arithmetic circuit is polynomial in the size of the circuit.

Our results suggest that partial derivatives are a powerful tool for learning multivariate polynomials, as we are able to generalize all previous work in this area and give new results for learning interesting algebraic models. Additionally, we can show there are depth-3 polynomial-size, homogeneous, arithmetic circuits whose partial derivatives induce a vector space of superpolynomial dimension. We feel this motivates the problem of learning depth-3 homogeneous, polynomial-size arithmetic circuits, as such a result would require significantly new techniques. We are hopeful

[3] A depth-3 circuit $p = \sum_{i=1}^{m} \prod_{j=1}^{d_i} L_{i,j}(x_1, \ldots, x_n)$ is homogeneous if in each $L_{i,j}$ the free term is zero.

that our characterizations involving partial derivatives will further inspire complexity theorists to use their techniques for developing learning algorithms.

1.4 The Relationship to Lower Bounds

In the case of learning Boolean functions, the ability to prove lower bounds against a class of Boolean circuits usually coincides with the ability to give strong learning algorithms for those circuits. For example the well known lower bounds of Håstad [19] against constant depth Boolean circuits are used heavily in the learning algorithm due to Linial, Mansour, and Nisan [24]. Jackson *et al.* [21] have shown that constant depth circuits with a majority gate, one of the strongest circuit classes for which we can prove lower bounds (see [3]), also admit nontrivial learning algorithms. Furthermore Jackson *et al.* [21] show that we will not be able to learn more complicated Boolean circuits unless certain cryptographic assumptions are false.

Our work furthers this relationship in the algebraic setting. The models of algebraic computation we can learn correspond to a large subset of the models of algebraic computation for which strong lower bounds are known. For example Nisan [25] gives exponential lower bounds for noncommutative formulae. Nisan and Wigderson [26] prove exponential lower bounds for depth-3 set-multilinear circuits. Moreover, in both papers the authors prove lower bounds by considering the partial derivatives spanned by the circuit and the function computed by it, a method similar to ours. Over finite fields there are exponential lower bounds for depth 3 circuits [15, 17], however no exponential lower bounds are known for general depth-3 arithmetic circuits over infinite fields (see [33]). As in the Boolean case, we exploit many of the insights from the lower bound literature to prove the correctness of our learning algorithms.

A preliminary version of this paper appeared in COLT 2003 [22].

1.5 Organization

In Section 2 we review relevant learning results for multiplicity automata as well as state some basic facts from algebraic complexity. In Section 3 we prove our main theorem, characterizing the learnability of arithmetic circuits via their partial derivatives. In Sections 4, 5, and 6 we state our main learning results for various arithmetic circuits and algebraic branching programs.

2 Preliminaries

We denote with $\mathbb{F}$ the underlying field, and with $\mathrm{char}(\mathbb{F})$ the characteristic of $\mathbb{F}$. When studying a polynomial f we either assume that $\mathrm{char}(\mathbb{F}) = 0$ or that the degree of each variable in f is smaller than $\mathrm{char}(\mathbb{F})$.

2.1 The Learning Model

We will work in the model of exact learning from membership and equivalence queries, first introduced by Angluin [1]. In this model a learner begins with some candidate hypothesis h for an unknown concept f and is allowed access to both a *membership query* oracle and an *equivalence query* oracle. The membership query oracle takes as input x and outputs $f(x)$. The equivalence query oracle takes as input the learner's current hypothesis h and outputs a counterexample, namely an input y such that $h(y) \neq f(y)$. We assume that making a membership or an equivalence query of length k takes time k. If no such counterexample exists then we say that the learner has exactly learned f. We say that a concept f is exactly learnable in time t if there exists an exact learner for f whose running time is bounded by t. A concept class is considered to be exactly learnable in polynomial time if for every f in the concept class there exists an exact learner for f running in time polynomial in the size of the smallest description of f. Known transformations imply that if a concept class is exactly learnable in polynomial time then it is also learnable in Valiant's PAC model in polynomial time with membership queries.

2.2 Multiplicity Automata

A multiplicity automaton is a nondeterministic automaton where each transition edge is assigned a weight, and the output of the automaton for any input x is the sum over all accepting paths of x of the products of the weights on each path.

Definition 1. *Let Σ be an alphabet. A multiplicity automaton A of size r over Σ consists of a vector $\bar{\gamma} \in \mathbb{F}^r$ (i.e. $\bar{\gamma} = (\gamma_1, \ldots, \gamma_r)$) and a set of matrices $\{\mu_\sigma\}_{\sigma \in \Sigma}$, where each μ_σ is an $r \times r$ matrix over $\mathbb{F}$. The output of A on input $x = (x_1, \ldots, x_n) \in \Sigma^n$ is defined to be the inner product of $(\prod_{i=1}^n \mu_{x_i})_1$ and $\bar{\gamma}$ where $(\prod_{i=1}^n \mu_{x_i})_1$ equals the first row of the matrix*[4] *$\prod_{i=1}^n \mu_{x_i}$. In other words the output is the first coordinate of the vector $(\prod_{i=1}^n \mu_{x_i}) \cdot \bar{\gamma}$.*

Intuitively each matrix μ_σ corresponds to the transition matrix of the automaton for symbol $\sigma \in \Sigma$. Iterative matrix multiplication keeps track of the weighted sum of paths from state i to state j for all $i, j \leq r$. The first row of the iterated product corresponds to transition values starting from the initial state and $\bar{\gamma}$ determines the acceptance criteria.

Next we define the Hankel matix of a function:

Definition 2. *Let Σ be an alphabet and $f : |\Sigma|^n \to \mathbb{F}$. Fix an ordering of all strings in $\Sigma^{\leq n}$. We construct a matrix H whose rows and columns are indexed by strings in $\Sigma^{\leq n}$ in the following way. For $x \in \Sigma^d$ and $y \in \Sigma^{n-d}$, for some $0 \leq d \leq n$, let the (x, y) entry of H be equal to $f(x \circ y)$. For any other pair of strings (x, y) such that $|x| + |y| \neq n$ let $H_{a,b} = 0$. The resulting matrix H is called the Hankel matrix of f for strings of length n. We define H_k to be the k-th "block" of H, i.e. H_k is the submatrix defined by all rows of H indexed by strings of length exactly k and all columns of H indexed by strings of length exactly $n - k$.*

The following key fact relates the rank of the Hankel matrix of a function for strings of length n with the size of multiplicity automaton computing f on inputs of length n:

Theorem 3. *[13, 14, 4] Let $f : \Sigma^n \to \mathbb{F}$. Then the rank of the Hankel matrix of f (over $\mathbb{F}$) is equal to the size of the smallest multiplicity automaton computing f on inputs of length n.*

Previous learning results have computed the rank of the Hankel matrices of particular polynomials yielding a bound on the size of their multiplicity automata. In fact, Beimel *et al.* [4], improving on [27], learn functions computed by multiplicity automata by iteratively learning their corresponding Hankel matrices:

Theorem 4. *[4] For $f : \Sigma^n \to \mathbb{F}$, let r be the rank of the Hankel matrix of f for strings of length n. Then there exists an exact learning algorithm for f running in time polynomial in n, r, and $|\Sigma|$. Furthermore the final hypothesis output by the learning algorithm is a multiplicity automaton of size r over alphabet Σ. Moreover, if for every variable x_i the degree of f as a polynomial in x_i (over $\mathbb{F}$) is at most d, then the running time of the learning algorithm is polynomial in n, r and d.*

[4] We denote $\mu_{x_n} \cdot \mu_{x_{n-1}} \cdot \ldots \cdot \mu_{x_1}$ with $\prod_{i=1}^n \mu_{x_i}$.

Our main technical contribution is to show that the rank of a function's Hankel matrix is bounded by (and in most cases equal to) the dimension of the vector space of a function's partial derivatives. Thus we reduce the problem of learning a polynomial to bounding the dimension of the vector space of its partial derivatives.

2.3 Set-Multilinear Polynomials

In this paper we will work primarily with polynomials that respect a fixed partition of the input variables:

Definition 5. *Let $X = \dot{\cup}_{i=1}^{d} X_i$ be a partition of the variables into d sets. A polynomial over the variables X is called* set-multilinear *if every monomial m is of the form $y_1 \cdot y_2 \cdots y_d$ where each y_i is some variable from X_i. Thus, any set-multilinear f is also homogeneous and multilinear of degree d.*

We will sometime use the notation $f(X_1, \ldots, X_d)$ to denote that f is set-multilinear w.r.t. the partition $X = \dot{\cup}_{i=1}^{d} X_i$.

Example 6. *Let $X = (x_{i,j})_{1 \leq i,j \leq d}$ be a $d \times d$ matrix. Let $X_i = \{x_{i,1}, \ldots, x_{i,d}\}$ be the i-th row of X. Clearly $X = \dot{\cup}_{i=1}^{d} X_i$. Note that both the determinant and the permanent of X are set-multilinear polynomials with respect to this partition.*

Another example is the class of depth-3 set-multilinear circuits, first defined[5] by Nisan and Wigderson [26], that computes only set-multilinear polynomials. To see this note that any polynomial computed by a depth-3 set-multilinear circuits is of the form $p = \sum_{i=1}^{n} \prod_{j=1}^{m} L_{i,j}(X_j)$ where each $L_{i,j}$ is a linear form and the X_j's are a partition of the input variables. In later sections we will show that certain algebraic branching programs also compute set-multilinear polynomials and will therefore be amenable to our learning techniques.

Another notation that we use is the following:

Definition 7. *Let $X = \dot{\cup}_{i=1}^{d} X_i$. For any $1 \leq k \leq d$ define*

$$\mathbb{SM}[X_1, ..., X_k] = \{\ M \mid M = \prod_{i=1}^{k} x_i\ , x_i \in X_i\ \}.$$

Thus $\mathbb{SM}[X_1, ..., X_k]$ is the set of all set-multilinear monomials of degree k.

2.4 Partial Derivatives

In this subsection we introduce some notation for computing partial derivatives.

Definition 8. *Let $\mathbb{M}[x_1, \ldots, x_n]$ be the set of monomials in the variables $x_1, \ldots, x_n$. Let $\mathbb{M}_d[x_1, \ldots, x_n]$ be the set of monomials of degree at most d in $x_1, \ldots, x_n$.*

Example 9. $\mathbb{M}_2[x_1, x_2] = \{1, x_1, x_2, x_1^2, x_1 \cdot x_2, x_2^2\}$.

[5] In the original paper [26] these circuits are called multilinear circuits, but in recent works [28, 29, 30] they are referred to as set-multilinear circuits.

Definition 10. *Let $d = \sum_{i=1}^{k} d_i$. For a function $f(x_1, ..., x_n)$ and a monomial $M = \prod_{i=1}^{k} {x_i}^{d_i}$ let*

$$\frac{\partial f}{\partial M} = \frac{1}{M!} \cdot \frac{\partial^d f}{\prod_{i=1}^{k} (\partial x_i)^{d_i}},$$

where $M! = \prod_{i=1}^{k} (d_i!)$.

Recall that in case that $\mathbb{F}$ is finite we only consider polynomials in which the degree of each variable is smaller than the characteristic of $\mathbb{F}$. In particular we will only consider partial derivatives with respect to monomials in which each variable has degree smaller than char($\mathbb{F}$).

Example 11. *Let $f(x_1, x_2, x_3) = x_1^2 x_2 + x_3$ and $M(x_1, x_2, x_3) = x_1 x_2$. We have that*

$$\frac{\partial f}{\partial M} = 2x_1, \ M! = 1.$$

Definition 12. *For a function $f(x_1, \ldots, x_n)$ and $k \leq n$ let*

$$\partial_k(f) = \{ \ \frac{\partial f}{\partial M} \mid \text{monomials } M \in \mathbb{M}[x_1, ..., x_k] \ \}.$$

Also define

$$\text{rank}_k(f) = \dim \left(\text{span} \left(\partial_k(f) \right) \right).$$

Note that in $\partial_k(f)$ we only consider partial derivatives w.r.t. the first k variables.

Example 13. *Let $X = (x_{i,j})_{1 \leq i,j \leq 3}$ be a 3×3 matrix. Let $f(X) = Det(X)$ (the determinant of X). Consider the following order of the variables $x_{i,j} < x_{i',j'}$ if $i < i'$ or $i = i'$ and $j < j'$. Then*

$$\partial_6(X) = \{x_{2,2}x_{3,3} - x_{2,3}x_{3,2}, x_{2,1}x_{3,3} - x_{2,3}x_{3,1}, x_{2,1}x_{3,2} - x_{2,2}x_{3,1}, x_{3,1}, x_{3,2}, x_{3,3}\}.$$

Thus, $\text{rank}_6(f) = 6$.

For set-multilinear polynomials we need a slightly different definition (although we use the same notations).

Definition 14. *Let $X = \dot{\cup}_{i=1}^{d} X_i$. For a set-multilinear polynomial $f(X_1, \ldots, X_d)$ and $k \leq d$ let*

$$\partial_k(f) = \{ \ \frac{\partial f}{\partial M} \mid \text{monomials } M \in \mathbb{SM}[X_1, ..., X_i] \text{ for } 1 \leq i \leq k \ \}.$$

We define $\text{s-rank}_k(f) = \dim \left(\text{span} \left(\partial_k(f) \right) \right)$.

Note that in particular we only consider partial derivatives with respect to monomials of the form $\prod_{i=1}^{k'} x_i$ where $x_i \in X_i$ and $k' \leq k$. We will never consider partial derivative with respect to the monomial $x_1 \cdot x_3$ (again, $x_i \in X_i$).

Example 15. *Let X be a 3×3 matrix (as in Example 6 with $d = 3$). Let $f = Det(X) = Det(X_1, X_2, X_3)$ be the determinant of X where where $X_1 = \{x_{1,1}, x_{1,2}, x_{1,3}\}, X_2 = \{x_{2,1}, x_{2,2}, x_{2,3}\}, and X_3 = \{x_{3,1}, x_{3,2}, x_{3,3}\}$. Then $\partial_2(f) = \{x_{3,1}, x_{3,2}, x_{3,3}\}$. Thus,* $\text{s-rank}_2(f) = 3$.

Note the difference from Example 13 where we ignored the fact that the determinant is a set-multilinear polynomial.

3 Characterizing Learnability via Partial Derivatives

In this section, we present our main criterion for establishing the learnability of both arithmetic circuits and algebraic branching programs. We prove that any polynomial whose partial derivatives form a low degree vector space induce low rank Hankel matrices. To relate the rank of the Hankel matrix of C to its partial derivatives we will need the following multivariate version of Taylor's theorem:

Fact 16. *Let $X = \{x_1, \dots, x_n\}$ and let $f(X)$ be a degree d polynomial. Let $\rho = (\rho_1 \circ \rho_2)$ be an assignment to the variables, where ρ_1 is an assignment to the first k variables and ρ_2 as assignment to the last $n-k$ variables. For a monomial M define $M(\rho)$ be value of M on assignment ρ. Then*

$$f(\rho) = f(\rho_1 \circ \rho_2) = \sum_{M \in \mathbb{M}_d[x_1,\dots,x_k]} M(\rho_1) \cdot \frac{\partial f}{\partial M}(\vec{0} \circ \rho_2).$$

Proof. Because of the linearity of the partial derivative operator it is enough to prove the claim for the case that f is a monomial. Let $f(x_1, \dots, x_n) = \prod_{i=1}^n {x_i}^{d_i}$, where $\sum d_i \le d$. Consider a monomial $M \in \mathbb{M}_d[x_1, \dots, x_k]$ given by $M = \prod_{i=1}^k {x_i}^{e_i}$, where $\sum e_i \le d$. Notice that if there is some $1 \le i \le k$ with $e_i > d_i$ then $\frac{\partial f}{\partial M} = 0$. Also notice that if for some $1 \le i \le k$ we have that $e_i < d_i$ then $\frac{\partial f}{\partial M}(\vec{0} \circ \rho_2) = 0$ because the assignment to x_i is zero. In particular the only contribution to the sum will come from the partial derivative w.r.t. $M_0 = \prod_{i=1}^k {x_i}^{d_i}$ that gives $\frac{\partial f}{\partial M_0} = \prod_{i=k+1}^n {x_i}^{d_i}$. In particular

$$\sum_{M \in \mathbb{M}_d[x_1,\dots,x_k]} M(\rho_1) \cdot \frac{\partial f}{\partial M}(\vec{0} \circ \rho_2) = M_0(\rho_1) \cdot \frac{\partial f}{\partial M_0}(\vec{0} \circ \rho_2) = \prod_{i=1}^k {x_i}^{d_i}(\rho_1) \cdot \prod_{i=k+1}^n {x_i}^{d_i}(\rho_2) = f(\rho).$$

□

Now we can state the main technical theorem of the paper:

Theorem 17. *Let $f(x_1, \dots, x_n)$ be a degree d polynomial. Then for every $k \le n$,*

$$\dim(H_k(f)) \le \mathrm{rank}_k(f).$$

If f is multilinear then equality holds.

Proof. We will define two matrices $V_{d,k}$ and E_k such that $\mathrm{rank}(E_k) \le \mathrm{rank}_k(f)$ and $H_k = V_{d,k} \cdot E_k$.

Construction of E_k (Evaluation Matrix): We index the rows of E_k by the set of monomials $\mathbb{M}_d[x_1, ..., x_k]$ (in lexicographical order) and the columns by elements of $\mathbb{F}^{n-k}$ (in lexicographical order). The (M, ρ) entry of E_k is equal to

$$(E_k)_{M,\rho} = \frac{\partial f}{\partial M}(\vec{0} \circ \rho),$$

where $\vec{0}$ is a length k vector and ρ is in $\mathbb{F}^{n-k}$. This is equal to the value of the partial derivative of f with respect to M at the point $\vec{0} \circ \rho$. When $k = 0$, the matrix has only one row (the partial derivative of order zero is the polynomial itself), in which the ρth position is equal to $f(\rho)$. The following is a standard fact from linear algebra:

Claim 18. $\mathrm{rank}(E_k) \le \mathrm{rank}_k(f)$, *and equality holds if f is multilinear.*

Proof. Row M of E_k is the evaluation of $\frac{\partial f}{\partial M}$ on all inputs of the form $\vec{0} \circ \rho$, where $\vec{0}$ is a length k vector and ρ is of length $n-k$. Hence each vector corresponds to part of the "truth table" of a particular partial derivative of f in which the assignment to the first k variables is zero. Clearly if a set of partial derivatives is linearly dependent then so are the corresponding rows. Thus $\text{rank}(E_k) \leq \text{rank}_k(f)$. When f ismultilinear, all of the variables in M disappear from the resulting polynomial, and we actually get that the rows of E_k represent the entire truth table of the corresponding partial derivative of f and hence $\text{rank}(E_k) = \text{rank}_k(f)$. □

Construction of $V_{d,k}$ (Generalized Vandermonde Matrix): The rows of $V_{d,k}$ are indexed by elements of $\mathbb{F}^k$ (in lexicographical order) and the columns are indexed by the set of monomials $\mathbb{M}_d[x_1, ..., x_k]$ (again in lexicographical order). The (ρ, M) entry of $V_{d,k}$ is equal to $M(\rho)$. When $k = 0$ the matrix contains only one column, whose entries are equal to 1. We note that the column rank of V_k is full (similarly to the usual Vandermonde matrix).

Consider the matrix product $V_{d,k} \cdot E_k$. Notice that its $(\rho_1 \circ \rho_2)$ entry is equal to

$$(V_{d,k} \cdot E_k)_{\rho_1 \circ \rho_2} = \sum_{M \in \mathbb{M}_d[x_1,...,x_k]} M(\rho_1) \frac{\partial f}{\partial M}(\vec{0} \circ \rho_2)$$

which by Fact 16 equals $f(\rho_1 \circ \rho_2)$. Thus $V_{d,k} \cdot E_k = H_k$. In particular $\text{rank}(H_k) \leq \text{rank}(E_k) \leq \text{rank}_k(f)$. When f is multilinear we have that, as before, $\text{rank}(E_k) = \text{rank}_k(f)$, and as the column rank of V_k is full it follows that $\text{rank}(H_k) = \text{rank}(E_k) = \text{rank}_k(f)$. □

By summing over all values of k we obtain

Corollary 19. *Let $f(x_1, \ldots, x_n)$ be a polynomial. Then*

$$\dim(H(f)) \leq \sum_{k=0}^{n} \text{rank}_k(f).$$

If f is multilinear then

$$\dim(H(f)) = \sum_{k=0}^{n} \text{rank}_k(f).$$

Now we consider set-multilinear polynomials. We must be careful here to take into account partial derivatives with respect to monomials M that are not in $\mathbb{SM}[X_1, \ldots, X_i]$ for any i. Below, we show that rows in E_k corresponding to such M's are zero.

Let $f(X_1, \ldots, X_d)$ be a set-multilinear polynomial w.r.t. $X = \dot{\cup} X_i$. We order the variables of X as follows: first we set $X_1 < X_2 < \ldots < X_d$, then we order the variables in each X_i in some linear order. Consider the (M, ρ) entry in E_k. Notice that if $M \notin \cup_{i=1}^{d} \mathbb{SM}[X_1, \ldots, X_i]$ then $\frac{\partial f}{\partial M}(\vec{0} \circ \rho) = 0$. Indeed, assume that the first k variables cover the sets $X_1, \ldots, X_i$, as well as some of the variables in the set X_{i+1}. Since we substitute 0 to the first k variables we see that M must contain a variable from each $X_1, \ldots, X_i$ (as otherwise, because f is set-multilinear, the entire M-th row of E_k is zero). We also note that M can't contain two variables from the set X_j (as again this would imply that the M-th row is zero). In particular, in order for the M-th row to be non zero we must have that $M \in \mathbb{SM}[X_1, \ldots, X_i]$ for that i. As a corollary we get

Corollary 20. *Let f be a set-multilinear polynomial with an ordering of the variables as above. For each $1 \leq k \leq n$ let i_k be defined by*

$$\left| \cup_{i=1}^{i_k} X_i \right| \leq k < \left| \cup_{i=1}^{i_k+1} X_i \right|.$$

Then $\mathrm{rank}(E_k) \leq \text{s-rank}_{i_k}(f)$.

This corollary implies the following version of Corollary 19 for set-multilinear polynomials.

Theorem 21. *Let* $X = \dot{\cup}_{i=1}^{d} X_i$*, with* $|X| = n$*. Let* $f(X_1, \ldots, X_d)$ *be a set-multilinear polynomial. Then*

$$\dim(H(f)) \leq n \sum_{k=0}^{d} \text{s-rank}_k(f).$$

Proof. According to Theorem 17 we have that $\dim(H(f)) = \sum_{k=0}^{n} \mathrm{rank}(E_k)$. By corollary 20 we get that $\mathrm{rank}(E_k) \leq \text{s-rank}_{i_k}(f)$, where i_k is such that $\left|\cup_{i=1}^{i_k} X_i\right| \leq k < \left|\cup_{i=1}^{i_k+1} X_i\right|$. In particular we get that

$$\dim(H(f)) = \sum_{k=0}^{n} \mathrm{rank}(E_k) \leq^{(*)} \sum_{i=0}^{d} |X_{i+1}| \cdot \text{s-rank}_i(f) \leq n \sum_{i=0}^{d} \text{s-rank}_i(f),$$

where inequality $(*)$ follows from the observation that for every k, such that $\left|\cup_{j=1}^{i} X_j\right| \leq k < \left|\cup_{j=1}^{i+1} X_j\right|$, it holds that $i_k = i$, and so there are $|X_{i+1}|$ such k's. □

4 Learning Depth-3 Arithmetic Circuits

In this section we learn depth-3 arithmetic circuits. The results that we obtain also follow from the works of [6, 4], however we reprove them in order to demonstrate the usefulness of our techniques. We begin by defining the model:

Definition 22. *A depth 3 arithmetic circuit is a layered graph with 3 levels and unbounded fan-in. At the top we have either a sum gate or a product gate. A depth 3 arithmetic circuit* $\mathcal{C}$ *with a sum (product) gate at the top is called a* $\Sigma\Pi\Sigma$ *(*$\Pi\Sigma\Pi$*) circuit and has the following structure:*

$$\mathcal{C} = \sum_{i=1}^{m} \prod_{j=1}^{d} L_{i,j}(X)$$

where each $L_{i,j}$ *is a linear function in the input variables* $X = x_1, \ldots, x_n$ *and* m *is the number of multiplication gates. The size of the circuit is the number of gates, in this case* $O(md)$*.*

A $\Sigma\Pi\Sigma$ circuit is a homogeneous circuit if all the linear forms are homogeneous linear forms (i.e. the free term is zero) and all the product gates have the same fan in (or degree). In other words every gate of the circuit computes a homogeneous polynomial. We will also be interested in set-multilinear depth 3 circuits. To define this sub-model we need to impose a partition on the variables:

Definition 23. *A* $\Sigma\Pi\Sigma$ *circuit is called set-multilinear with respect to* $X = \dot{\cup}_{i=1}^{d} X_i$ *if every linear function computed at the bottom is a homogeneous linear form in one of the sets* X_i*, and each multiplication gate multiplies* d *homogeneous linear forms* $L_1, \ldots, L_d$ *where every* L_i *is over a distinct set of variables* X_i*. That is to say a depth-3 set-multilinear circuit* $\mathcal{C}$ *has the following structure:*

$$\mathcal{C} = \sum_{i=1}^{m} \prod_{j=1}^{d} L_{i,j}(X_j)$$

where $L_{i,j}$ *is an homogeneous linear form.*

We now give an algorithm for learning set-multilinear depth-3 circuits. The algorithm is based on the following lemma that characterizes the dimension of a set-multilinear circuit's partial derivatives:

Lemma 24. *If a polynomial f is computed by a set-multilinear depth 3 circuit with m product gates then for every $1 \leq k \leq d$,*

$$\text{s-rank}_k(f) \leq km.$$

Proof. First notice that for every product gate

$$P = \prod_{i=1}^{d} L_i(X_i)$$

we have $\text{s-rank}_k(P) \leq k$. Indeed, let $1 \leq r \leq k$. then for any monomial $M \in \mathbb{SM}[X_1, ..., X_r]$ we have that

$$\frac{\partial P}{\partial M} = \alpha_M \cdot \prod_{i=r+1}^{d} L_i(X_i)$$

for some constant α_M depending on M and P. Thus, as we vary over all r between 1 and k we obtain only k distinct partial derivatives. The proof of the lemma now follows from the linearity of the partial derivative operator. □

Applying Lemma 24, Theorem 21, and Theorem 4 we obtain the following learning result:

Theorem 25. *Let $\mathcal{C}$ be a set-multilinear depth-3 circuit with m product gates over n variables with coefficients from a field $\mathbb{F}$. Then $\mathcal{C}$ is learnable in time polynomial in m and n.*

We note that this result also follows immediately from the results of [6, 4].

4.1 Learning General Depth 3 Circuits

We now give our learning algorithm for general depth-3 arithmetic circuits. Unlike the algorithm in the set-multilinear case, this algorithm runs in time exponential in the degree of the circuit (and polynomial in the other parameters). Thus we can learn in subexponential time any depth-3 circuit of sublinear degree. The running time of the algorithm is determined by the following lemma:

Lemma 26. *Let $f : \mathbb{F}^n \to \mathbb{F}$ be a polynomial over a variable set X of size n computed by a depth-3 circuit with m product gates each of degree at most d. Then for every $1 \leq k \leq d$,*

$$\text{rank}_k(f) \leq m \cdot \sum_{i=1}^{k} \binom{d}{i}.$$

Proof. The proof is similar to the case of set-multilinear depth-3 circuits. Notice that for every product gate

$$P = \prod_{i=1}^{d} L_i(X)$$

we have $\text{rank}_k(P) \leq \sum_{i=1}^{k} \binom{d}{i}$. Indeed, for any monomial M of degree r we have that

$$\frac{\partial P}{\partial M} \in \text{span}\left\{ \prod_{i \in T} L_i(X) \mid T \subset [d],\ |T| = d - r \right\}.$$

Since there are at most m product gates we obtain the claimed bound. □

Applying the above lemma with Theorem 17 and Theorem 4 we get the following learning result (that was also obtained in [6]):

Theorem 27. *Let $f : \mathbb{F}^n \to \mathbb{F}$ be computed by a depth-3 arithmetic circuit with m product gates each of fan in at most d. Then f is learnable in time polynomial in $n, 2^d$, and m.*

4.2 Discussion

The fact that the rank of f was bounded by the number of product gates is unique to set-multilinear depth-3 circuits. For example consider the following depth-2 $\Pi\Sigma$ circuit:

$$f(z, x_1, ..., x_n) = \prod_{i=1}^{n} (z + x_i).$$

For every ordering of the variables, the dimension of the span of the partial derivatives of f (and hence the rank of the Hankel matrix of f) is exponential in n; this follows from the observation that the coefficient of z^d is the $n - d$ symmetric polynomial whose partial derivatives have dimension $2^{\Omega(n-d)}$ (see [33]). Thus it is no surprise that Beimel *et al.* [4] only considered depth-2 $\Pi\Sigma$ circuits where the product gate at the root has fan in at most $O(\log n)$; fan in larger than $O(\log n)$ would correspond to Hankel matrices of superpolynomial dimension and thus would not be learnable by multiplicity automata techniques.

To show the limits of current learning techniques we point out that the following homogeneous depth-3 arithmetic circuit

$$C' = \prod_{i=1}^{n} (z + x_i) + \prod_{i=1}^{n} (v + u_i)$$

is both irreducible and has exponentially many linearly independent partial derivatives. As its degree is n we can only learn it in time exponential in n. We leave open the problem of learning homogeneous depth-3 arithmetic circuits (as well as the more difficult problem of learning general depth-3 arithmetic circuits) of super logarithmic degree.

5 Learning Classes of Algebraic Branching Programs

Algebraic and Boolean branching programs have been intensely studied by complexity theorists and have been particularly fruitful for proving lower bounds. Considerably less is known in the learning scenario — Bshouty *et al.* [12] and Bergadano *et al.* [5] have shown some partial progress for learning restricted width Boolean branching programs. In this section we will show how to learn any polynomial size algebraic branching program that is both read once and oblivious. As such we will be able to show that multiplicity automata are essentially equivalent to read once, oblivious algebraic branching programs, a characterization that may be of independent interest. We begin with a general definition of algebraic branching programs:

Definition 28. *An algebraic branching program (ABP), first defined by Nisan [25], is a directed acyclic graph with one vertex of in-degree zero, which is called* source*, and one vertex of out-degree zero, which is called the* sink*. The vertices of the graph are partitioned into levels numbered $0, ..., d$. Edges are labeled with a homogeneous linear form in the input variables and may only connect vertices from level i to vertices from level $i + 1$. The source is the only vertex at level 0 and the sink is the only vertex at the level d. Finally the size of the ABP is the number of vertices in the graph.*

The polynomial that is computed by an ABP is the sum over all directed paths from the source to the sink of the product of linear functions that labeled the edges of the path. It is clear that an ABP with $d+1$ levels computes a homogeneous polynomial of degree d.

In this section we will show how to learn a natural restriction of an algebraic branching program as mentioned above: the read once, oblivious algebraic branching program or ROAB.

Definition 29. *Let $X = \dot{\cup}_{i=1}^{d} X_i$ be a partition of the input variables into d disjoint sets. An ABP is* oblivious *if for every level i only one set of variables X_j appears. A function is a ROAB, a read once, oblivious algebraic branching program, if it is an oblivious ABP and every set of variables X_j appears in at most one level.*

We are interested in learning ROABs with respect to the partition $X = \dot{\cup}_{i=1}^{d} X_i$ in which the variables in X_i appear on edges from level i to level $i+1$. In this section we measure the complexity of a polynomial in terms of its smallest ROAB:

Definition 30. *For a polynomial f we define $B(f)$ to be the size of the smallest ABP for f. For a set-multilinear polynomial f we denote $OB(f)$ to be the size of the smallest ROAB for f.*

The main theorem of this section shows that for set-multilinear polynomials, the size of its smallest ROAB is equal to the dimension of the vector space induced by its partial derivatives:

Theorem 31. *For a set-multilinear polynomial $f(X_1, ..., X_d)$ we have that*

$$\sum_{k=1}^{d} \text{s-rank}_k(f) = OB(f).$$

To prove Theorem 31 we will need the following theorem which is implicit in Nisan [25]:

Theorem 32. *[25] Let $f(X_1, \ldots, X_d)$ be a set-multilinear polynomial. For each $0 \le k \le d$ define a matrix $\mathcal{M}_k(f)$ as follows:*

- *Each row is labeled with a monomial $M_1 \in \mathbb{SM}[X_1, \ldots, X_k]$.*
- *Each column is labeled with a monomial $M_2 \in \mathbb{SM}[X_{k+1}, \ldots, X_d]$.*

If $k = 0$ then $M_1 = 1$ and if $k = d$ then $M_2 = 1$. The (M_1, M_2) entry of $\mathcal{M}_k(f)$ is equal to the coefficient of the monomial $M_1 \cdot M_2$ in f. We have that

$$OB(f) = \sum_{k=0}^{d} \text{rank}(\mathcal{M}_k(f)).$$

Proof of Theorem 31: We will show that $\text{rank}(\mathcal{M}_k(f)) = \text{s-rank}_k(f)$ which, combined with Theorem 32, completes the proof. Consider a row of $\mathcal{M}_k(f)$ corresponding to some monomial $M \in \mathbb{SM}[X_1, \ldots, X_k]$. Since f is a set-multilinear polynomial it follows that $\frac{\partial f}{\partial M}$ is equal to $\sum_t \alpha_t M_t$ where each α_t is an element of the field and $M_t \in \mathbb{SM}[X_{k+1}, \ldots, X_d]$, for all t. Notice, however that row M of $\mathcal{M}_k(f)$ is precisely equal to the row vector $(\alpha_1, \ldots, \alpha_t)$. Hence row M of $\mathcal{M}_k(f)$ is equal to the coefficients of the partial derivative of f viewed as a set-multilinear polynomial in $X_{k+1}, \ldots, X_d$. It is a standard fact from linear algebra that the dimension of a vector space spanned by a set of polynomials is equal to the rank of the matrix of their coefficients. □

Combining Theorem 31 and Theorem 21 we see that any polynomial-size ROAB obeying the above partition is computed by a polynomial-size multiplicity automata. Applying the learning algorithm of Beimel *et al.* [4] we obtain

Theorem 33. *Let $X = \dot\bigcup_{i=1}^{d} X_i$. Let $f(X_1, \dots, X_d)$ be a set-multilinear polynomial that is computed by a ROAB of size m. Then f is learnable in time polynomial in m, and $|X|$.*

Notice that ROABs can be thought of as the arithmetic generalization of OBDDs (Ordered Binary Decision Diagrams, which are also known as oblivious read once branching programs), a model for which Bergadano *et al.* [5] gave a learning algorithm based on multiplicity automata.

5.1 Equivalence of ROABs and Multiplicity Automata

We can now prove that ROABs are essentially equivalent to multiplicity automata. Since our learning algorithm outputs as a hypothesis a multiplicity automaton, Theorem 33 implies that every ROAB of size m in n variables is computed by a multiplicity automaton of size polynomial in m and n. We cannot show that every multiplicity automaton is computed by a ROAB, but we can show that every multiplicity automaton is computed by a ROAB which computes higher degree polynomials at each edge.

Definition 34. *Define a ROAB of degree d to be a ROAB where every edge is labelled with a polynomial of degree d.*

Lemma 35. *Let f be any polynomial over n variables computed by an algebraic multiplicity automaton of size r. Assume also the the degree of each variable in f is bounded by d. Then f can be computed by a ROAB with $n+2$ levels of size $nr+2$ and degree d.*

Proof. Let $S \subseteq \Sigma$ be a subset of the alphabet of size $d+1$. Let f be computed by a multiplicity automaton A of size r consisting of the set of matrices $\{\mu_\sigma\}_{\sigma \in \Sigma}$ and the vector $\vec{\gamma} \in \Sigma^r$. Construct a matrix T where the i,j entry of T is a degree d univariate polynomial, $T_{i,j}$, interpolating the (i,j) entry of μ_σ for every $\sigma \in S$. That is, $T_{i,j}(\sigma)$ is the (i,j) entry of μ_σ (for $\sigma \in S$). Consider a ROAB with $n+2$ levels each of size r where every level $1 \le i \le n$ has a copy of the r states of A (in particular we enumerate the vertices in each level with $\{1, \dots, r\}$). Connect every vertex at level k, for $1 \le k \le n-1$, to every vertex at level $k+1$. For the j-th vertex in level k and the i-th vertex in level $k+1$ we label edge (j,i) with the polynomial in the (i,j) entry of T, having x_k as its variable (i.e. the label is $T_{i,j}(x_k)$). Connect every vertex in level n to the sink and label edge (i, sink) with the polynomial in the $T_{1,i}(x_n)$ (recall the output of a multiplicity automata is the inner product of $\vec{\gamma}$ with the first row of the product of the μ_σ's). Also connect the source to every one of the r vertices in the first level and label the edge to vertex i with γ_i. It is clear that this ROAB computes a polynomial of degree at most d in each variable, and that for every input from S^n the output of the ROAB agrees with f. Therefore, by the following version of the Schwartz Zippel Lemma [32, 36] we get that this ROAB computes f as required.

Lemma 36 ([32, 36])**.** *Let $f, g : \mathbb{F}^n \to \mathbb{F}$ be two n-variate polynomials over $\mathbb{F}$. Assume that the degree of each variable in f and g is at most d. Let $S \subseteq \mathbb{F}$ be a set of size $d+1$. If for every $x \in S^n$ we have that $f(x) = g(x)$ then $f = g$.*

□

6 Learning Noncommutative Formulae

In this section we show how to learn another type of arithmetic circuits: polynomial size noncommutative formulae. A noncommutative formula is an arithmetic formula where multiplication does not necessarily commute; i.e. different orderings of inputs to a product gate result in different outputs. Intuitively this restriction makes it difficult for a formula to use the power of cancellation. This may seem to be a strange restriction, but it is very natural in the context of function computation where an ordering is enforced on groups of variables. For example, the product of k matrices $M_1, \ldots, M_k$ where matrix M_i uses variables from a set X_i can be viewed as a set-multilinear non-commutative polynomial over an ordering of the variables $X = \dot{\cup} X_i$ (changing the order of the matrices will result in a different output). In addition, many of the known algorithms for computing polynomials are non-commutative by nature. For example, the well known algorithm for the above mentioned iterated matrix multiplication can be viewed as a non-commutative set-multilinear circuit. Similarly, Ryser's algorithm for computing the permanent (see, e.g. [35]) can be viewed as a non-commutative set-multilinear formula.

Nisan proved the first lower bounds for noncommutative formulae in [25]; here we will give the first learning algorithm for set-multilinear polynomials computed by noncommutative formulae. Previously only algorithms for learning read-once arithmetic formulae were known (see e.g. [18, 10, 9, 8]). We begin with a general definition for arithmetic formulae:

Definition 37. *An arithmetic formula is a tree whose edges are directed towards the root. The leaves of the tree are labeled with input variables. Every inner vertex is labeled with one of the arithmetic operations* $\{+, \times\}$. *Every edge is labeled with a constant from the field in which we are working. The size of the formula is defined to be the number of vertices.*

An arithmetic formula computes a polynomial in the obvious manner. We now define non-commutative formulas. Roughly, a formula is noncommutative if for any two input variables x_i and x_j, $x_i x_j - x_j x_i \neq 0$. More formally, let $\mathbb{F}\{x_1, ..., x_n\}$ be the polynomial ring over the field $\mathbb{F}$ in the non-commuting variables $x_1, \ldots, x_n$. That is, in $\mathbb{F}\{x_1, ..., x_n\}$ the formal expressions $x_{i_1} \cdot x_{i_2} \cdot \ldots \cdot x_{i_k}$ and $x_{j_1} \cdot x_{j_2} \cdot \ldots \cdot x_{j_l}$ are equal if and only if $k = l$ and $\forall m \; i_m = j_m$ (whereas in the commutative ring of polynomials we have that any monomial remains the same even if we permute its variables, e.g. $x_1 \cdot x_2 = x_2 \cdot x_1$). A non-commutative arithmetic formula is an arithmetic formula where multiplications are done in the ring $\mathbb{F}\{x_1, ..., x_n\}$. As two polynomials in this ring do not necessarily commute, we have to distinguish in every multiplication gate between the left son and the right son. For a polynomial f let $F(f)$ be the size of the smallest noncommutative formula computing f. When considering non-commutative formulas we are interested in *syntactic* computations, e.g. given the polynomial $x_1 \cdot x_2$ we want the formula to output this exact polynomial and not the polynomial $x_2 \cdot x_1$, even though they are semantically equal when considering assignments from a field. In particular the formula $(x_1 - x_2) \cdot (x_1 + x_2)$ does not compute the polynomial $x_1^2 - x_2^2$.

Note that every polynomial can be computed by a non-commutative formula, and that given a non-commutative formula we can evaluate it over a commutative domain.

In [25] Nisan proved exponential lower bounds on the size of noncommutative formula computing the permanent and the determinant. An important ingredient of Nisan's result is the following lemma relating noncommutative formula size to algebraic branching program size:

Lemma 38. *[25] Let* $f(X_1, \ldots, X_d)$ *be a set-multilinear polynomial. Then*

$$B(f) \leq d(F(f) + 1)$$

Using this we can give the following relationship between noncommutative formulae and ROABs:

Theorem 39. *Let* $f(X_1, \ldots, X_d)$ *be a set-multilinear polynomial computed by a noncommutative formula of size* m*, then* f *is computed by a ROAB of size* $d(m + 1)$.

Proof. Applying Lemma 38 we see that f is computed by an algebraic branching program B of size $d(m+1)$. We will show that B is also computed by a ROAB of size $d(m+1)$, by constructing a ROAB with $d+1$ levels, in which the variables in X_k label the edges that go from level $k-1$ to level k.

Consider the set of edges in B from level $i-1$ to i. Assume that two different sets of variables appear from level $i-1$ to level i say X_i and X_j. Then the output of B will contain a monomial of the form and Yx_jZ where $x_j \in X_j$, Y is a set of variables appearing in levels less than $i-1$ in B, and Z is the set of variables appearing in levels greater than i in B. Note however that f is a set-multilinear polynomial, in particular in each monomial of f the variables from X_j appear as the j-th multiplicand. In particular no monomial of the form Yx_jZ appear in f. Thus, the coefficient of any monomial Yx_jZ must be zero. As such, we can substitute the constant 0 for all of the variables X_j appearing on these edges and obtain an oblivious branching program B' computing the same polynomial as B. B' can be made read-once in a similar fashion. At the end we get a ROAB with $d+1$ levels in which the variables from X_i label the edges from level $i-1$ to level i. □

Combining Theorem 39 with Theorem 33 we obtain

Theorem 40. *Let $f(X_1, \dots, X_d)$ be a set-multilinear polynomial, over $X = \dot\cup_{i=1}^{d} X_i$, that is computable by a noncommutative formula of size m with coefficients from a field $\mathbb{F}$. Then f is learnable in time polynomial in $|X|$ and m.*

7 Acknowledgments

We thank Ran Raz for many helpful discussions in all stages of this work. We also thank Eli Ben-Sasson for important conversations at an early stage of this research. We thank the anonymous referees for their valuable comments, and for bringing [6] to our attention.

References

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

[2] S. Arora, C. Lund, R. Motwani, M. Sudan, and M. Szegedy. Proof verification and the hardness of approximation problems. *Journal of the ACM*, 45(3):501–555, 1998.

[3] J. Aspnes, R. Beigel, M. Furst, and S. Rudich. The expressive power of voting polynomials. In *Proc. 23rd STOC*, pages 402–409. ACM Press, 1991.

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

[5] F. Bergadano, N. H. Bshouty, C. Tamon, and S. Varricchio. On learning branching programs and small depth circuits. In *Proc. of the 3rd European Conf. on Computational Learning Theory (EuroCOLT'97)*, volume 1208 of *LNCS*, pages 150–161, 1997.

[6] F. Bergadano, N. H. Bshouty, and S. Varricchio. Learning multivariate polynomials from substitution and equivalence queries. *Electronic Colloquium on Computational Complexity*, 3(8), 1996.

[7] F. Bergadano and S. Varricchio. Learning behaviors of automata from multiplicity and equivalence queries. *SIAM Journal on Computing*, 25(6):1268–1280, 1996.

[8] D. Bshouty and N. H. Bshouty. On interpolating arithmetic read-once formulas with exponentiation. *Journal of Computer and System Sciences*, 56(1):112–124, 1998.

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

[10] N. H. Bshouty, T. R. Hancock, and L. Hellerstein. Learning boolean read-once formulas over generalized bases. *Journal of Computer and System Sciences*, 50(3):521–542, 1995.

[11] N. H. Bshouty and Y. Mansour. Simple learning algorithms for decision trees and multivariate polynomials. *SIAM Journal on Computing*, 31(6):1909–1925, 2002.

[12] N. H. Bshouty, C. Tamon, and D. K. Wilson. On learning width two branchinng programs. *Information Processing Letters*, 65(4):217–222, 1998.

[13] J. W. Carlyle and A. Paz. Realizations by stochastic finite automata. *Journal of Computer and System Sciences*, 5(1):26–40, 1971.

[14] M. Fliess. Matrices de Hankel. *Journal de Mathématiques Pures et Appliquées*, 53:197–224, 1974.

[15] D. Grigoriev and M. Karpinski. An exponential lower bound for depth 3 arithmetic circuits. In *Proc. 30th STOC*, pages 577–582. ACM Press, 1998.

[16] D. Grigoriev, M. Karpinski, and M. F. Singer. Computational complexity of sparse rational interpolation. *SIAM Journal on Computing*, 23(1):1–11, 1994.

[17] D. Grigoriev and A. A. Razborov. Exponential complexity lower bounds for depth 3 arithmetic circuits in algebras of functions over finite fields. In *Proc. 39th FOCS*, pages 269–278. IEEE Computer Society Press, 1998.

[18] T. R. Hancock and L. Hellerstein. Learning read-once formulas over fields and extended bases. In *Proc. of the 4th Ann. Conf. on Computational Learning Theory (COLT '91)*, pages 326–336. Morgan Kaufmann, 1991.

[19] J. Håstad. Almost optimal lower bounds for small depth circuits. In *Proc. 18th STOC*, pages 6–20. ACM Press, 1986.

[20] M. A. Huang and A. J. Rao. Interpolation of sparse multivariate polynomials over large finite fields with applications. *Journal of Algorithms*, 33(2):204–228, 1999.

[21] J. C. Jackson, A. R. Klivans, and R. A. Servedio. Learnability beyond AC^0. In *Proc. 34th STOC*, pages 776–784. ACM Press, 2002.

[22] A. Klivans and A. Shpilka. Learning arithmetic circuits via partial derivatives. In *Proc. of the 16th Ann. Conf. on Learning Theory (COLT '03)*, pages 463–476, 2003.

[23] A. R. Klivans and D. Spielman. Randomness efficient identity testing of multivariate polynomials. In *Proc. 33rd STOC*, pages 216–223. ACM Press, 2001.

[24] N. Linial, Y. Mansour, and N. Nisan. Constant depth circuits, Fourier transform and learnability. *Journal of the ACM*, 40(3):607–620, 1993.

[25] N. Nisan. Lower bounds for non-commutative computation. In *Proc. 23rd STOC*, pages 410–418, 1991.

[26] N. Nisan and A. Wigderson. Lower bounds on arithmetic circuits via partial derivatives. *Computational Complexity*, 6(3):217–234, 1997.

[27] H. Ohnishi, H. Seki, and T. Kasami. A polynomial time learning algorithm for recognizable series. *IEICE Transactions on Information and Systems*, E77-D(10):1077–1085, 1994.

[28] R. Raz. Multi-linear formulas for permanent and determinant are of super-polynomial size. In *Proc. 36th STOC*, pages 633–641. ACM Press, 2004.

[29] R. Raz. Separation of multilinear circuit and formula size. *Theory of Computing*, 2(6):121–135, 2006. Preliminary version appeared in FOCS'04, pp. 344–351.

[30] R. Raz and A. Shpilka. Deterministic polynomial identity testing in non-commutative models. *Computational Complexity*, 14(1):1–19, 2005.

[31] R. E. Schapire and L. M. Sellie. Learning sparse multivariate polynomials over a field with queries and counterexamples. *Journal of Computer and System Sciences*, 52(2):201–213, 1996.

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

[33] A. Shpilka and A. Wigderson. Depth-3 arithmetic circuits over fields of characteristic zero. *Computational Complexity*, 10(1):1–27, 2001.

[34] M. Sudan, L. Trevisan, and S. P. Vadhan. Pseudorandom generators without the XOR lemma. *Journal of Computer and System Sciences*, 62(2):236–266, 2001.

[35] J. H. van Lint and R. M. Wilson. *A Course in Combinatorics.* Cambridge University Press, 2001.

[36] R. Zippel. Probabilistic algorithms for sparse polynomials. In *Proc. Intern. Symp. on Symbolic and Algebraic Manipulation*, volume 72 of *Lecture Notes in Computer Science*, pages 216–226. Springer, 1979.