Limits on Representing Boolean Functions by Linear Combinations of Simple Functions: Thresholds, ReLUs, and Low-Degree Polynomials

Richard Ryan Williams[1]
EECS and CSAIL, MIT, 32 Vassar St., Cambridge MA, USA
rrw@mit.edu
https://orcid.org/0000-0003-2326-2233

Abstract

We consider the problem of representing Boolean functions exactly by "sparse" linear combinations (over $\mathbb{R}$) of functions from some "simple" class $\mathcal{C}$. In particular, given $\mathcal{C}$ we are interested in finding low-complexity functions lacking sparse representations. When $\mathcal{C}$ forms a basis for the space of Boolean functions (e.g., the set of PARITY functions or the set of conjunctions) this sort of problem has a well-understood answer; the problem becomes interesting when $\mathcal{C}$ is "overcomplete" and the set of functions is not linearly independent. We focus on the cases where $\mathcal{C}$ is the set of linear threshold functions, the set of rectified linear units (ReLUs), and the set of low-degree polynomials over a finite field, all of which are well-studied in different contexts.

We provide generic tools for proving lower bounds on representations of this kind. Applying these, we give several new lower bounds for "semi-explicit" Boolean functions. Let $\alpha(n)$ be an unbounded function such that $n^{\alpha(n)}$ is time constructible (e.g. $\alpha(n) = \log^\star(n)$). We show:

- Functions in $\mathsf{NTIME}[n^{\alpha(n)}]$ that require super-polynomially many linear threshold functions to represent (depth-two neural networks with sign activation function, a special case of depth-two threshold circuit lower bounds).
- Functions in $\mathsf{NTIME}[n^{\alpha(n)}]$ that require super-polynomially many ReLU gates to represent (depth-two neural networks with ReLU activation function).
- Functions in $\mathsf{NTIME}[n^{\alpha(n)}]$ that require super-polynomially many $O(1)$-degree $\mathbb{F}_p$-polynomials to represent exactly, for every prime p (related to problems regarding Higher-Order "Uncertainty Principles"). We also obtain a function in E^{NP} requiring $2^{\Omega(n)}$ linear combinations.
- Functions in $\mathsf{NTIME}[n^{\mathrm{poly}(\log n)}]$ that require super-polynomially many $\mathsf{ACC} \circ \mathsf{THR}$ circuits to represent exactly (further generalizing the recent lower bounds of Murray and the author).

We also obtain "fixed-polynomial" lower bounds for functions in NP, for the first three representation classes. All our lower bounds are obtained via algorithms for *analyzing* linear combinations of simple functions in the above scenarios, in ways which substantially beat exhaustive search.

2012 ACM Subject Classification Theory of computation → Circuit complexity, Computer systems organization → Neural networks

Keywords and phrases linear threshold functions, lower bounds, neural networks, low-degree polynomials

Digital Object Identifier 10.4230/LIPIcs.CCC.2018.6

Acknowledgements I thank Lijie Chen, Pooya Hatami, Adam Klivans, and Shachar Lovett for useful discussions on the topics of this paper. In particular, I am grateful to Shachar for noticing

[1] Supported by NSF CCF-1553288. Any opinions, findings and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation.

33rd Computational Complexity Conference (CCC 2018).
Editor: Rocco A. Servedio; Article No. 6; pp. 6:1–6:24

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

a gap in a lemma in an earlier version of this paper. I am also grateful to Brynmor Chapman for his proofreading, and patience with my explanations regarding this paper. Thanks also to the CCC'18 reviewers for their comments and corrections.

1 Introduction

Given $f : \{0,1\}^n \to \{0,1\}$ and a class $\mathcal{C}$ of "simple" functions, when can f be represented exactly as a *short* $\mathbb{R}$-linear combination of functions from $\mathcal{C}$? When $\mathcal{C}$ forms a basis for B_n (the set of all Boolean functions on n inputs) the question has a unique answer that is generally easy to obtain, by analyzing the appropriate linear system (the cases where $\mathcal{C}$ is the set of all parity functions or the set of all conjunctions are canonical examples). For $|\mathcal{C}| \gg 2^n$, the situation becomes much more interesting, as there can be many possible representations. The general problem of understanding which functions do and do not have sparse representations for simple $\mathcal{C}$ arises in many different mathematical topics. Three relevant to TCS are depth-two threshold circuits, depth-two neural networks with various activation functions, and higher-order Fourier analysis. We use the notation

$$\mathsf{SUM} \circ \mathcal{C}$$

to denote the class of $\mathbb{R}$-linear combinations of $\mathcal{C}$-functions; for example, $\mathsf{SUM} \circ \mathsf{MOD2}$ denotes $\mathbb{R}$-linear combinations of PARITY functions. The relevant complexity measure for a "circuit" in $\mathsf{SUM} \circ \mathcal{C}$ is the fan-in of the SUM gate, which we call the *sparsity* of the circuit.

Sums of Threshold Circuits

Let $\mathsf{SUM} \circ \mathsf{THR}$ be linear combinations of linear threshold functions (LTFs).[2] As there are $2^{\Theta(n^2)}$ n-variate threshold functions [55], a function $f : \{0,1\}^n \to \{0,1\}$ has *many* possible representations as a $\mathsf{SUM} \circ \mathsf{THR}$. Such circuits are also known in the machine learning literature as *depth-two neural networks with sign activation functions*.

In 1994, Roychowdhury, Orlitsky, and Siu [38] noted that no interesting size lower bounds were known for computing Boolean functions with $\mathsf{SUM} \circ \mathsf{THR}$ circuits (beyond the few that are/were known for $\mathsf{THR} \circ \mathsf{THR}$ [22, 38, 28, 14, 43, 2]). The problem was raised again more recently in CCC'10 by Hansen and Podolskii [23]. In particular, the following remains largely unanswered:

> **Problem:** *Find an explicit $f : \{0,1\}^\star \to \{0,1\}$ without polynomially-sparse $\mathsf{SUM} \circ \mathsf{THR}$, i.e., every linear combination of LTFs computing f on n-bit inputs needs $n^{\omega(1)}$ LTFs, for infinitely many n.*

Because of prior lower bounds in weaker settings (such as majority-of-majority [22] and majority-of-thresholds [36]), it is natural to think that correlation bounds against linear threshold functions should help.[3] Correlation bounds do imply lower bounds for $\mathsf{SUM} \circ \mathsf{THR}$, but only when the weights in the linear combination are not too large (i.e., the weights must be in $[-2^{\delta n}, 2^{\delta n}]$ for small $\delta < 1$). However, if arbitrary weights are allowed, interesting lower bounds on $\mathsf{SUM} \circ \mathsf{THR}$ (beyond $\Omega(n^{2.5}$ wires [28]) were open, to the best of our knowledge. In Section 4, we prove arbitrary polynomial lower bounds for NP functions:

---

[2] From here on, "linear combination" means "$\mathbb{R}$-linear combination", unless otherwise specified.

[3] That is, one wants to show that a function cannot be $(1/2 + \varepsilon(n))$-approximated by a linear threshold function, for the tiniest $\varepsilon(n) > 0$ possible.

▶ **Theorem 1.** *For all k, there is an $f_k \in \mathsf{NP}$ without $\mathsf{SUM} \circ \mathsf{THR}$ circuits of n^k sparsity. Furthermore, for every unbounded $\alpha(n)$ such that $n^{\alpha(n)}$ is time constructible, there is a function in $\mathsf{NTIME}[n^{\alpha(n)}]$ that does not have $\mathsf{SUM} \circ \mathsf{THR}$ circuits of polynomial sparsity.*

Note that for arbitrary circuits (even for $\mathsf{THR} \circ \mathsf{THR}$ circuits) the best known complexity for such functions without n^k-size circuits (for fixed k) is $\mathsf{MA/1}$ ([40]) and S_2^p.

Sums of ReLU Gates

A ReLU (rectified linear unit) gate is a function $f : \{0,1\}^t \to \mathbb{R}^+$ such that there is a vector $w \in \mathbb{R}^t$ and scalar $a \in \mathbb{R}$ such that for all x,

$$f(x) = \max\{0, \langle x, w\rangle + a\}.$$

It is important to note that ReLU gates might not be Boolean-valued, but they must output non-negative numbers on all Boolean inputs. Linear combinations of ReLU gates are also known as *depth-two neural networks with ReLU activation functions*, and they are intensely studied in machine learning. Several lower bounds for Sums-of-ReLU functions (which for consistency we call $\mathsf{SUM} \circ \mathsf{ReLU}$) have recently been shown for functions with *real-valued* inputs and outputs (examples include [16, 44, 3, 15, 39]) but none of the methods extend to Boolean functions, to the best of our knowledge. Recently, Mukherjee and Basu [33] have proved $\Omega(n^{1-\delta})$-gate lower bounds for $\mathsf{SUM} \circ \mathsf{ReLU}$ circuits computing the Andreev function, extending ideas in [28, 13].

Observing that for $|\langle x, w\rangle| \geq 1$ we have

$$\max\{0, \langle x, w\rangle + 1\} - \max\{0, \langle x, w\rangle\} = \mathrm{sign}(\langle x, w\rangle),$$

it follows that every $\mathsf{SUM} \circ \mathsf{THR}$ circuit can be simulated by a $\mathsf{SUM} \circ \mathsf{ReLU}$ circuit with only a doubling of the sparsity. In Section 5 we extend our lower bounds to Sums-of-ReLU circuits:

▶ **Theorem 2.** *For all k, there is an $f_k \in \mathsf{NP}$ without $\mathsf{SUM} \circ \mathsf{ReLU}$ circuits of n^k sparsity. Furthermore, for every unbounded $\alpha(n)$ such that $n^{\alpha(n)}$ is time constructible, there is a function in $\mathsf{NTIME}[n^{\alpha(n)}]$ that does not have $\mathsf{SUM} \circ \mathsf{ReLU}$ circuits of polynomial sparsity.*

Representing Boolean Functions With Higher-Order Polynomials

Higher-order Fourier analysis of Boolean functions deals with representing Boolean functions by $\mathbb{R}$-linear combinations of $\mathbb{F}_2$-polynomials of degree higher than one (see [25] for a survey of some applications in CS theory). The question of which (if any) explicit functions lack *sparse* representations, even for degree-two polynomials, has been wide open. Letting $\mathsf{MOD2}$ be the class of parity functions, this question asks to find lower bounds for $\mathsf{SUM} \circ \mathsf{MOD2} \circ \mathsf{AND}_2$ circuits (in our notation, AND_k denotes ANDs of fan-in at most k). Such lower bound problems appear much more difficult than the degree-one case of $\mathsf{SUM} \circ \mathsf{MOD2}$. Even understanding the sparsity of the AND function in the quadratic (and in general, degree-$O(1)$) setting is a prominent open problem:

▶ **Hypothesis 3** (Quadratic Uncertainty Principle [17])**.** *There is an $\varepsilon > 0$ such that the AND function on n variables does not have $\mathsf{SUM} \circ \mathsf{MOD2} \circ \mathsf{AND}_2$ circuits of $2^{\varepsilon n}$ sparsity.*

Although it is believed that AND needs exponential sparsity, to our knowledge the *only* lower bound known for an explicit function in $\mathsf{SUM} \circ \mathsf{MOD2} \circ \mathsf{AND}_2$ was $\Omega(n)$-sparsity. For completeness we include a proof provided to us by Lovett [30]) in Appendix A. Again, when

the weights in the linear combination are required to be small (magnitudes are $2^{\varepsilon n}$ for small $\varepsilon > 0$), correlation bounds yield some results: one example (among many) is the work of Green [20] showing that a majority vote of quadratic $\mathbb{F}_3$-polynomials needs $2^{\Omega(n)}$ polynomials to compute PARITY. (Other works in this vein include [24, 10, 9, 19]; see Viola [49] for a survey.) However, for arbitrary weights, no non-trivial lower bounds have been reported (to our knowledge).

In Section 6, we prove polynomial sparsity lower bounds for Boolean functions in NP and $2^{\Omega(n)}$-size lower bounds for E^{NP}, against linear combinations of polynomials over any prime field with any constant degree:

▶ **Theorem 4.** *For every integer $k, d \geq 1$ and prime p, there is an $f_k \in \mathsf{NP}$ without $\mathsf{SUM} \circ \mathsf{MODp} \circ \mathsf{AND}_d$ circuits of n^k sparsity. Furthermore, for every unbounded $\alpha(n)$ such that $n^{\alpha(n)}$ is time constructible, there is a function in $\mathsf{NTIME}[n^{\alpha(n)}]$ that does not have $\mathsf{SUM} \circ \mathsf{MODp} \circ \mathsf{AND}_d$ circuits of polynomial sparsity.*

▶ **Theorem 5.** *For every $d \geq 1$ and prime p, there is an $\alpha > 0$ and an $f \in \mathsf{E}^{\mathsf{NP}}$ without $\mathsf{SUM} \circ \mathsf{MODp} \circ \mathsf{AND}_d$ circuits of $2^{\alpha n}$ sparsity.*

Note the "smallest" known complexity class for a function lacking $2^{\Omega(n)}$-size circuits is $\mathsf{E}^{\Sigma_2\mathsf{P}}$ [32], and it is a longstanding open problem to reduce the complexity class for such a function, even against depth-3 AC^0 circuits.

1.1 Intuition

Here we give an overview of some of the ideas used to prove the lower bounds in this work. The lower bounds of this paper follow the high-level strategy of proving circuit lower bounds by designing circuit-analysis (satisfiability) algorithms [51, 53, 52]. However, in this work we must execute this strategy differently. All previous lower bounds proved in this framework utilize the "polynomial method" from circuit complexity in various ways (representing a circuit by a low-degree polynomial of some kind), combined with fast matrix multiplication and/or fast polynomial evaluation. These approaches do not seem to work for solving SAT on linear combinations of thresholds, low-degree polynomials, or ReLU gates. For example, we do not know how to get a sparse (probabilistic or approximate) polynomial (over any field) for computing an OR of many $\mathsf{SUM} \circ \mathsf{THR}$s, and it is likely that any reasonable approach via polynomials would fail to yield non-trivial results. However, we are able to adapt some bits of the polynomial method to the setting of low-degree polynomials (see Section 6).

Another complication is that, in the prior lower bound arguments, a nondeterministic procedure *guesses* a small circuit C of the kind one wishes to prove a lower bound against, and composes C with other Boolean circuitry to form a SAT instance. In our case, if we guess some arbitrary $\mathsf{SUM} \circ \mathcal{C}$ circuit, we first need to know if this circuit is actually computing a Boolean function; if not, then the satisfiability question itself is not well-defined, and it will not be possible to meaningfully compose such a circuit with other Boolean circuits. Thus we need a way to efficiently check whether a linear combination is Boolean-valued.

We give a generic way to "lift" non-trivial algorithms for counting SAT assignments to short products of $\mathcal{C}$ circuits to non-trivial algorithms for detecting if a given $\mathsf{SUM} \circ \mathcal{C}$ circuit is Boolean-valued and for counting SAT assignments. More precisely, we show that in order to prove lower bounds for linear combinations of $\mathcal{C}$-functions, it suffices to solve a certain sum-product task faster than exhaustive search:

Sum-Product over $\mathcal{C}$: Given k functions $f_1, \ldots, f_k$ from $\mathcal{C}$, each on Boolean variables $x_1, \ldots, x_n$, compute

$$\sum_{x \in \{0,1\}^n} \prod_{i=1}^{k} f_i(x).$$

Note the Sum-Product is computed over $\mathbb{R}$, and the task makes sense even if the functions $f_1, \ldots, f_k$ output *non-Boolean* values. Further note that if the functions $f_1, \ldots, f_k$ are Boolean-valued, then the product of k of them is simply the AND of k of them. In general, the Sum-Product problem will be NP-hard for most interesting representation classes: for example, it is already equivalent to Subset Sum when $\mathcal{C}$ is the set of *exact* threshold functions (see Section 2 for a definition). Our meta-theorem states that mild improvements over exhaustive search for Sum-Product over $\mathcal{C}$ imply strong lower bounds for $\mathsf{SUM} \circ \mathcal{C}$:

▶ **Theorem 6.** *Suppose every $C \in \mathcal{C}$ has a poly(n)-bit representation, where each C can be evaluated on a given input in poly(n) time. Assume there is an $\varepsilon > 0$ and for $k = 1, \ldots, 4$ there is an $n^{O(1)} \cdot 2^{n-\varepsilon n}$-time algorithm for computing the Sum-Product of k functions $f_1(x_1, \ldots, x_n), \ldots, f_k(x_1, \ldots, x_n)$ from $\mathcal{C}$. Then:*

1. *For every k, there is a function in NP that does not have $\mathsf{SUM} \circ \mathcal{C}$ circuits of sparsity n^k.*
2. *For every unbounded $\alpha(n)$ such that $n^{\alpha(n)}$ is time constructible, there is a function in $\mathsf{NTIME}[n^{\alpha(n)}]$ that does not have $\mathsf{SUM} \circ \mathcal{C}$ circuits of polynomial sparsity.*

Theorem 6 is used to prove lower bounds against $\mathsf{SUM} \circ \mathsf{THR}$, $\mathsf{SUM} \circ \mathsf{ReLU}$, and $\mathsf{SUM} \circ \mathsf{MOD2} \circ \mathsf{AND}_{O(1)}$, by providing non-trivial algorithms solving the Sum-Product problem for these various classes. For the E^{NP} lower bounds, we use a closure property of $\mathsf{SUM} \circ \mathsf{MOD2} \circ \mathsf{AND}_{O(1)}$ combined with standard ideas from this line of work (see Theorem 21).

Theorem 6 (and its components) can also be used to easily "lift" existing circuit lower bounds to *linear combinations* of those circuits:

▶ **Theorem 7.** *For every $d, m \geq 1$, there is a $b \geq 1$ and an $f \in \mathsf{NTIME}[n^{\log^b n}]$ that does not have $\mathsf{SUM} \circ \mathsf{AC}^0_d[m] \circ \mathsf{THR}$ circuits of n^a size, for every a.*

That is, we obtain super-polynomial sparsity lower bounds on representing nondeterministic quasi-polynomial-time functions with $\mathbb{R}$-linear combinations of $\mathsf{ACC} \circ \mathsf{THR}$ circuits (each of quasi-polynomial size). This applies the fact that we can solve the Sum-Product problem on $\mathsf{ACC} \circ \mathsf{THR}$ circuits (because we can count SAT assignments to them), with an analogous running time as the best SAT algorithm. More details on Theorem 7 can be found in Section 3.

Outline

The next section is the Preliminaries, which gives background knowledge. Section 3 proves Theorem 6. In Sections 4, 5, and 6, Sum-Product algorithms for THR, ReLU, and $\mathsf{MODp} \circ \mathsf{AND}_d$ (degree-d $\mathbb{F}_p$-polynomials) are provided which beat exhaustive search. The algorithms for THR and ReLU (Theorems 24 and 25) build upon and extend old Subset-Sum algorithms (Theorem 9). The algorithm for $\mathsf{MODp} \circ \mathsf{AND}_d$ (Theorem 26) uses tools from the polynomial method in a new way. Applying Theorem 6 to each of these algorithms, we obtain strong lower bounds for $\mathsf{SUM} \circ \mathcal{C}$ for all three classes $\mathcal{C}$.

2 Preliminaries

Let $\mathcal{C}$ be a class of functions of the form $f : \{0,1\}^n \to \mathbb{R}$. Each member $C \in \mathcal{C}$ has a number of inputs n and a size, which is the length of the representation of C in bits. For the classes THR, $\mathsf{MOD2} \circ \mathsf{AND}_{O(1)}$, and ReLU, the size $|C|$ of a representation is $\mathrm{poly}(n)$ bits, without loss of generality; see Proposition 8. (For classes such as $\mathsf{MOD2} \circ \mathsf{AND}_{\log_2(n)}$, a member of the class takes $\Omega(n^{\log n})$ bits to represent, in the worst case.) We assume that for all n, our class $\mathcal{C}$ contains the projection functions $f_i(x_1, \ldots, x_n) = x_i$ for all $i = 1, \ldots, n$. We also assume that $\mathcal{C}$ is *evaluatable*, meaning that there is a universal $k \geq 1$ such that every $C \in \mathcal{C}$ can be evaluated on a given input in $O(|C|^k)$ time. All classes we consider have this property.

As is standard, we let ANY_c denote the class of Boolean functions with c inputs (the class contains "any" such function).

An arbitrary $\mathsf{SUM} \circ \mathcal{C}$ circuit C over n variables represents some function $f : \{0,1\}^n \to \mathbb{R}$. We say that C is *Boolean-valued* if for all $x \in \{0,1\}^n$, the output of C on x is in $\{0,1\}$. The following proposition is useful to keep in mind, as it shows that every sparse linear combination of Boolean functions implementing another Boolean function has an equivalent linear combination with "reasonable" coefficients.

▶ **Proposition 8.** *Let $\mathcal{C}$ be a class of functions with co-domain $\{0,1\}$, and let C be a $\mathsf{SUM} \circ \mathcal{C}$ circuit of sparsity s that is Boolean-valued. There is an equivalent $\mathsf{SUM} \circ \mathcal{C}$ circuit C' such that every weight in the linear combination of C' has the form j/k, where both j and k are integers in $[-s^{s/2}, s^{s/2}]$.*

Proof. (See also [34, 4].) Let C be a linear combination of s functions from $\mathcal{C}$. WLOG, the set of s Boolean functions from $\mathcal{C}$ is a linearly independent set (otherwise, we could obtain a smaller linear combination representing the same function). The problem of finding coefficients for the Boolean-valued C is equivalent to solving a certain linear system $Ax = b$ in s unknowns over the rationals, where $b \in \{0,1\}^{2^n}$ and $A \in \{0,1\}^{s \times 2^n}$. Take a linearly independent subsystem of s of these 2^n equations. Since the determinant of any $s \times s$ Boolean matrix is in $[-s^{s/2}, s^{s/2}]$ [21], the result follows from Cramer's rule. ◀

The relevant theorem for sums of ReLU gates is more involved, but Maass [31] shows how the weights for a circuit of size s need only $\mathrm{poly}(s, n)$ bits of precision. Such "analog-to-digital" results are crucial for our work, as in our lower bound proofs we will need a discrete nondeterministic algorithm to *guess* a $\mathsf{SUM} \circ \mathcal{C}$ circuit and check various properties of it.

Useful Results For Thresholds

We draw from several algorithms and representation theorems from past work. For $\mathsf{SUM} \circ \mathsf{THR}$, we eventually appeal to a classic result from exact algorithms:

▶ **Theorem 9** (Horowitz and Sahni [26])**.** *The number of Subset Sum solutions to any arbitrary instance of n items with integer weights of magnitude $[-2^W, 2^W]$ can be computed in $2^{n/2} \cdot \mathrm{poly}(W)$ time.*

Theorem 9 is usually stated in terms of *finding* a subset sum solution, but the algorithm can be easily adapted to count solutions as well.

A Boolean function f is called an *exact threshold function* if there are real-valued $\alpha_1, \ldots, \alpha_n$ and t such that for all x,

$$f(x) = 1 \iff \sum_i \alpha_i x_i = t.$$

Let ETHR be the class of exact threshold functions. For our SUM ∘ THR circuit results, the following transformation is extremely useful:

▶ **Theorem 10** (Hansen and Podolskii [23]). *Every linear threshold function in n variables can be represented as an linear combination of poly(n) exact threshold functions, each with coefficient* 1*.*

It follows that every SUM ∘ THR of sparsity s has an equivalent SUM ∘ ETHR of sparsity poly(s). The idea is that a THR function defines a set of points in the Boolean hypercube lying on one side of a given hyperplane; we can "cover" all the points lying on one side by a *disjoint* sum of poly(n) hyperplanes, which function as ETHR gates. Thus each coefficient in the linear combination is simply 1.

Another useful property of ETHR gates is that they are closed under AND:

▶ **Theorem 11** (Hansen and Podolskii [23]). *Every conjunction of t exact threshold functions in n variables with integer weights in $[-W, W]$ can be converted in poly(t, n) time to an equivalent* single *exact threshold gate, with weights in $[-(nW)^{\Theta(t)}, (nW)^{\Theta(t)}]$.*

The idea is simple: if we multiply the ith exact threshold gate's linear form by the factor $(nW)^i$, no linear form will "interfere" with the other sums, and we can determine if all of them are satisfied simultaneously with one exact threshold.

Useful Results for Finite Field Polynomials

Two tools from the literature will be helpful for our results on linear combinations of polynomials. The first is *modulus-amplifying polynomials*, which have been used in Toda's Theorem [46], representations of ACC and ACC-SAT algorithms [6, 53], algorithms for All-Pairs Shortest Paths [12], and algorithms for solving polynomial systems [29]:

▶ **Lemma 12** (Beigel and Tarui [6]). *For all $\ell \in \mathbb{Z}^+$, the degree-$(2\ell - 1)$ polynomial (over $\mathbb{Z}$)*

$$P_\ell(y) = 1 - (1-y)^\ell \sum_{j=0}^{\ell-1} \binom{\ell + j - 1}{j} y^j$$

has the property for all integers $m \geq 2$,

- *if $y = 0 \bmod m$ then $P_\ell(y) = 0 \bmod m^\ell$,*
- *if $y = 1 \bmod m$ then $P_\ell(y) = 1 \bmod m^\ell$.*

Furthermore, each coefficient in F_ℓ has magnitude at most $2^{O(\ell)}$.

Recall that a multivariate polynomial is *multilinear* if it contains no powers larger than one. The second tool is a classic result on rapidly evaluating a multilinear polynomial on all points in the Boolean hypercube.

▶ **Theorem 13** (cf. [8], Section 2.2). *Given the 2^n-coefficient vector of a multilinear polynomial $p \in \mathbb{Z}[x_1, \ldots, x_n]$ where each coefficient is in $[-W, W]$, the value of p on all points in $\{0,1\}^n$ can be computed in $2^n \cdot$ poly$(n, \log W)$ time.*

The algorithm of Theorem 13 can be obtained by divide-and-conquer (as described in [50]) or by dynamic programming (as in [8], Section 2.2).

Connections Between Nondeterministic Circuit UNSAT Algorithms and Circuit Lower Bounds

We also appeal to several known connections between circuit UNSAT algorithms that beat exhaustive search and circuit lower bounds against nondeterministic time classes, which build on prior work [51, 27, 41, 7].

▶ **Theorem 14** ([35]). *If there is an $\varepsilon > 0$ such that Circuit Unsatisfiability for (fan-in 2) circuits with n inputs and $2^{\varepsilon n}$ size is solvable in $O(2^{n-\varepsilon n})$ nondeterministic time, then for every k there is a function in* NP *that does not have n^k-size (fan-in 2) circuits.*

▶ **Theorem 15** (Corollary 12 in Tell [45], following [35]). *If there is a $\delta > 0$ and $c \geq 1$ such that Circuit Unsatisfiability for (fan-in 2) circuits with n variables and m gates is solvable in $O(2^{n(1-\delta)} \cdot m^c)$ nondeterministic time, then for every unbounded $\alpha(n)$ such that $n^{\alpha(n)}$ is time-constructible, there is a function in* $\mathsf{NTIME}[n^{\alpha(n)}]$ *that is not in* $\mathsf{P}/poly$.

▶ **Theorem 16** ([35]). *If there is an $\varepsilon > 0$ such that Circuit Unsatisfiability for (fan-in 2) circuits with n inputs and 2^{n^ε} size is solvable in $O(2^{n-n^\varepsilon})$ nondeterministic time, then for every k there is a function in* $\mathsf{NTIME}[n^{poly(\log n)}]$ *that does not have $n^{\log^k n}$-size (fan-in 2) circuits.*

In fact, all of these algorithms-to-lower-bounds connections still hold when we replace Circuit Unsatisfiability with the promise problem of distinguishing unsatisfiable circuits from circuits with 2^{n-1} satisfying assignments.

The Power of Linear Combinations of Low-Degree Polynomials

We note that classical work suggests that $\mathbb{R}$-linear combinations of higher-degree $\mathbb{F}_2$-polynomials can be quite powerful. For example, applying Valiant's depth reduction [47] and using the representation of the AND function in the Fourier basis, it is easy to show that every $O(n)$-size $O(\log n)$-depth circuit can be represented by a linear combination of $2^{O(n/\log\log n)}$ $\mathbb{F}_2$-polynomials of degree $O(n^\varepsilon)$, for any desired $\varepsilon > 0$. Moreover, one can represent any $O(n)$-size "Valiant series-parallel" circuit (see [11]) by a linear combination of $2^{\varepsilon n}$ $\mathbb{F}_2$-polynomials of degree $2^{2^{O(1/\varepsilon)}}$. Hence there is a natural barrier to proving exponential-sparsity lower bounds for linear combinations of "somewhat-low" degree polynomials.

3 Meta-Theorem for Lower Bounds on Linear Combinations of Simple Functions

In this section, we prove our generic theorem which is applied in subsequent sections to prove lower bounds against linear combinations of threshold functions, ReLU gates, and constant-degree polynomials. Recall (from the Introduction) the Sum-Product problem:

> **Sum-Product over $\mathcal{C}$:** Given k functions $f_1, \ldots, f_k$ from $\mathcal{C}$, each on Boolean variables $x_1, \ldots, x_n$, compute
>
> $$\sum_{x \in \{0,1\}^n} \prod_{i=1}^{k} f_i(x).$$

▶ **Reminder of Theorem 6.** *Suppose every $C \in \mathcal{C}$ has a $poly(n)$-bit representation, where each C can be evaluated on a given input in $poly(n)$ time. Assume there is an $\varepsilon > 0$ and for $k = 1, \ldots, 4$ there is an $n^{O(1)} \cdot 2^{n-\varepsilon n}$-time algorithm for computing the Sum-Product of k functions $f_1(x_1, \ldots, x_n), \ldots, f_k(x_1, \ldots, x_n)$ from $\mathcal{C}$. Then:*

1. *For every c, there is a function in* NP *that does not have* $\mathsf{SUM} \circ \mathcal{C}$ *circuits of sparsity* n^c.
2. *For every unbounded* $\alpha(n)$ *such that* $n^{\alpha(n)}$ *is time constructible, there is a function in* $\mathsf{NTIME}[n^{\alpha(n)}]$ *that does not have* $\mathsf{SUM} \circ \mathcal{C}$ *circuits of polynomial sparsity.*

The remainder of this section will prove Theorem 6, and an extension to E^{NP} in some cases. We are able to use much of the earlier arguments [51, 53, 35] as black boxes. However we need several modifications.

The first new component needed is a method for checking that a given linear combination of $\mathcal{C}$ circuits actually encodes a Boolean function (i.e. is Boolean-valued on all Boolean inputs). This is provided by the following theorem:

▶ **Theorem 17.** *Assume there is an $\varepsilon > 0$ and for $k = 1, \ldots, 4$ there is an $n^{O(1)} \cdot 2^{n-\varepsilon n}$-time algorithm for computing the Sum-Product of k functions $f_1(x_1, \ldots, x_n), \ldots, f_k(x_1, \ldots, x_n)$ from $\mathcal{C}$.*
Then there is a $2^{n-\varepsilon n} \cdot poly(n, s)$-time algorithm that, given $f(x_1, \ldots, x_n)$ which is an arbitrary linear combination of s functions from $\mathcal{C}$, determines whether or not $f(a) \in \{0, 1\}$ for all $a \in \{0, 1\}^n$.

Proof. Suppose we are given $f = \sum_{i=1}^{s} \alpha_i c_i$, where $\alpha_i \in \mathbb{R}$ and $c_i \in \mathcal{C}$ each have n inputs. Consider the polynomial

$$h(x) := f(x)^2 \cdot (1 - f(x))^2 = f(x)^2 - 2f(x)^3 + f(x)^4.$$

Observe that:

- If $f(a) \in \{0, 1\}$ for all $a \in \{0, 1\}^n$, then $h(a) = 0$ for all a.
- $f(b) \notin \{0, 1\}$ implies $h(b) > 0$.
- For all $a \in \{0, 1\}^n$, $h(a) \geq 0$.

Therefore $\sum_{a \in \{0,1\}^n} h(a) = 0$ if and only if $f(a) \in \{0, 1\}$ for all $a \in \{0, 1\}^n$. By applying the distributive law to each of $f(x)^2$, $f(x)^3$, $f(x)^4$, and exchanging the order of summation, we have

$$\begin{aligned}\sum_{a \in \{0,1\}^n} h(a) = &\sum_{i_1, i_2} \beta_{i_1, i_2} \left(\sum_{a \in \{0,1\}^n} f_{i_1}(x) \cdot f_{i_2}(x) \right) \\ &+ \sum_{i_1, i_2, i_3} \gamma_{i_1, i_2, i_3} \left(\sum_{a \in \{0,1\}^n} f_{i_1}(x) \cdot f_{i_2}(x) \cdot f_{i_3}(x) \right) \\ &+ \sum_{i_1, i_2, i_3, i_4} \delta_{i_1, i_2, i_3, i_4} \left(\sum_{a \in \{0,1\}^n} f_{i_1}(x) \cdot f_{i_2}(x) \cdot f_{i_3}(x) \cdot f_{i_4}(x) \right)\end{aligned}$$

for $\beta_{i_1, i_2} = \alpha_{i_1} \cdot \alpha_{i_2}$, $\gamma_{i_1, i_2, i_3} = -2\alpha_{i_1} \cdot \alpha_{i_2} \cdot \alpha_{i_3}$, $\delta_{i_1, i_2, i_3, i_4} = \alpha_{i_1} \cdot \alpha_{i_2} \cdot \alpha_{i_3} \cdot \alpha_{i_4}$.

Observe that each sum over $a \in \{0, 1\}^n$ on the RHS is precisely a Sum-Product task over $\mathcal{C}$, with products ranging from $k = 2$ to $k = 4$. Therefore we can check that the sum $\sum_{a \in \{0,1\}^n} h(a)$ is zero with $O(s^4)$ calls to Sum-Product over $\mathcal{C}$. By assumption, this can be done in $O(2^{n-\varepsilon n} \cdot \text{poly}(n, s))$ time. ◀

The second crucial component yields the ability to solve Circuit Unsatisfiability efficiently with nondeterminism, under the hypotheses (in fact, weaker hypotheses). This is provided by the following lemma, which is similar to (but more complicated than) Lemma 3.1 in [53]:

▶ **Lemma 18.** *Assume:*

- *There is an $\varepsilon > 0$ and for $k = 1, \ldots, 4$ there is an $n^{O(1)} \cdot 2^{n-\varepsilon n}$-time algorithm for computing the Sum-Product of k functions from $\mathcal{C}$.*
- *The Circuit Evaluation problem has $\mathsf{SUM} \circ \mathcal{C}$ circuits of sparsity n^c, for some $c > 0$.*

Then there is a nondeterministic $2^{n-\varepsilon n} \cdot poly(n, s)$-time algorithm for Circuit Unsatisfiability, on arbitrary fan-in-2 circuits with n inputs and s gates.

Proof. Suppose we are given a circuit C with n inputs and s gates of fan-in 2, and wish to nondeterministically prove it is unsatisfiable. Let us index the gates in topological order, so that gates $1, \ldots, n$ are the input gates, and the s-th gate is the output gate.

Our nondeterministic algorithm begins by guessing a $\mathsf{SUM} \circ \mathcal{C}$ circuit $EVAL$ with $n + O(\log s)$ inputs and sparsity at most $(n+s)^{c+1}$, which is intended to encode the Circuit Evaluation function:

$$EVAL(C, x, i) := \text{Evaluate } C \text{ on } x, \text{ and output the value of the } i\text{-th gate of } C.$$

(Note i is encoded as an $O(\log s)$-bit string.) Let

$$D(x, i) := EVAL(C, x, i),$$

i.e., we think of C as hard-coded in the function, to simplify the notation. Applying Theorem 17, we can check that D encodes a Boolean function in $2^{n-\varepsilon n} \cdot \text{poly}(s, n)$ time.

Next, we check that $D(a, s) = 0$ for all $a \in \{0,1\}^n$; in other words, D claims that C outputs 0 on every input. Suppose D has the form

$$D(x, i) = \sum_{j=1}^{(n+s)^{c+1}} \alpha_j \cdot c_j(x, i),$$

for some $\alpha_j \in \mathbb{R}$ and $c_j \in \mathcal{C}$. Since D has already been determined to be Boolean, it suffices to compute $\sum_{a \in \{0,1\}^n} D(a, s)$ to know whether or not $D(x, s) = 0$ for all a. By exchanging the order of summation,

$$\begin{aligned} \sum_{a \in \{0,1\}^n} D(a, s) &= \sum_{a \in \{0,1\}^n} \left(\sum_j \alpha_j \cdot c_j(a, i) \right) \\ &= \sum_j \alpha_j \cdot \left(\sum_{a \in \{0,1\}^n} c_j(a, i) \right). \end{aligned}$$

Therefore we only need to make $(n+s)^{c+1}$ calls to Sum-Product over $\mathcal{C}$ (with $k = 1$) to determine that $D(x, s) = 0$ for all $a \in \{0,1\}^n$. This can be done in $2^{n-\varepsilon n} \cdot \text{poly}(n, s)$ time, by assumption.

Next, we have to check that for every gate $i = 1, \ldots, s$, and every $a \in \{0,1\}^n$, $D(a, i)$ correctly reports the output of the i-th gate when C evaluates a. To check the input gates, we need to check that $D(x, i) = x_i$ for all $i = 1, \ldots, n$; we can do this by checking that

$$\sum_{a \in \{0,1\}^n} (D(x, i) - x_i)^2 = 0,$$

which (by distributivity and re-arranging the order of summation, as in the proof of Theorem 17) can be computed with $O((n+s)^{2(c+1)})$ calls to Sum-Product over $\mathcal{C}$ (with $k = 2$) in $2^{n-\varepsilon n} \cdot \text{poly}(n, s)$ time.

For all gates i other than the input gates, the ith-gate takes inputs from previous gates indexed by some $i_1 < i$ and $i_2 < i$, and computes a function of their two outputs. To check the consistency of gate i, we can form a degree-3 polynomial $p_i(A, B, C)$ which outputs 0-1 values on all $A, B, C \in \{0,1\}$, such that $p_i(A, B, C) = 0$ if and only if A is the output of gate i, given that B is the output of gate i_1 and C is the output of gate i_2.

Since D is Boolean-valued, we have reduced our problem to determining that

$$\sum_{a \in \{0,1\}^n} p(D(a,i), D(a,i_1), D(a,i_2)) = 0,$$

for each gate $i = n+1, \ldots, s$, and each gate i's corresponding input gates i_1 and i_2. Applying the distributive law to the LHS and exchanging the order of summation (as before), this results in $O((n+s)^{3(c+1)})$ Sum-Product-over-$\mathcal{C}$ computations with up to $k = 3$ products, computable in $2^{n-\varepsilon n} \cdot \text{poly}(n, s)$ time.

Our nondeterministic algorithm determines that the input circuit C is unsatisfiable if and only if all of the above checks pass. If C is satisfiable, then every possible D guessed will fail some check. If C is unsatisfiable, then under the hypotheses of the theorem, a $\mathsf{SUM} \circ \mathcal{C}$ circuit D simulating every gate of C always exists. By guessing this D, and running the assumed Sum-Product algorithm, our nondeterministic algorithm accepts. ◀

After the above preparation, we turn back to the proof of Theorem 6. At this point, it is simply a matter of applying the above Lemma 18 with the known algorithms-to-lower-bound connections:

Proof of Theorem 6. Suppose every $C \in \mathcal{C}$ has a poly(n)-bit representation, where each C can be evaluated on a given input in poly(n) time. Recall the hypothesis of the theorem is:

> (A) There is an $\varepsilon > 0$ and for $k = 1, \ldots, 4$ there is an $n^{O(1)} \cdot 2^{n-\varepsilon n}$-time algorithm for computing the Sum-Product of k functions $f_1(x_1, \ldots, x_n), \ldots, f_k(x_1, \ldots, x_n)$ from $\mathcal{C}$.

Furthermore, recall that Lemma 18 states:

> Assuming (A) and assuming Circuit Evaluation has $\mathsf{SUM} \circ \mathcal{C}$ circuits of sparsity n^k for some k, there is a nondeterministic $2^{n-\varepsilon n} \cdot \text{poly}(n, s)$-time algorithm for Circuit Unsatisfiability, on arbitrary fan-in-2 circuits with n inputs and s gates.

We can then prove the lower bounds of the theorem readily, as follows.

(1) Assume every function in NP has $\mathsf{SUM} \circ \mathcal{C}$ circuits of n^k sparsity circuits, for some fixed k. Then both hypotheses of Lemma 18 are satisfied (note Circuit Evaluation is in P), and the conclusion implies that there is an $\varepsilon > 0$ such that Circuit Unsatisfiability for (fan-in 2) circuits with n inputs and $2^{\varepsilon n}$ size is solvable in $O(2^{n-\varepsilon n})$ nondeterministic time. Therefore by Theorem 14, for every k there is a function in NP that *does not* have n^k-size (fan-in 2) circuits. This is a contradiction because $\mathsf{SUM} \circ \mathcal{C}$ circuits of n^k sparsity can be simulated with n^{ck}-size fan-in-2 circuits, for some universal c.

(2) The same argument as in (1) and (2) (but with Theorem 15 applied) shows that for every unbounded $\alpha(n)$ such that $n^{\alpha(n)}$ is time-constructible, there is a function in $\mathsf{NTIME}[n^{\alpha(n)}]$ that does not have $\mathsf{SUM} \circ \mathcal{C}$ circuits of polynomial sparsity.

This completes the proof. ◀

A Note on Lower Bounds for Linear Combinations of ACC Circuits

There are other new lower bound consequences of the arguments in Theorem 6 that we will not study in detail here, because they follow easily from combining known results. Here is an example:

▶ **Reminder of Theorem 7.** *For every $d, m \geq 1$, there is a $b \geq 1$ and an $f \in \mathsf{NTIME}[n^{\log^b n}]$ that does not have $\mathsf{SUM} \circ \mathsf{AC}^0_d[m] \circ \mathsf{THR}$ circuits of n^a size, for every a.*

This lower bound can be obtained as follows. First, the argument of Lemma 18 also shows:

▶ **Theorem 19.** *Assume*

- *There is an $\varepsilon > 0$ and for $k = 1, \ldots, 4$ there is an $n^{O(1)} \cdot 2^{n-n^\varepsilon}$-time algorithm for computing the Sum-Product of k functions from $\mathcal{C}$.*
- *The Circuit Evaluation problem has $\mathsf{SUM} \circ \mathcal{C}$ circuits of sparsity n^a, for some $a > 0$.*

Then there is a nondeterministic $2^{n-n^\varepsilon} \cdot poly(n, s)$-time algorithm for Circuit Unsatisfiability, on arbitrary fan-in-2 circuits with n inputs and s gates.

Now we combine this theorem with the following two facts:

1. For every depth d and integer $m \geq 2$, there is an $\varepsilon > 0$ such that the Sum-Product of $O(1)$ $\mathsf{AC}^0_d[m] \circ \mathsf{THR}$ circuits of 2^{n^ε} size can be computed in 2^{n-n^ε} time. This simply applies the algorithm for counting satisfying assignments of $\mathsf{AC}^0_d[m] \circ \mathsf{THR}$ circuits ([52]).
2. If for some $\alpha > 0$ there is a nondeterministic 2^{n-n^α}-time Circuit Unsatisfiability algorithm for 2^{n^α}-size circuits, then for every $a \geq 1$, there is a $b \geq 1$ such that $\mathsf{NTIME}[n^{\log^b n}]$ does not have $n^{\log^a n}$-size circuits (this is a theorem of Murray and Williams [35]).

Theorem 7 is immediate: Assuming $\mathsf{NTIME}[n^{\log^b n}]$ has $\mathsf{SUM} \circ \mathsf{AC}^0_d[m] \circ \mathsf{THR}$ circuits of n^a size for some $a \geq 1$, both hypotheses of Theorem 19 are satisfied for $\mathcal{C} = \mathsf{AC}^0_d[m] \circ \mathsf{THR}$, and the conclusion of Theorem 19 combined with item 2 above yields a contradiction.

3.1 Lower Bounds for Exponential Time With an NP Oracle

For classes $\mathcal{C}$ with a natural closure property, the lower bounds can be extended to $2^{\Omega(n)}$ sparsity for a function in E^{NP}. Recall ANY_c denotes the class of Boolean functions with c inputs (the class contains "any" such function).

For an integer $c \geq 1$, we say that $\mathcal{C}$ is *efficiently closed under* NC^0_c if there is a polynomial-time algorithm A such that, given any circuit C of the form $\mathcal{C} \circ \mathsf{ANY}_c$, algorithm A outputs an equivalent circuit D from $\mathcal{C}$ (which is only polynomially larger). We note this property is true of $O(1)$-degree polynomials:

▶ **Proposition 20.** *For every integer $m \geq 2$ and $c \geq 1$, the class $\mathcal{C} = \bigcup_{d \geq 1} \mathsf{MODm} \circ \mathsf{AND}_d$ is efficiently closed under NC^0_c.*

Proof. Every $\mathsf{MODm} \circ \mathsf{AND}_d \circ \mathsf{ANY}_c$ circuit can be represented by an $\mathsf{MODm} \circ \mathsf{AND}_{dc}$ circuit. In particular, every Boolean function on c inputs has an exact representation as a sum (modulo m) of ANDs of fan-in c; composing such a sum with a $\mathsf{MODm} \circ \mathsf{AND}$ circuit and applying the distributive law yields the result. ◀

▶ **Theorem 21.** *There is a universal $c \geq 1$ satisfying the following. Suppose $\mathcal{C}$ is efficiently closed under NC^0_c, and suppose every $C \in \mathcal{C}$ has a poly(n)-bit representation, where each C can be evaluated on a given input in poly(n) time.*
Assume there is an $\varepsilon > 0$ and for $k = 1, \ldots, 4$ there is an $n^{O(1)} \cdot 2^{n-\varepsilon n}$-time algorithm for

computing the Sum-Product of k functions $f_1(x_1,\ldots,x_n),\ldots,f_k(x_1,\ldots,x_n)$ from $\mathcal{C}$. Then there is a function in E^{NP} that does not have $\mathsf{SUM}\circ\mathcal{C}$ circuits of sparsity $2^{\alpha n}$, for some $\alpha > 0$.

The remainder of this section sketches the proof of Theorem 21; we give only a sketch, as the argument closely resembles others [53, 27]).

Let $\varepsilon \in (0,1)$. Assume $\mathcal{C}$ is efficiently closed under NC^0_c. Furthermore:

(A) There is an $\varepsilon > 0$ and an $O(2^{n-\varepsilon n})$-time algorithm for computing the Sum-Product of k functions from $\mathcal{C}$, *and*

(B) For all functions $f \in \mathsf{TIME}[2^{O(n)}]^{\mathsf{NP}}$ and all $\alpha > 0$, f has $\mathsf{SUM}\circ\mathcal{C}$ circuits of sparsity $2^{\alpha n}$.

We wish to establish a contradiction. In particular, we will show that assumptions (A) and (B) together imply that every problem in $\mathsf{NTIME}[2^n]$ can be simulated by a nondeterministic $o(2^n)$-time algorithm, contradicting the (strong) nondeterministic time hierarchy theorem [42, 56].

Let $L \in \mathsf{NTIME}[2^n]$. On a given input x, our nondeterministic $o(2^n)$-time algorithm for L has two parts:

(i) It *guesses a witness for x* of $o(2^n)$ size.

(ii) It *verifies that witness for x* in $o(2^n)$ time.

To handle (i), we use assumption (B) to show that one can nondeterministically *guess* a $2^{\alpha n}\cdot\mathrm{poly}(n)$-size $\mathsf{SUM}\circ\mathcal{C}$ circuit that encodes a witness for x, applying a simple "easy witness" lemma from [51]:

▶ **Lemma 22** (Lemma 3.2 in [51]). *Let $\mathcal{D}$ be any class of circuits. If E^{NP} has circuits of size $S(n)$ from class $\mathcal{D}$, then for every $L \in \mathsf{NTIME}[2^n]$ and every verifier V for L, and every $x \in L$ of length $n = |x|$, there is a y of length $O(2^n)$ such that $V(x,y)$ accepts and the $\mathcal{D}$-circuit complexity of y (construed as a function $f : \{0,1\}^{n+O(1)} \to \{0,1\}$) is at most $S(n)$.*

In other words, assumption (B) implies that every yes-instance of L has $S(n)$-size "witness circuits": a witness of length $O(2^n)$ that can be represented as an $S(n)$-size $\mathsf{SUM}\circ\mathcal{C}$ Boolean-valued circuit. Furthermore, this holds for every verifier for L.

To handle (ii), we choose an appropriate verifier, so that verifying witnesses becomes equivalent to a simple Sum-Product call. In particular we use the following extremely "local" reduction from $L \in \mathsf{NTIME}[2^n]$ to 3SAT instances of $2^n\cdot\mathrm{poly}(n)$ length:

▶ **Lemma 23** ([27]). *Every $L \in \mathsf{NTIME}[2^n]$ can be reduced to 3SAT instances of $O(2^n\cdot n^4)$ size. Moreover, there is an algorithm that, given an instance x of L and an integer $i \in [O(2^n\cdot n^4)]$ in binary, **reads only** $O(1)$ **bits of** x and outputs the i-th clause of the resulting 3SAT formula, in $O(n^4)$ time.*

Since in Lemma 23 each bit of the output is a function of some $c \le O(1)$ inputs, each bit of the output is a member of ANY_c. So for every instance x of length n for the language L, we can produce (in deterministic $\mathrm{poly}(n)$ time) a circuit D_x which is an ordered collection of $O(n)$ functions from ANY_c. The circuit D_x takes $n + O(\log n)$ binary inputs, construes that input as an integer i, and outputs the i-th clause of a formula F_x which is satisfiable if and only if $x \in L$.

Our nondeterministic algorithm for L guesses a $2^{O(\alpha n)}$-sparse $\mathsf{SUM}\circ\mathcal{C}$ circuit C_x that takes $n + O(\log n)$ inputs and is meant to encode a satisfying assignment for the formula F_x. We can check C_x is Boolean-valued on all $2^n\cdot\mathrm{poly}(n)$ inputs in $2^{n-\varepsilon n/2}$ time, by applying Theorem 17 and letting $\alpha > 0$ be sufficiently small.

Composing C_x with the $O(n)$ polynomials forming D_x, we obtain a $2^{O(\alpha n)}$-sparse $\mathsf{SUM} \circ \mathcal{C} \circ \mathsf{ANY}_c$ circuit E with $n + O(\log n)$ inputs (composed of three copies of C_x, and $O(n)$ copies of D_x) such that

E is unsatisfiable if and only if C_x encodes a satisfying assignment for F_x.

(We leave out the details, as they are provided in multiple other papers [51, 53].) To complete the $o(2^n)$-time algorithm for L, it suffices to check unsatisfiability of the resulting $2^{O(\alpha n)}$-size circuit E in $o(2^n)$ nondeterministic time. This would yield the desired contradiction.

Such a nondeterministic UNSAT algorithm is provided by first converting E into a $\mathsf{SUM} \circ \mathcal{C}$ circuit in $2^{O(\alpha n)}$ time (using the fact that $\mathcal{C}$ is efficiently closed under NC^0). This yields a sum of $2^{O(\alpha n)}$ $\mathcal{C}$-circuits. Analogously to the proof of Lemma 18, checking the unsatisfiability of such an E can be reduced to $2^{O(\alpha n)}$ calls to Sum-Product of $\mathcal{C}$, by applying distributivity. Applying the Sum-Product algorithm of assumption (A) that runs in $O(2^{n-\varepsilon n})$ time, and setting $\alpha > 0$ to be sufficiently small, the running time is $o(2^n)$.

This completes the proof of Theorem 21.

4 Sparse Combinations of Threshold Functions

We now turn to proving $\mathsf{SUM} \circ \mathsf{THR}$ lower bounds. Due to Lemma 6, it suffices to give a $2^{n-\varepsilon n}$-time algorithm for the Sum-Product Problem over THR:

Sum-Product over THR: Given k linear threshold functions $f_1, \ldots, f_k$, each on Boolean variables $x_1, \ldots, x_n$, compute

$$\sum_{x \in \{0,1\}^n} \prod_{i=1}^{k} f_i(x).$$

Putting together various pieces (described in the Preliminaries), there is a substantially faster-than-2^n time algorithm:

▶ **Theorem 24.** *The Sum-Product of k linear threshold functions on n variables (with weights in $[-n^n, n^n]$) can be computed in $2^{n/2} \cdot n^{O(k)}$ time.*

Note that having weights in $[-n^n, n^n]$ is without loss of generality (in our lower bound proofs, our nondeterministic algorithm can always guess an equivalent circuit with such weights, as described by Proposition 8).

Proof. Let $f_1, \ldots, f_k$ be n-variable threshold functions. Applying Theorem 10, we can write each f_i as a sum of $t = \mathrm{poly}(n)$ *exact* threshold functions:

$$f_i(x) = \sum_{i=1}^{t} g_i(x),$$

where each $g_i(x)$ is defined by some weights $w_{i,1}, \ldots, w_{i,n} \in \mathbb{R}$ and a threshold value $t \in \mathbb{R}$. Therefore we can write the product $f_1 \cdots f_k$ as

$$\prod_{i=1}^{k} f_i = \sum_{(i_1,\ldots,i_k) \in [t]^k} g_{i_1} \cdots g_{i_k}.$$

Each term $g_{i_1} \cdots g_{i_k}$ is a conjunction of k exact thresholds. Applying Theorem 11, each such term can be replaced with a single exact threshold gate, with weights of magnitude $n^{O(kn)}$, i.e., each weight is representable with $O(kn \log n)$ bits. Thus

$$\prod_{i=1}^{k} f_i = \sum_{(i_1,\ldots,i_k)\in[t]^k} h_{i_1,\ldots,i_k}$$

for some exact threshold gates $h_{i_1,\ldots,i_k}$. The desired sum can therefore be written as

$$\begin{aligned}\sum_{a\in\{0,1\}^n}\prod_{i=1}^{k} f_i(a) &= \sum_{a\in\{0,1\}^n}\sum_{(i_1,\ldots,i_k)\in[t]^k} h_{i_1,\ldots,i_k}(a)\\ &= \sum_{(i_1,\ldots,i_k)\in[t]^k}\left(\sum_{a\in\{0,1\}^n} h_{i_1,\ldots,i_k}(a)\right).\end{aligned}$$

Now observe that each sum $\sum_{a\in\{0,1\}^n} h_{i_1,\ldots,i_k}(a)$ on the RHS is equivalent to an instance of #Subset Sum. In particular, each such sum is counting the number of subsets of a given set of n weights in $[-n^{\Omega(kn)}, n^{O(kn)}]$ which sum to zero. By Theorem 9, this can be computed in $\mathrm{poly}(k,n)\cdot 2^{n/2}$ time. Since there are $n^{O(k)}$ such sums to compute in the outer sum, the total running time is $n^{O(k)}\cdot 2^{n/2}$. ◀

The following are immediate from Theorem 6:

▶ **Reminder of Theorem 1.** *For all k, there is an $f_k \in \mathsf{NP}$ without $\mathsf{SUM}\circ\mathsf{THR}$ circuits of n^k sparsity. Furthermore, for every unbounded $\alpha(n)$ such that $n^{\alpha(n)}$ is time constructible, there is a function in $\mathsf{NTIME}[n^{\alpha(n)}]$ that does not have $\mathsf{SUM}\circ\mathsf{THR}$ circuits of polynomial sparsity.*

5 Sparse Combinations of ReLU Gates

Recall that a function $f:\{0,1\}^n\to\mathbb{R}$ from the class ReLU is defined with respect to a weight vector $w\in\mathbb{R}^n$ and a scalar $a\in R$, such that for all $a\in\{0,1\}^n$,

$$f(x) = \max\{0, \langle w, x\rangle + a\}.$$

To prove $\mathsf{SUM}\circ\mathsf{ReLU}$ lower bounds, we give a $2^{n-\varepsilon n}$-time algorithm for the Sum-Product Problem over ReLU:

Sum-Product over ReLU: Given k ReLU functions $f_1,\ldots,f_k$, each on Boolean variables $x_1,\ldots,x_n$, compute

$$\sum_{x\in\{0,1\}^n}\prod_{i=1}^{k} f_i(x).$$

▶ **Theorem 25.** *The Sum-Product of k ReLU functions on n variables (with weights in $[-W,W]$) can be computed in $2^{n/2}\cdot n^{O(k)}\cdot \mathrm{poly}(k,n,\log W)$ time.*

The proof is similar in spirit to the algorithm for Sum-Product of threshold functions (Theorem 24), except that complications arise due to the real-valued outputs of ReLU functions. We end up having to solve a problem generalizing #Subset Sum, but which turns out to have a nice "split-and-list" $2^{n/2}$-time algorithm, analogously to #Subset Sum.

Proof. Let $f_1, \ldots, f_k$ be n-variable ReLU functions, defined by weight vectors $w_1, \ldots, w_k \in \mathbb{R}^n$ and scalars $a_1, \ldots, a_k \in \mathbb{R}$, respectively. Our task is to compute

$$\sum_{x \in \{0,1\}^n} \max\{0, \langle x, w_1 \rangle + a_1\} \cdots \max\{0, \langle x, w_k \rangle + a_k\}.$$

First, we note the above sum is equal to

$$\sum_{x \in \{0,1\}^n} [\langle x, w_1 \rangle \geq -a_1] \cdot (\langle x, w_1 \rangle + a_1) \cdots [\langle x, w_k \rangle \geq -a_k] \cdot (\langle x, w_k \rangle + a_k),$$

where we are using the Iverson bracket notation $[P]$ to denote a function that outputs 1 if P is true and 0 otherwise. Applying Theorem 10, each of the threshold functions $[\langle x, w_i \rangle \geq -a_i]$ can be represented as a linear combination of $t = \text{poly}(n)$ exact threshold functions. In particular there are exact thresholds $g_{i,j}$ such that the above sum equals

$$\sum_x \left(\sum_{j=1}^{t} g_{1,j}(x) \right) \cdot (\langle x, w_1 \rangle + a_1) \cdots \left(\sum_{j=1}^{t} g_{k,j}(x) \right) \cdot (\langle x, w_k \rangle + a_k).$$

Applying the distributive law, the above sum equals

$$\sum_x \sum_{j_1, \ldots, j_k \in [t]^k} g_{1,j_1}(x) \cdots g_{k,j_k}(x) \cdot (\langle x, w_1 \rangle + a_1) \cdots (\langle x, w_k \rangle + a_k).$$

Re-arranging the summation order yields

$$\sum_{j_1, \ldots, j_k \in [t]^k} \left(\sum_x g_{1,j_1}(x) \cdots g_{k,j_k}(x) \cdot (\langle x, w_1 \rangle + a_1) \cdots (\langle x, w_k \rangle + a_k) \right).$$

Applying Theorem 11, each $g_{1,j_1}(x) \cdots g_{k,j_k}(x)$ can be replaced by a single exact threshold $h_{j_1,\ldots,j_k}(x)$.

Our task has been reduced to $n^{O(k)}$ computations of the form

$$\sum_{x \in \{0,1\}^n} h_{j_1,\ldots,j_k}(x) \cdot (\langle x, w_1 \rangle + a_1) \cdots (\langle x, w_k \rangle + a_k). \tag{1}$$

Without the $(\langle x, w_1 \rangle + a_1) \cdots (\langle x, w_k \rangle + a_k)$ term, (1) would be exactly a #Subset Sum instance, as in Theorem 24. In this new situation, we need to count a "weighted" sum over the subset sum solutions, where the weights are determined by a product of k inner products of the solution vectors with some fixed vectors.

Let us now describe how to solve the generalized problem given by (1). To keep the exposition clear, we will walk through an attempted solution and fix it as it breaks.

Suppose the exact threshold function $h_{j_1,\ldots,j_k}(x)$ of (1) is defined by weights $\alpha_1, \ldots, \alpha_n \in \mathbb{R}$ and threshold value $t \in \mathbb{R}$, so that

$$h_{j_1,\ldots,j_k}(x) = 1 \iff \sum_{i=1}^{n} \alpha_i x_i = t.$$

As with the Subset Sum problem, we begin by splitting the set of variables x into two halves, $\{x_1, \ldots, x_{n/2}\}$ and $\{x_{n/2+1}, \ldots, x_n\}$ (WLOG, assume n is even). Correspondingly, we split each of the k weight vectors $w_i \in \mathbb{R}^n$ of (1) into two halves, $w_i^{(1)} \in \mathbb{R}^{n/2}$ and $w_i^{(2)} \in \mathbb{R}^{n/2}$ for the first and second halves of variables, respectively.

We list all $2^{n/2}$ partial assignments to the first half, and all $2^{n/2}$ partial assignments to the second. For each partial assignment $A = (A_1, \ldots, A_{n/2})$ to the first half of variables $\{x_1, \ldots, x_{n/2}\}$, we compute a vector v_A, as follows:

- $v_A[0] := -t + \sum_{i=1}^{n/2} \alpha_i A_i$,
- for all $j = 1, \ldots, k$, $v_A[j] := a_j + \left\langle w_j^{(1)}, (A_1, \ldots, A_{n/2}) \right\rangle$.

For each partial assignment $A' = (A_{n/2+1}, \ldots, A_n)$ from the second half, we compute a vector $w_{A'}$:

- $w_{A'}[0] := \sum_{i=n/2+1}^{n} \alpha_i A_i$,
- for all $j = 1, \ldots, k$, $w_{A'}[j] := \left\langle w_j^{(2)}, (A_{n/2+1}, \ldots, A_n) \right\rangle$.

Notice that $v_A[0] + w_{A'}[0] = 0$ if and only if $h_{j_1,\ldots,j_k}(A, A') = 1$. Thus in our sum, we only need to consider pairs of vectors v_A from the first half and vectors $w_{A'}$ from the second half such that $v_A[0] + w_{A'}[0] = 0$. Moreover, note that for all $j = 1, \ldots, k$,

$$v_A[j] + w_{A'}[j] = \langle x, w_j \rangle + a_j.$$

It follows that (1) equals

$$\sum_{(v_A, w_{A'}) \,:\, v_A[0]+w_{A'}[0]=0} (v_A[1] + w_{A'}[1]) \cdots (v_A[k] + w_{A'}[k]).$$

The Subset-Sum algorithm of Horowitz and Sahni [26] shows how to efficiently find pairs $(v_A, w_{A'})$ with $v_A[0] + w_{A'}[0] = 0$: sorting all vectors in the second half by their 0-th coordinate, for each vector v_A from the first half we can compute (in $\mathrm{poly}(n)$ time) the number of second-half vectors $w_{A'}$ satisfying $v_A[0] + w_{A'}[0] = 0$ (even if there are exponentially many such vectors). However it is unclear how to incorporate the odd-looking $(v_A[1] + w_{A'}[1]) \cdots (v_A[k] + w_{A'}[k])$ multiplicative factors into a weighted sum.

To do so, we modify the vectors v_A and w_B as follows. Consider the expansion of $\prod_{i=1}^{k}(v_A[i] + w_{A'}[i])$ into a sum of 2^k products: it can be seen as the inner product of two 2^k-dimensional vectors, where one vector's entries is a function solely of v_A and the other vector's entries is a function solely of $w_{A'}$. (Furthermore, note that the number of bits needed to describe entries in these new vectors has increased only by a multiplicative factor of k.)

Thus we can assign $(2^k + 1)$-dimensional vectors v'_A (in place of the v_A) and w'_B (in place of the w_B) such that $v'_A[0] = v_A[0]$, $w'_A[0] = w_A[0]$, and for all A, A' we have

$$(v_A[1] + w_{A'}[1]) \cdots (v_A[k] + w_{A'}[k]) = \sum_{j=1}^{2^k} v'_A[j] \cdot w'_{A'}[j].$$

Now our goal is to compute

$$\sum_{(v'_A, w'_{A'}) \,:\, v'_A[0]+w'_{A'}[0]=0} \left(\sum_{j=1}^{2^k} v'_A[j] \cdot w'_{A'}[j] \right). \tag{2}$$

We can get a more efficient algorithm for the problem defined by (2), by preprocessing the second half of vectors (i.e., the $w'_{A'}$ vectors). For each distinct value $e = w'_A[0] \in \mathbb{R}$ among the $2^{n/2}$ vectors in the second half, we make a new $(2^k + 1)$-dimensional vector W'_e where:

- $W'_e[0] = e$, and
- for all $i = 1, \ldots, 2^k$, $W'_e[i] = \sum_{w'_A \,:\, w'_A[0]=e} w'_A[i]$.

That is, the coordinates $1, \ldots, 2^k$ of W'_e are obtained by component-wise summing all vectors w'_A such that $w'_A[0] = e$. The preparation of the vectors W'_e can be done in $2^{n/2} \cdot \mathrm{poly}(k, n, \log W)$ time, by partitioning all $2^{n/2}$ vectors w'_A from the second half of variables into equivalence classes (where two vectors are equivalent if their 0-coordinates are equal), then obtaining each W'_e by summing the vectors in one equivalence class.

Finally, we can use the $W'_{A'}$ vectors to compute the sum (2) in $2^{n/2} \cdot 2^k \cdot \text{poly}(k, n, \log W)$ time. Have a running sum that is initially 0. Iterate through each vector v'_A from the first half of variables, look up the corresponding second-half vector W'_e (with $v'_A[0] = -W'_e[0]$) in $\text{poly}(k, n, \log W)$ time, and add the inner product

$$\sum_{i=1}^{2^k} v'_A[i] \cdot W'_e[i]$$

to the running sum. Because each vector $(W'_e[1], \ldots, W'_e[2^k])$ is the sum of *all* vectors $(w'_{A'}[1], \ldots, w'_{A'}[2^k])$ such that $v'_A[0] + w'_{A'}[0] = 0$, each inner product $\sum_{i=1}^{2^k} v'_A[i] \cdot W'_e[i]$ contributes

$$\sum_{w'_{A'} \,:\, v'_A[0]+w'_{A'}[0]=0} \left(\sum_{j=1}^{2^k} v'_A[j] \cdot w'_A[j] \right)$$

to the running sum. Therefore after iterating through all vectors v'_A, our running sum has computed (2) exactly, in only $2^{n/2} \cdot 2^k \cdot \text{poly}(n, \log W)$ time. ◀

From the algorithm of Theorem 25, we immediately obtain the $\mathsf{SUM} \circ \mathsf{ReLU}$ lower bounds of Theorem 2.

6 Sparse Combinations of Low-Degree Polynomials over Finite Fields

We can also prove lower bounds for linear combinations of low-degree $\mathbb{F}_p$-polynomials in n variables, for any prime p, by giving a faster Sum-Product algorithm. In this context, the Sum-Product problem becomes:

Sum-Product over $\mathsf{MOD}_p \circ \mathsf{AND}_d$: Given k polynomials $p_1, \ldots, p_k \in \mathbb{F}_p[x_1, \ldots, x_n]$, each of degree at most d, compute

$$\sum_{x \in \{0,1\}^n} \left(\prod_{i=1}^{k} p_i(x) \right),$$

where the sum over all $x \in \{0,1\}^n$ is taken over the reals (or rationals).

That is, we treat each $\prod_{i=1}^{k} p_i(x)$ as a function from $\{0,1\}^n$ to $\{0, 1, \ldots, p-1\} \subset \mathbb{Q}$, and wish to compute the sum of these integers over all $x \in \{0,1\}^n$.

In related work, Lokshtanov *et al.* [29] showed how to (deterministically) count solutions in $\mathbb{F}_p^n$ to a system of ℓ degree-d $\mathbb{F}_p$-polynomials in $p^{n+o(n)-n/O(dp^{6/7})} \cdot \text{poly}(\ell)$ time. For our Sum-Product problem, we need to compute a "weighted" sum (the terms can take on values in $\{0, \ldots, p-1\}$), and we need to count the weighted sum over only *Boolean* assignments. We can achieve this, with a comparable runtime savings involving k and p:

▶ **Theorem 26.** *The Sum-Product of k degree-d polynomials $p_1, \ldots, p_k \in \mathbb{F}_p[x_1, \ldots, x_n]$ can be computed in $p^{2k} \cdot (1.9^n + 2^{n-n/(6dp)}) \cdot \text{poly}(n)$ time.*

Proof. Let $p_1, \ldots, p_k$ be given. We wish to compute

$$\sum_{x \in \{0,1\}^n} \left(\prod_{i=1}^{k} p_i(x) \right), \tag{3}$$

where each product outputs an integer in $\{0, 1, \ldots, p-1\}$. We first convert the Sum-Product problem of (3) to an equivalent sum where each "term" in the sum is a small system of polynomial equations.

We say that a function $f : \{0,1\}^n \to \{0,1\}$ is an *exact $\mathbb{F}_p$-polynomial function* if there is a polynomial $p \in \mathbb{F}_p[x_1, \ldots, x_n]$ and $a \in \mathbb{F}_p$ such that for all $x \in \{0,1\}^n$,

$$f(x) = 1 \iff p(x) = a.$$

We use the notation $[p(x) = a]$ to denote such an exact polynomial function. Let us replace each polynomial $p_i(x)$ in the sum-product expression with an equivalent linear combination (over $\mathbb{Z}$) of exact polynomial functions. In particular, replace each $p_i(x)$ with the sum *over the integers*

$$\sum_{a \in \mathbb{F}_p} a \cdot [p_i(x) = a].$$

That is, we are replacing $p_i(a)$ with an equivalent integer-valued sum of p Boolean functions. Now the desired sum (3) has the form:

$$\sum_{x \in \{0,1\}^n} \left(\prod_{i=1}^{k} \left(\sum_{a \in \mathbb{F}_p} a \cdot [p_i(x) = a] \right) \right)$$

$$= \sum_{x \in \{0,1\}^n} \sum_{(a_1, \ldots, a_k) \in \mathbb{F}_p^k} a_1 \cdots a_k \cdot \prod_{i=1}^{k} [p_i(x) = a_i] \quad \text{(by distributivity)} \tag{4}$$

$$= \sum_{(a_1, \ldots, a_k) \in \mathbb{F}_p^k} a_1 \cdots a_k \cdot \left(\sum_{x \in \{0,1\}^n} [p_1(x) = a_1] \cdots [p_k(x) = a_k] \right). \tag{5}$$

Each inner sum in (5) counts the number of Boolean solutions to a system of polynomial equations $p_1(x) = a_1, \ldots, p_k(x) = a_k$. We can further reduce this problem to counting the number of Boolean solutions to *one* equation, by applying a simple reduction (from [54]). Namely, we have the equation

$$\sum_{x \in \{0,1\}^n} \prod_{i=1}^{k} [p_i(x) = a_i] \tag{6}$$

$$= \frac{1}{p^k} \sum_{(b_1, \ldots, b_k) \in \mathbb{F}_p^k} \sum_{x \in \{0,1\}^n} \left(\left[\sum_{j=1}^{k} b_j \cdot (p_j(x) - a_j) = 0 \right] - \left[\sum_{j=1}^{k} b_j \cdot (p_j(x) - a_j) = 1 \right] \right). \tag{7}$$

To see why (6) holds, let $x \in \{0,1\}^n$ such that $[p_1(x) = a_1] \cdots [p_k(x) = a_k] = 1$. Then for *every* $(b_1, \ldots, b_k) \in \mathbb{F}_p^k$, we have $[\sum_{j=1}^{k} b_j \cdot (p_j(x) - a_j) = 0] = 1$. So every solution x to the system of k equations is counted for p^k times in (6); since the result is divided by p^k, each solution contributes 1 to (6). On the other hand, if x is not a solution to the system, and $[p_1(x) = a_1] \cdots [p_k(x) = a_k] = 0$, then for some j, $p_j(a) - a_j \neq 0$. It follows that there are precisely p^{k-1} vectors $(b_1, \ldots, b_k) \in \mathbb{F}_p^k$ such that $[\sum_{j=1}^{k} b_j \cdot (p_j(x) - a_j) = 0] = 1$, and there are precisely p^{k-1} (other) vectors $(b'_1, \ldots, b'_k) \in \mathbb{F}_p^k$ such that $[\sum_{j=1}^{k} b'_j \cdot (p_j(x) - a_j) = 1] = 1$. These two equal counts cancel out in the sum of (6), so non-solutions to the system contribute 0 to the sum of (6).

Putting (5) and (6) together, the original Sum-Product problem (3) can now be reduced to the computation of $O(p^{2k})$ sums, each of the form

$$\sum_{x\in\{0,1\}^n} [q(x_1,\ldots,x_n)=0],$$

where q is an $\mathbb{F}_p$-polynomial of degree at most d. That is, to obtain (3), we only need to count the Boolean roots of $O(p^{2k})$ polynomials q, and take the appropriate $\mathbb{R}$-linear combination of these counts.

Let us now focus on counting roots to a single polynomial $q(x_1,\ldots,x_n)$ of degree d. Let $P_\ell(z)$ be the modulus-amplifying polynomial of degree $2\ell-1$, from Theorem 12. Let $\delta\in(0,1/2)$ be a parameter, and consider the following "reduced" polynomial in $n-\delta n$ variables, over the integers:

$$Q(x_1,\ldots,x_{n-\delta n}) := \sum_{a_1,\ldots,a_{\delta n}\in\{0,1\}} P_{\delta n}(1-q(x_1,\ldots,x_{n-\delta n},a_1,\ldots,a_{\delta n})^{p-1}).$$

Note that Q has degree less than $2dp\delta n$. Set $\delta=1/(6dp)$, and note that $2dp\delta n<(n-\delta n)/2$. Over $\mathbb{F}_p$, the polynomial $1-q(x)^{p-1}$ equals 1 mod p if x is a root of q, and is 0 mod p otherwise. Applying the modulus-amplifying properties of $P_{\delta n}$, we have:

- If x is a root of q, then $P_{\delta n}(1-q(x)^{p-1})=1 \bmod p^{\delta n}$.
- If x is not a root of q, then $P_{\delta n}(1-q(x)^{p-1})=0 \bmod p^{\delta n}$.

As the sum in Q is over only $2^{\delta n}$ such $P_\delta(\cdots)$ terms, and $p\geq 2$, we conclude that for all $b_1,\ldots,b_{n-\delta n}\in\{0,1\}$, the quantity $(Q(b_1,\ldots,b_{n-\delta n}) \bmod p^{\delta n})$ *equals* the number of $a_1,\ldots,a_{\delta n}\in\{0,1\}$ such that

$$q(b_1,\ldots,b_{n-\delta n},a_1,\ldots,a_{\delta n})=0.$$

Therefore if we evaluate the polynomial Q over all $2^{n-\delta n}$ Boolean assignments $(b_1,\ldots,b_{n-\delta n})$, compute each value separately modulo $p^{\delta n}$, then sum those values over the integers, we will obtain the number of Boolean roots of q.

Over Boolean assignments, we may assume without loss of generality that Q is multilinear (i.e. $x_i^2=x_i$ for all i). Since $2dp\delta n<(n-\delta n)/2$, standard properties of binomial coefficients imply that the number of monomials of Q is

$$O\left(\binom{n-\delta n}{2dp\delta n}\right).$$

By constructing Q term-by-term (expanding each $P_{\delta n}(1-q(x_1,\ldots,x_{n-\delta n},a_1,\ldots,a_{\delta n})^{p-1})$ one-by-one, and adding them to a running sum, similar to [12, 29]), we may represent Q as a sum of $O\left(\binom{n-\delta n}{2dp\delta n}\right)$ monomials, constructed in $\mathrm{poly}(n)\cdot\binom{n-\delta n}{2dp\delta n}$ time. Letting $\delta=1/(6dp)$, the number of monomials of Q is less than $\binom{n}{n/3}\leq 1.9^n$. Applying the fast polynomial evaluation algorithm of Theorem 13, Q can be evaluated on all $2^{n-n/(6dp)}$ Boolean assignments in time $(1.9^n+2^{n-n/(6dp)})\cdot\mathrm{poly}(n)$ time. ◀

Therefore, for every *fixed* degree d and prime p, there is an $\varepsilon>0$ such that the relevant Sum-Product problem is in $2^{n-\varepsilon n}\cdot\mathrm{poly}(n)$ time. This immediately implies the lower bounds of Theorems 4 and 5. In particular, to prove 5 we apply Theorem 21. Fix an integer degree d, and let $c\geq 1$ be the universal constant (from Theorem 21) such that we need to solve Sum-Product for $\mathsf{MODp}\circ\mathsf{AND}_d\circ\mathsf{ANY}_c$ circuits. Converting to $\mathsf{SUM}\circ\mathsf{MODp}\circ\mathsf{AND}_{dc}$, Theorem 26 says that the Sum-Product problem can be solved in $2^{n-n/O(dc)}$ time (omitting low-order terms).

7 Conclusion

Applying old and new tools, we have established several strong new lower bounds for representing Boolean functions in different regimes. Among the most interesting open problems remaining, we find the Quadratic Uncertainty Principle (the conjecture that AND requires large $\mathbb{R}$-linear combinations of quadratic $\mathbb{F}_2$-polynomials) to be especially intriguing. Quadratic polynomials have special properties that higher degrees do not; for example, one can count the roots of a given quadratic $\mathbb{F}_p$-polynomial in *polynomial time* (see [54] for a recent application of this phenomenon). Therefore in some cases, our $2^{n-\varepsilon n}$-time algorithms become poly(n)-time algorithms. Intuitively, an extremely efficient counting algorithm *should* imply lower bounds for functions *in polynomial time* against linear combinations of quadratic $\mathbb{F}_2$-polynomials, perhaps even lower bounds against the AND function, but so far we have not yet been able to prove such bounds.

The Constant Degree Hypothesis?

A longstanding problem in circuit complexity – seemingly related to the Quadratic Uncertainty Principle – is the Constant Degree Hypothesis of Barrington, Straubing, and Therien [5]:

▶ **Hypothesis 27** (Constant Degree Hypothesis (CDH)). *For every constant $d \geq 1$ and primes p, q, there is an $\varepsilon > 0$ such that the* AND *function on n variables cannot be computed by* $\mathsf{MOD}_p \circ \mathsf{MOD}_q \circ \mathsf{AND}_d$ *circuits of $2^{\varepsilon n}$ size.*

The CDH is currently only known to be true for $d = 1$, and for $p = q$. Can the techniques of this paper say anything about such problems, even for the case of $d = 2$?

Split-and-List as a Lower Bound Technique?

As noted by a CCC reviewer, the algorithmic approaches applied in this paper (in particular, the "split-and-list" paradigm [18]) were essentially known in the literature, and yet they were already powerful enough to prove strong lower bounds against functions in NP. Is there a more direct method for proving circuit lower bounds that "corresponds" to the split-and-list paradigm, *without* having to go through a generic connection between SAT algorithms and circuit lower bounds?

Intuitively, the algorithmic split-and-list paradigm is related to communication complexity. In split-and-list, the variable space of an instance is "split" into smaller parts, and we find a global solution to the instance by "listing" partial solutions to the variables, and combining partial solutions together in some interesting way. This feels related to the situation where multiple parties hold parts of a global input, and they communicate to determine if the global input is a solution to some problem. Indeed, intuitive connections between the two have been successfully made in several papers, and articulated fairly strongly in [48, 37, 1].

However, there is a sense in which split-and-list seems more powerful. A good example is the algorithm for Subset-Sum: it splits the variables of the solution space into two parts, and uses the ability to *quickly and deterministically sort and search* the list of $2^{n/2}$ partial solutions to find a Subset-Sum faster. In contrast, deterministic communication between two parties holding $n/2$ bits each (with public knowledge of a Subset-Sum instance of n items) cannot always determine with low communication if their joint n-bit assignment is a solution to the instance. (When the weights of the instance are exponentially large, the communication problem becomes as hard as EQUALITY.)

References

1 Amir Abboud, Aviad Rubinstein, and R. Ryan Williams. Distributed PCP theorems for hardness of approximation in P. In *FOCS*, pages 25–36, 2017.

2 Josh Alman, Timothy M. Chan, and R. Ryan Williams. Polynomial representations of threshold functions and algorithmic applications. In *FOCS*, pages 467–476, 2016.

3 Raman Arora, Amitabh Basu, Poorya Mianjy, and Anirbit Mukherjee. Understanding deep neural networks with rectified linear units. *arXiv preprint arXiv:1611.01491*, 2016.

4 László Babai, Kristoffer Arnsfelt Hansen, Vladimir V. Podolskii, and Xiaoming Sun. Weights of exact threshold functions. In *Mathematical Foundations of Computer Science*, pages 66–77, 2010.

5 David A. Mix Barrington, Howard Straubing, and Denis Thérien. Non-uniform automata over groups. *Inf. Comput.*, 89(2):109–132, 1990.

6 Richard Beigel and Jun Tarui. On ACC. *Computational Complexity*, pages 350–366, 1994.

7 Eli Ben-Sasson and Emanuele Viola. Short PCPs with projection queries. In *ICALP*, pages 163–173, 2014.

8 Andreas Björklund, Thore Husfeldt, and Mikko Koivisto. Set partitioning via inclusion-exclusion. *SIAM J. Comput.*, 39(2):546–563, 2009.

9 Jean Bourgain. Estimation of certain exponential sums arising in complexity theory. *C.R. Acad. Sci. Paris Ser. I*, 340:627–631, 2005.

10 Jin-yi Cai, Frederic Green, and Thomas Thierauf. On the correlation of symmetric functions. *Mathematical Systems Theory*, 29(3):245–258, 1996.

11 Chris Calabro. A lower bound on the size of series-parallel graphs dense in long paths. *Electronic Colloquium on Computational Complexity (ECCC)*, 15(110), 2008.

12 Timothy M. Chan and Ryan Williams. Deterministic APSP, Orthogonal Vectors, and more: Quickly derandomizing Razborov-Smolensky. In *SODA*, pages 1246–1255, 2016.

13 Arkadev Chattopadhyay and Nikhil S. Mande. Weights at the bottom matter when the top is heavy. *Electronic Colloquium on Computational Complexity (ECCC)*, 24:83, 2017.

14 Ruiwen Chen, Rahul Santhanam, and Srikanth Srinivasan. Average-case lower bounds and satisfiability algorithms for small threshold circuits. In *CCC*, pages 1:1–1:35, 2016.

15 Amit Daniely. Depth separation for neural networks. In *Proceedings of COLT*, pages 690–696, 2017.

16 Ronen Eldan and Ohad Shamir. The power of depth for feedforward neural networks. In *Proceedings of COLT*, pages 907–940, 2016.

17 Yuval Filmus, Hamed Hatami, Steven Heilman, Elchanan Mossel, Ryan O'Donnell, Sushant Sachdeva, Andrew Wan, and Karl Wimmer. Real Analysis in Computer Science: A collection of open problems, Simons Institute, 2014. URL: `https://simons.berkeley.edu/sites/default/files/openprobsmerged.pdf`.

18 Fedor V. Fomin and Dieter Kratsch. *Exact Exponential Algorithms*. Springer, 2010.

19 Anna Gál and Vladimir Trifonov. On the correlation between parity and modular polynomials. *Theory Comput. Syst.*, 50(3):516–536, 2012.

20 Frederic Green. The correlation between parity and quadratic polynomials mod 3. *Journal of Computer and System Sciences*, 69(1):28–44, 2004.

21 Jacques Hadamard. Résolution d'une question relative aux déterminants. *Bull. Sci. Math.*, 17:30–31, 1893.

22 András Hajnal, Wolfgang Maass, Pavel Pudlák, Mario Szegedy, and György Turán. Threshold circuits of bounded depth. *J. Comput. Syst. Sci.*, 46(2):129–154, 1993.

23 Kristoffer Arnsfelt Hansen and Vladimir V Podolskii. Exact threshold circuits. In *CCC*, pages 270–279, 2010.

24 Johan Håstad and Mikael Goldmann. On the power of small-depth threshold circuits. *Computational Complexity*, 1:113–129, 1991.

25 Hamed Hatami, Pooya Hatami, and Shachar Lovett. Higher-order Fourier analysis and applications. Manuscript, 2016. URL: `https://cseweb.ucsd.edu/~slovett/files/survey-higher_order_fourier.pdf`.

26 Ellis Horowitz and Sartaj Sahni. Computing partitions with applications to the knapsack problem. *JACM*, 21(2):277–292, 1974.

27 Hamid Jahanjou, Eric Miles, and Emanuele Viola. Local reductions. In *Proceedings of ICALP*, pages 749–760, 2015.

28 Daniel M. Kane and Ryan Williams. Super-linear gate and super-quadratic wire lower bounds for depth-two and depth-three threshold circuits. In *STOC*, pages 633–643, 2016.

29 Daniel Lokshtanov, Ramamohan Paturi, Suguru Tamaki, R. Ryan Williams, and Huacheng Yu. Beating brute force for systems of polynomial equations over finite fields. In *SODA*, pages 2190–2202, 2017.

30 Shachar Lovett. Personal communication, 2017.

31 Wolfgang Maass. Bounds for the computational power and learning complexity of analog neural nets. *SIAM Journal on Computing*, 26(3):708–732, 1997.

32 Peter Bro Miltersen, N. V. Vinodchandran, and Osamu Watanabe. Super-polynomial versus half-exponential circuit size in the exponential hierarchy. In *COCOON, Springer LNCS 1627*, pages 210–220, 1999.

33 Anirbit Mukherjee and Amitabh Basu. Lower bounds over Boolean inputs for deep neural networks with ReLU gates. *ArXiv e-prints*, 2017. `arXiv:1711.03073`.

34 S. Muroga, I. Toda, and S. Takasu. Theory of majority decision elements. *Journal of the Franklin Institute*, 271:376–418, 1961.

35 Cody Murray and Ryan Williams. Circuit lower bounds for nondeterministic quasi-polytime: An easy witness lemma for NP and NQP. *Electronic Colloquium on Computational Complexity (ECCC)*, TR17-188, 2017.

36 Noam Nisan. The communication complexity of threshold gates. In *Proceedings of "Combinatorics, Paul Erdos is Eighty"*, pages 301–315, 1994.

37 Mihai Pătraşcu and Ryan Williams. On the possibility of faster sat algorithms. In *SODA*, pages 1065–1075, 2010.

38 Vwani P. Roychowdhury, Alon Orlitsky, and Kai-Yeung Siu. Lower bounds on threshold and related circuits via communication complexity. *IEEE Transactions on Information Theory*, 40(2):467–474, 1994.

39 Itay Safran and Ohad Shamir. Depth-width tradeoffs in approximating natural functions with neural networks. In *International Conference on Machine Learning*, pages 2979–2987, 2017.

40 Rahul Santhanam. Circuit lower bounds for Merlin–Arthur classes. *SIAM J. Comput.*, 39(3):1038–1061, 2009.

41 Rahul Santhanam and Ryan Williams. On medium-uniformity and circuit lower bounds. In *IEEE Conf. Computational Complexity*, pages 15–23, 2013.

42 Joel Seiferas, Michael Fischer, and Albert Meyer. Separating nondeterministic time complexity classes. *Journal of the ACM*, 25(1):146–167, jan 1978.

43 Suguru Tamaki. A satisfiability algorithm for depth two circuits with a sub-quadratic number of symmetric and threshold gates. *Electronic Colloquium on Computational Complexity (ECCC)*, 23:100, 2016.

44 Matus Telgarsky. benefits of depth in neural networks. In *Proceedings of COLT*, pages 1517–1539, 2016.

45 Roei Tell. Proving that prBPP=prP is as hard as "almost" proving that P $\neq$ NP. *Electronic Colloquium on Computational Complexity (ECCC)*, 18(3), 2018.

46 S. Toda. PP is as hard as the polynomial-time hierarchy. *sicomp*, 20(5):865–877, 1991.

47 L. G. Valiant. Graph-theoretic arguments in low-level complexity. In J. Gruska, editor, *MFCS*, volume 53 of *LNCS*, pages 162–176, Tatranská Lomnica, Czechoslovakia, sep 1977. Springer.

48 Virginia Vassilevska Williams and Ryan Williams. Finding, minimizing, and counting weighted subgraphs. *SIAM Journal on Computing*, 42(3):831–854, 2013.

49 Emanuele Viola. Guest column: correlation bounds for polynomials over {0, 1}. *SIGACT News*, 40(1):27–44, 2009.

50 Ryan Williams. A casual tour around a circuit complexity bound. *SIGACT News*, 42(3):54–76, 2011.

51 Ryan Williams. Improving exhaustive search implies superpolynomial lower bounds. *sicomp*, 42(3):1218–1244, 2013.

52 Ryan Williams. New algorithms and lower bounds for circuits with linear threshold gates. In *STOC*, pages 194–202, 2014.

53 Ryan Williams. Nonuniform ACC circuit lower bounds. *JACM*, 61(1):2, 2014.

54 Ryan Williams. Counting solutions to polynomial systems via reductions. In Raimund Seidel, editor, *1st Symposium on Simplicity in Algorithms (SOSA 2018)*, pages 6:1–6:15, 2018.

55 R. O. Winder. *Threshold Logic*. PhD thesis, Princeton University, 1962. Preliminary version in FOCS'60.

56 Stanislav Žák. A Turing machine time hierarchy. *Theoretical Computer Science*, 26(3):327–333, 1983.

A Linear Lower Bound for AND With Sums of Quadratic Polynomials

For reference, we report a folklore $\Omega(n)$ lower bound on representing AND with linear combinations of quadratic $\mathbb{F}_2$-polynomials (recall it is conjectured that the sparsity lower bound is $2^{\Omega(n)}$). The below proof was communicated to us by Shachar Lovett.

▶ **Theorem 28** (Lovett [30]). *The* AND *function on n inputs does not have* $\mathsf{SUM} \circ \mathsf{MOD2} \circ \mathsf{AND}_2$ *circuits of sparsity less than $n/2$.*

Proof. Let $f : \{0,1\}^n \to \{0,1\}$ be the NOR function (which by DeMorgan's laws has the same sparsity as AND). Suppose we can write

$$f(x) = \sum_{i=1}^{s} \alpha_i (-1)^{q_i(x)},$$

where the $q_i(x)$ are quadratic $\mathbb{F}_2$-polynomials, and all $\alpha_i \in \mathbb{R}$. Note that without loss of generality we may assume $q_i(0) = 0$ for all i (if $q_i(0) = 1$, then replacing α_i by $-\alpha_i$ and $q_i(x)$ by $q_i(x) + 1$ yields an equivalent expression). If $s < n/2$, then by the Chevalley–Warning theorem, the number of common roots of $\{q_1, ..., q_r\}$ is divisible by 2. But then there is another common root $x^\star$, so $f(0) = f(x^\star)$, contradicting the definition of NOR. ◀